Tivoli® Composite Application Manager for Transactions

Version 7.1
for AIX, Linux, Solaris, Windows, and z/OS

Transaction Tracking API User's Guide

SC23-9755-00

Note
Before using this information and the product it supports, read the information in "Notices" on page 71.

This edition applies to version 7, release 1 of IBM Tivoli Composite Application Manager for Transactions (product number 5724-S79) and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Figures v

Tables vii

About this publication ix
Publications ix
Documentation library ix
Prerequisite publications x
Accessing terminology online x
Accessing publications online x
Ordering publications x
Accessibility xi
Tivoli technical training xi
Support information xi
Conventions used in this guide xi
Typeface conventions xi
Operating system-dependent variables and paths xii

Chapter 1. Introduction 1

Chapter 2. Before you start 3

Chapter 3. Preparing your environment 5

Chapter 4. Getting started 7
Introduction 7
Program requirements and include files 8
Compiling, linking, and executing with Transaction Tracking API 8
Error handling 10
Error logging and debugging 10

Chapter 5. How to build an event . . . 13
Event types 13
Event type examples 15
Linking and stitching 15
Transaction Instance IDs 18
Context information 19
Blocking events 21
Example: blocking events 21
Platform specific issues 22

Chapter 6. High Level Language reference 23
Functions 23
C types and structures 27

Chapter 7. Java reference 33

Chapter 8. High Level Assembler Reference 35
HLASM Macro: CYTADFV 35
HLASM Macro: CYTAINIT 37
HLASM Macro: CYTANV 38
HLASM Macro: CYTATOK 39
HLASM Macro: CYTATRAK 40

Appendix A. Transport address format 45

Appendix B. Return codes 47

Appendix C. Samples 49

Appendix D. kto_stitching file 63

Appendix E. Transaction Collector Context Mask 67

Appendix F. Accessibility 69

Notices 71
Trademarks 73

Glossary 75

Index 83

Figures

1. Synchronous transaction 15
2. Contextual information in a transaction 20
3. Partially asynchronous transaction 22
4. Transaction Collector Configuration dialog box 68

Tables

1. Transaction Tracking API support 3
2. Logging configuration environment variables 10
3. Event components 13
4. Event types 14
5. Samples in the SCYTSAMP library 49
6. Field matching 64

About this publication

This guide provides information about instrumenting applications to provide tracking information for ITCAM for Transaction Tracking.

Intended audience

This guide is for system administrators who enable applications to send events to ITCAM for Transaction Tracking.

Use the information in the IBM® Tivoli® Composite Application Manager for Transactions *User's Guide* and *Administrator's Guide* together with the *IBM Tivoli Monitoring User's Guide* to monitor and manage the performance of your systems.

Publications

This section lists publications relevant to the use of the IBM Tivoli Composite Application Manager for Transactions. It also describes how to access Tivoli publications online and how to order Tivoli publications.

Documentation library

The following documents are available in the IBM Tivoli Composite Application Manager for Transactions library:

- *IBM Tivoli Composite Application Manager for Transactions Administrator's Guide*

  This guide provides information about configuring elements of IBM Tivoli Composite Application Manager for Transactions.
- *IBM Tivoli Composite Application Manager for Transactions Installation and Configuration Guide*

  This guide provides information about installing and configuring elements of IBM Tivoli Composite Application Manager for Transactions.
- *IBM Tivoli Composite Application Manager for Transactions Quick Start Guide*

  This guide provides a brief overview of IBM Tivoli Composite Application Manager for Transactions.
- *IBM Tivoli Composite Application Manager for Transactions Troubleshooting Guide*

  This guide provides information about using all elements of IBM Tivoli Composite Application Manager for Transactions.
- *IBM Tivoli Composite Application Manager for Transactions Transaction Tracking API User's Guide*

  This guide provides information about the Transaction Tracking API.
- *IBM Tivoli Composite Application Manager for Transactions User's Guide*

  This guide provides information about the GUI for all elements of IBM Tivoli Composite Application Manager for Transactions.
- *IBM Tivoli Composite Application Manager for Transactions Installation and Configuration Guide for z/OS*

  This guide provides information about using IBM Tivoli Composite Application Manager for Transactions on z/OS.

Prerequisite publications

To use the information in this guide effectively, you must have some knowledge of IBM Tivoli Monitoring products that you can obtain from the following documentation:

- *IBM Tivoli Monitoring Administrator's Guide*, version 6.2 Fix Pack 1
- *IBM Tivoli Monitoring Installation and Setup Guide*, version 6.2 Fix Pack 1
- *IBM Tivoli Monitoring User's Guide*, version 6.2 Fix Pack 1

Accessing terminology online

The *Tivoli Software Glossary* includes definitions for many of the technical terms related to Tivoli software. The *Tivoli Software Glossary* is available at the following Tivoli software library Web site:

http://publib.boulder.ibm.com/tividd/glossary/tivoliglossarymst.htm

The IBM Terminology Web site consolidates the terminology from IBM product libraries in one convenient location. You can access the Terminology Web site at the following Web address:

http://www.ibm.com/software/globalization/terminology

Accessing publications online

IBM posts publications for this and all other Tivoli products, as they become available and whenever they are updated, to the Tivoli software information center Web site.

Access the Tivoli software information center by going to the Tivoli software library at the following Web address:

http://publib.boulder.ibm.com/infocenter/tivihelp/v3r1/index.jsp.

Ordering publications

You can order many Tivoli publications online at the following Web site:

http://www.elink.ibmlink.ibm.com/publications/servlet/pbi.wss

You can also order by telephone by calling one of these numbers:

- In the United States: 800-879-2755
- In Canada: 800-426-4968

In other countries, contact your software account representative to order Tivoli publications. To locate the telephone number of your local representative:

1. Go to http://www.ibm.com/planetwide/
2. In the alphabetical list, select the letter for your country and then click the name of your country. A list of numbers for your local representatives is displayed.

Accessibility

Accessibility features help users with a physical disability, such as restricted mobility or limited vision, to use software products. With this product, you can use assistive technologies to hear and navigate the interface. You can also use the keyboard instead of the mouse to operate most features of the graphical user interface.

For additional information, see Accessibility Appendix F, "Accessibility," on page 69.

Tivoli technical training

For information about Tivoli technical training, refer to the following IBM Tivoli Education Web site:

http://www.ibm.com/software/tivoli/education

Support information

If you have a problem with your IBM software, you want to resolve it quickly.

IBM provides the following ways for you to obtain the support you need:

Online

Go to the IBM Software Support site at http://www.ibm.com/software/support/probsub.html and follow the instructions.

IBM Support Assistant

The IBM Support Assistant (ISA) is a free local software serviceability workbench that helps you resolve questions and problems with IBM software products. The ISA provides quick access to support-related information and serviceability tools for problem determination. To install the ISA software, go to http://www.ibm.com/software/support/isa.

Conventions used in this guide

This guide uses several conventions for operating system-dependent commands and paths, special terms, actions, and user interface controls.

Typeface conventions

This guide uses the following typeface conventions:

Bold

- Lowercase commands and mixed case commands that are otherwise difficult to distinguish from surrounding text
- Interface controls (check boxes, push buttons, radio buttons, spin buttons, fields, folders, icons, list boxes, items inside list boxes, multicolumn lists, containers, menu choices, menu names, tabs, property sheets), labels (such as **Tip:**, and **Operating system considerations**:)
- Keywords and parameters in text

Italic

- Words defined in text
- Emphasis of words
- New terms in text (except in a definition list)
- Variables and values you must provide

Monospace

- Examples and code examples
- File names, programming keywords, and other elements that are difficult to distinguish from surrounding text
- Message text and prompts addressed to the user
- Text that the user must type
- Values for arguments or command options

Operating system-dependent variables and paths

This guide uses the UNIX® system convention for specifying environment variables and for directory notation.

When using the Windows® command line, replace **$***variable* with **%***variable***%** for environment variables and replace each forward slash (**/**) with a backslash (\) in directory paths. The names of environment variables are not always the same in the Windows and UNIX environments. For example, %TEMP% in Windows environments is equivalent to $TMPDIR in UNIX environments.

Note: If you are using the bash shell on a Windows system, you can use the UNIX conventions.

Chapter 1. Introduction

ITCAM for Transaction Tracking is a solution for tracking transactions across applications and networks. It provides an upgrade path from ITCAM for Response Time Tracking, and consolidates domain-specific tracking technologies.

ITCAM for Transaction Tracking tracks applications by accepting information from applications, monitoring software and other sources that specify a point in the life of an application. Each piece of information is an *event*. ITCAM for Transaction Tracking Data collectors such as ITCAM for MQ Tracking automatically send these events to ITCAM for Transaction Tracking.

The Transaction Tracking Application Programming Interface (Transaction Tracking API), provides developers with a means of sending their own events and providing tracking information to ITCAM for Transaction Tracking 7.1. This allows developers to enhance tracking beyond that provided by Transactions Data Collectors.

This guide describes how to use the Transaction Tracking API. It describes how to construct events and how to send these events to Transactions for processing.

Chapter 2. Before you start

Transaction Tracking API is supported on a range of operating systems and architectures. It supports a number of programming languages.

Platform support

Table 1 lists the supported operating systems and hardware architectures on which you can use Transaction Tracking API.

Table 1. Transaction Tracking API support

Operating system	Architecture
AIX® 5.3	iSeries® / pSeries®
AIX 5L™ 5.4	iSeries / System p™
Novell Linux® Desktop 9	x86 32-bit 64-bit
Red Hat Desktop 4.0	x86 32-bit 64-bit
Red Hat Desktop 5.0	x86 32-bit 64-bit
RHEL 4	x86 32-bit 64-bit
RHEL 5	x86 32-bit 64-bit
SLED 10	x86 32-bit 64-bit
Solaris 9	SPARC
Solaris 10	SPARC
SUSE (SLES) 9.0	x86 32-bit 64-bit
SUSE (SLES) 10.0	x86 32-bit 64-bit
Windows Longhorn Standard Edition	x86 32-bit 64-bit
Windows Longhorn Datacenter Edition	x86 32-bit 64-bit
Windows Longhorn Enterprise Edition	x86 32-bit 64-bit
Windows Server 2003 Enterprise Edition	x86 32-bit
IBM z/OS®	IBM z/Series

Supported programming languages

The Transaction Tracking API supports the following programming languages:

- C
- C++
- Enterprise COBOL (z/OS only)
- Enterprise PL/I (z/OS only)
- IBM High Level Assembler (HLASM - z/OS only)
- Java™ 1.4 and 1.5

The following program environments are supported on z/OS:

- C, C++ and Java applications running in 64 bit mode on z/OS
- C and C++ XPLINK and non-XPLINK programs on z/OS

- C and C++ programs statically and dynamically linking Transaction Tracking API
- COBOL and PL/I programs statically linking Transaction Tracking API

Chapter 3. Preparing your environment

The way in which you prepare your environment is dependent on whether it is a distributed or z/OS environment.

Distributed environments

You may unpack the Transaction Tracking API Software Development Kit (Transaction Tracking API SDK) anywhere on your system. Depending on the target platform, the Transaction Tracking API SDK is packaged as a single *zip* or *tar* archive. This archive contains everything required to instrument an application. However, it does not include any other related components, such as the Transaction Collector.

The files contained in the Transaction Tracking API SDK are:

```
include/
  ttapi.h
lib/
  ttapi4j.jar
  ttapi.lib, ttapi.dll, pthread.dll (Windows)
  libttapi.so (UNIX - suffixes vary by platform)
  kbb.dll on Windows
  libkbb.so on UNIX
```

z/OS environments

For z/OS systems, the Transaction Tracking API SDK is installed as part of the Transactions Base install. However:

- C programmers on UNIX Systems Services (USS) may wish to copy the SCYTSAMP member CYTAPI to a USS directory of their choice, renaming it to cytapi.h. This file holds all C and C++ includes necessary to use the Transaction Tracking API.
- Java programmers must ensure that the Transactions JAR files are in the Java classpath, and external links to the Transactions JNI modules are in the Java libpath. See the *IBM Tivoli Composite Application Manager for Transactions Installation and Configuration Guide* for more information.

Chapter 4. Getting started

Use this information to help you create and send events.

Introduction

To send a transactions event you must complete these steps.

1. Initialize Transaction Tracking API using the `init` function.
2. Construct the event.
3. Send the event using the `track` function.
4. Shut down the Transaction Tracking API using the `shutdown` function.

Initialize Transaction Tracking API

The `init` function only needs to be called once per process. This function sets up the Transaction Tracking API environment, and populates the Configuration Block with information required by all other Transaction Tracking API functions. The Configuration Block must not be changed once `init` has been called. `init` must be called before any other Transaction Tracking API function. However, `TT_check_version` should be called before `init`, otherwise it has no effect.

Callers of the init function must allocate an area for the Configuration Block, and populate the servername field – the destination where events are to be sent. This must be in the format specified in Appendix A, "Transport address format," on page 45.

Java callers use the `ServerFactory.getServer` class, however this performs the same functions as the `init` function. Non-z/OS C and C++ users may choose to use the `check_version` function to check the header versions.

Create the event

An event block must be allocated and completed by the caller. Chapter 5, "How to build an event," on page 13 describes the event format in detail, and shows how to code an event block.

Java users construct events by calling the `createEvent` method of a `ttapi4j.Server` object.

Send the event

The `track` function (native) and `ttapi4j.Server.track` method (Java) are used to send the event. Note that calling `track` does not modify the contents of the event constructed. Users calling `track` multiple times do not have to recreate the entire event, but can reuse the existing event – replacing only the individual fields required.

Shutdown the Transaction Tracking API

The Transaction Tracking API should be shutdown when it is no longer needed to track events.

Java users should invoke the close method of the Server object.

Note: The shutdown function is not required for z/OS.

Program requirements and include files

Before using the Transaction Tracking API, you must first provide standard preamble statements and include files in your code.

C/C++

You must include the Transaction Tracking API include file. For example:

```
#include <ttapi.h>
```

For z/OS users:

```
#include <cytapi.h>
```

COBOL

You must copy the CYTABCON constants copybook into the Data-Division of your Working Storage Section. For example:

```
 DATA DIVISION.
       Working-Storage Section.
       COPY CYTABCON.
```

PL/I

You must include the supplied event block structure. For example:

```
%include CYTAPEVT;
```

Compiling, linking, and executing with Transaction Tracking API

Compiling

C/C++

To compile C or C++ programs against the Transaction Tracking API library, add the `include` directory found in the Transaction Tracking API SDK or SCYTSAMP dataset to the compiler's preprocessor include path.

For example, if compiling with Microsoft® Visual C 7.1 and the SDK is installed in C:\TTAPI:

```
cl /I C:\TTAPI\include <custom-flags> /c <source-filename>
```

Java

To compile a Java program, ensure that the Transactions cytapi4j.jar (z/OS) or ttapi4j.jar (non-z/OS platforms) JAR file is in the Java classpath. This can be achieved by adding the file's absolute path to the `CLASSPATH` environment variable, or by specifying it on the command line using the `-classpath` flag.

COBOL and PL/I

To compile COBOL or PL/I programs, ensure the SCYTSAMP library is in the compiler's SYSLIB DD concatenation.

High Level Assembler

To assemble HLASM programs, ensure the SCYTSAMP library is in the assembler's SYSLIB DD concatenation.

Linking on distributed platforms

To link the resultant objects into an application, link against the libttapi library provided with the Transaction Tracking API SDK. For example, if linking with Microsoft Visual C 7.1 and the SDK is installed in C:\TTAPI:

```
link /libpath: C:\TTAPI\lib <custom-flags> <object-files> ttapi.lib
```

Binding on z/OS

The Transaction Tracking API programs are stored as DLLs in the SCYTLOAD library. When binding programs with Transaction Tracking API on z/OS:

- If a C or C++ program is dynamically calling Transaction Tracking API:
  - If compiling in batch, include the SCYTSAMP member CYTASIDE in the SYSLIN DD.
  - If compiling in UNIX Systems Services, copy the SCYTSAMP member CYTASIDE to an HFS directory, renaming it to cytaside.x. Include cytaside.x when binding your program. For example: `c89 –W'l,dll' pgm1.o cytaside.x`
- Otherwise ensure the SCYTLOAD library is included in the binder search path. For example, adding SCYTLOAD to the binder SYSLIB DD if binding in batch.

Running with Transaction Tracking API

C/C++

If running on distributed (non-z/OS) platforms, ensure the lib directory of the Transaction Tracking API SDK is included in the runtime library search path.

For example, on Linux you must modify the LD_LIBRARY_PATH environment variable to include the directory.

If running on z/OS, ensure the SCYTLOAD library is in the z/OS linklist concatenation, and that external links to the Transactions JNI modules are defined in the Java libpath. See *IBM Tivoli Composite Application Manager for Transactions Installation and Configuration Guide* for more information.

COBOL and PL/I

If running on z/OS, ensure the SCYTLOAD library is in the z/OS linklist concatenation.

Java

If running on distributed (non-z/OS) platforms, ensure the lib directory of the Transaction Tracking API SDK is included in the runtime library search path.

For example, on Linux you must modify the `LD_LIBRARY_PATH` environment variable to include the directory.

If running on z/OS, ensure the SCYTLOAD library is in the z/OS linklist concatenation.

In both cases, ensure the cytapi4j.jar (z/OS) or ttapi4j.jar (non-z/OS) JAR files are in the Java classpath.

Error handling

Transaction Tracking API functions that fail return an error code, which must be checked to determine whether the Transaction Tracking API function has succeeded.

If a function succeeds it returns TT_SUCCESS (zero). If it fails, it returns a non-zero error code, as described Appendix B, "Return codes," on page 47. Additionally, Transaction Tracking API provides logging facilities to further isolate the problem. For normal operation, this logging is disabled.

Invalid events

The `track` function validates all events and returns a code, as described in Appendix B, "Return codes," on page 47, which specifies in detail the invalid field or value.

Undefined behavior

There are certain error conditions that the Transaction Tracking API cannot detect. For example:

- Passing one Configuration Blocks to `init`, and a different block to other functions.
- Modifying the Configuration Block after `init` has been called.
- Incorrect length value specified.
- Configuration Block, event or name/value pairs not initialized to nulls before use.

In the preceding cases, Transaction Tracking API processing is unpredictable.

Error logging and debugging

In addition to returning error codes from Transaction Tracking API functions, Transaction Tracking API logs error and debug messages at significant points in the process of initializing, shutting down, sending events to a Transaction Collector, and various states in between. In general, this logging will not be of interest to users - it will usually be turned on to provide support professionals with enough information to help isolate an error or misuse of the API.

On platforms other than z/OS, Transaction Tracking API uses the IBM Tivoli Monitoring standard `RAS1` logging. On z/OS, log information is written to standard output. You can control the amount of logging produced by the `RAS1` logger by configuring the environment variables listed in Table 2.

Table 2. Logging configuration environment variables

Environment variable	Description
`KBB_RAS1=ALL`	Enable logging of all messages.
`KBB_RAS1=ERROR`	Enable logging of error messages.
`KBB_RAS1=`	Disable all message logging. This is the default.
`KBB_RAS1_LOG=`	Log to standard output.

Table 2. Logging configuration environment variables (continued)

Environment variable	Description
KBB_RAS1_LOG=...	Set the log file name and other parameters. See the format example that follows this table..
KBB_VARPREFIX=%	Set the prefix for variables specified in KBB_RAS1_LOG.

KBB_RAS1_LOG has the following format:

```
KBB_RAS1_LOG=filename [INVENTORY=inventory_filename]
    [COUNT=count]
    [LIMIT=limit]
    [PRESERVE=preserve]
    [MAXFILES=maxfiles]
```

The settings for KBB_RAS1_LOG are:

count Maximum number of log files to create in one invocation of the application.

inventory_filename
A file in which to record the history of log files across invocations of the application.

limit Maximum size per log file.

maxfiles
Maximum number of log files to create in any number of invocations of the application. This value takes effect only when *inventory* is specified.

preserve
Number of log files to preserve when log files wrap over *count*.

Chapter 5. How to build an event

Instrumenting an application requires you to create events that indicate the flow of a transaction.

The Transaction Tracking API comes in multiple forms:

- High Level Language (HLL) package – for C, C++, COBOL and PL/I programs.
  See "C types and structures" on page 27 for information specific to instrumenting C/C++ applications.
- TTAPI4J – wrapper for Java applications.
  See Chapter 7, "Java reference," on page 33 for information specific to instrumenting Java applications.
- z/OS macros for HLASM callers.
  See Chapter 8, "High Level Assembler Reference," on page 35.

Detailed information on each of these forms is given in the appropriate reference chapter. Complete examples for all languages are provided in the Appendixes.

Transaction Tracking API events contain the major components described in Table 3.

Table 3. Event components

Component	Description
Event type	The type of the event, for example outbound or inbound.
Instance ID	Information specific to the event's enclosing transaction instance.
Horizontal ID	Information used to correlate events, where the events occur in separate processes, potentially on separate machines.
Vertical ID	Information used to correlate events, where the events occur in the same process.
Horizontal context	Information used to aggregate events across processes.
Vertical context	Information used to aggregate events within a process.
Blocked Status	An attribute that indicates whether or not an event is related to a synchronous interaction in its transaction.

These components are described in detail in the following sections.

Event types

Every event sent by Transaction Tracking API has an associated type that is used in event correlation.

The event types are described in Table 4.

Table 4. Event types

Event type	Description
STARTED	The beginning of a transaction. No events in a transaction may come before STARTED. The STARTED event type is used as the lower bound by the correlation system when searching for related events.
FINISHED	The end of a transaction. No events in a transaction may come after FINISHED. The FINISHED event type is used as the upper bound by the correlation system when searching for related events.
INBOUND	A message has been received. These events are typically used to correlate cross-process interactions.
OUTBOUND	A message has been sent. These events are typically used to correlate cross-process interactions.
HERE	Usually indicates the blocking or unblocking of an asynchronous transaction. May also be used in situations where there is not enough context to determine whether an event is the result of an outgoing or incoming message.
STARTED_INBOUND	Combination of STARTED and INBOUND events used to reduce the number of events produced. STARTED_INBOUND events are used both as a STARTED lower bound, and for INBOUND correlation.
OUTBOUND_FINISHED	Combination of OUTBOUND and FINISHED events used to reduce the number of events produced. OUTBOUND_FINISHED events are used both as a FINISHED upper bound, and for OUTBOUND correlation.
INBOUND_FINISHED	Combination of INBOUND and FINISHED events used to reduce the number of events produced. INBOUND_FINISHED events are used both as a FINISHED upper bound, and for INBOUND correlation.

The event is set in the event block `Type` field. For example:

Java

```
event.setType(Event.Type.OUTBOUND);
```

C/C++

```
event.type = TT_OUTBOUND_EVENT;
```

COBOL

```
MOVE CYTA-STARTED TO CYTA-E-TYPE.
```

PL/I

```
cytaetyp = cytaesta;
```

HLASM

```
CYTATRAK STARTED,
```

Event type examples

The Transaction Tracking API can be used to track both synchronous and asynchronous transactions.

Synchronous transactions

The simple example shown in Figure 1 demonstrates event type usage in a client application making a synchronous request to a server application.

The dashed line in Figure 1 illustrates the flow of the transaction from start to finish.

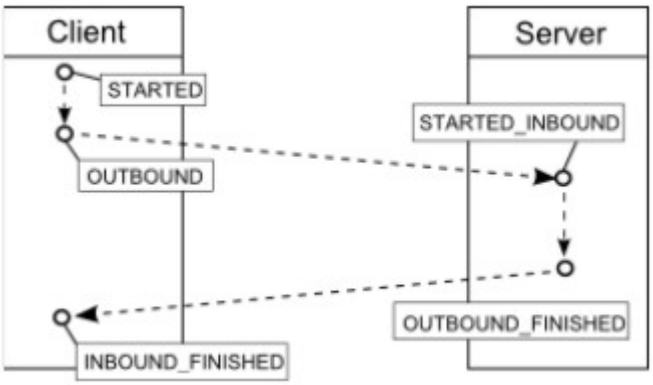


Figure 1. Synchronous transaction

Transactions typically start with a `STARTED` event, unless the transaction is known to have started as the result of an inbound message, in which case a `STARTED_INBOUND` event type is used. In the example, the overall transaction begins with a `STARTED` event, and the subtransaction in the server process begins with a `STARTED_INBOUND`.

Transactions typically terminate with an `INBOUND_FINISHED` or `OUTBOUND_FINISHED` event, because a transaction usually terminates upon receipt of a reply to a prior request. If the transaction terminates because of some other condition, for example if the request from the client to the server times out, indicate this with a `FINISHED` event.

Asynchronous transactions

See “Blocking events” on page 21 for an example of event-type usage in asynchronous transactions.

Linking and stitching

One of the most important elements of the Transaction Tracking API event is the association ID, which is composed of linking and stitching IDs. Linking and stitching IDs are used to determine the relationships between events.

For example, if an outbound and an inbound event are related to a particular interprocess interaction, then they must both contain some identical information so that they can be matched against each other. Each event may have either or both

horizontal and *vertical* association IDs. Typically, the horizontal ID correlates events across processes, and the vertical ID correlate events within a single process.

Note: The term *technology domain* is introduced in this section. This term refers to a (potential) Transaction Tracking API event source, such as ARM, MQ, ITCAM for SOA, or a custom application. Each domain is expected to provide enough information to correlate Transaction Tracking API events.

Linking and stitching IDs

Linking IDs identify interactions within a single technology domain. For example, where a domain, such as ARM, tracks transactions by passing tokens along, that token, or some part of it, might be used as the linking ID. The linking ID will not match any other domain, but it allows events within the ARM domain to be correlated. You must provide Transaction Tracking API with one and only one linking ID.

Stitching IDs identify interactions between technology domains, both within and across processes. In-process interactions occur only if there are two sources of tracking information within that process. If a process lies at the edge of two technology domains, for example ARM and MQ, then it is possible that the process will produce events for both domains.

Linking and stitching IDs are opaque to the Transaction Tracking API; they carry no special meaning, and have no particular formatting constraints beyond their size limitations. The event correlation system performs a simple byte-array comparison for equality.

Linking and stitching IDs must be globally unique for each interaction, from the first STARTED event to the last FINISHED event.

Tip: You can improve the uniqueness of linking IDs in custom applications by adding a prefix or suffix to all linking IDs generated by your application. In doing this, you will achieve the same effect as setting the caller type to some value unique to your application, provided that the prefix or suffix you choose is not used by another application.

Stitching IDs

While Transaction Tracking API provides an interface for providing arbitrary stitching IDs, they are useless if there is no commonality between each technology domain. Horizontal stitching IDs generally must be provided on a pair-by-pair basis - that is for each pair of technology domains. The developers instrumenting these domains will have to communicate to determine common stitching IDs. Transaction Tracking API events may contain multiple stitching IDs. For any two events, if a stitching ID of one event is equal to the stitching ID with the same name of the other event, then some interaction is assumed to have occurred between the two events. The Transaction Collector kto_stitching.xml file defines how this stitching occurs. See Appendix D, “kto_stitching file,” on page 63 for further information on this file.

Vertical stitching can generally be accomplished by using the thread ID of the thread in which the transaction event occurred. This enables the correlation system to correlate events from one transaction that are interleaved with events from another transaction in another thread. This depends on the structure of the

instrumented application, and on a single thread being used to service a transaction.

Link types

All links must have a type (CALL TYPE ID) that is an integer value and can be any number 0 to 255. Events have a default caller type of ANY - this caller type will only be matched against other events with the ANY caller type, and may be used instead of a domain-specific caller type. However, use values from the range 200-255 rather than ANY.

Currently defined values are:

- 0 ANY
- 1 GPS
- 2 ARM
- 3 WSA
- 4 CICS Transaction Gateway
- 5 Websphere MQ
- 6 SOA
- 7 Web Resource
- 8 CICS
- 9 IMS
- 10-199 Reserved for IBM use
- 200-255 Available to users

Examples

Vertical linking

This example uses C/C++ to link events using the current thread ID:

```
pthread_t current_thread;
char thread_id[sizeof(pthread_t)];
struct tt_event_t event;
 /* Set the event's vertical link ID to the current thread ID. */
current_thread = pthread_self();
memcpy(thread_id, &current_thread, sizeof(pthread_t));
event.vertical_id.link_id = thread_id;
event.vertical_id.caller_type = TT_ARM_CALLER;
event.vertical_id.link_id_size = sizeof(pthread_t);
```

Horizontal linking

This example uses C/C++ to link events using a token embedded in a message sent by the instrumented application. Because both applications are members of the same technology domain they are capable of communicating this way. That is, the server knows how to decode the message so that it can extract the value of the horizontal link ID:

```
/* Create the token that will be sent with the message */
uint16_t token_size = 0;
char *token = create_token(&token_size);
 /* Set the event's horizontal link ID to the message's token value. */
event.horizontal_id.link_id = token;
event.horizontal_id.caller_type = TT_IMS_CALLER;
event.horizontal_id.link_id_size = token_size;
```

Stitching

Using horizontal and vertical stitching IDs is similar to using horizontal and vertical linking IDs. Below are some simple examples of how to configure stitching IDs.

Java:

```
Event event = server.createEvent();
event.getHorizontalID().getStitchingIDs().put("name", "value");
event.getVerticalID().getStitchingIDs().put("name", "value");
```

C/C++:

```
tt_event_t event;
tt_values_list_t horizontal_stitching_ids;
tt_values_list_t vertical_stitching_ids;
 horizontal_stitching_ids.name  = "name";
horizontal_stitching_ids.value = "value";
horizontal_stitching_ids.size  = sizeof("name") - 1;
horizontal_stitching_ids.next  = 0;
event.horizontal_id.stitch_ids = &horizontal_stitching_ids;
vertical_stitching_ids.name  = "name";
vertical_stitching_ids.value = "value";
vertical_stitching_ids.size  = sizeof("name") - 1;
vertical_stitching_ids.next  = 0;
event.vertical_id.stitch_ids = &vertical_stitching_ids;
```

Transaction Instance IDs

Transaction Tracking API events have an optional instance ID.

The instance ID contains either or both of the following fields:

- Transaction ID
- Transaction Data

The transaction ID exists purely for assisting the event correlation system; it is an identifier common to all (or a subset of) events belonging to an instance of a transaction. Normally, the correlation system will have to iteratively build up a transaction by following each linking and stitching ID. If a transaction ID is specified, the correlation system can request all events with that ID up-front, thus reducing the time to completion.

Events may also contain transaction data. Transaction data is data particular to an instance of a transaction. It is only used for labeling in reports and graphs of transactions. Only data that is particular to a instance of the transaction is included, that is, data that is not aggregated, and is not used for linking and stitching. The data is presented when the transaction instance is visualized. For example, a web application might report the parameters of an HTTP request in instance data, and the server and page part of the request in the aggregated data (context).

Example: Java

```
Event event = server.createEvent();
event.getInstanceID().setTransactionID("SomeUniqueTransaction");
event.getInstanceID().getTransactionData.put("name", "value");
```

Example: C/C++

```
tt_event_t event;
tt_values_list_t transaction_data;

transaction_data.name = "name";
transaction_data.value = "value";
transaction_data.size = sizeof("value") - 1;
transaction_data.next = 0;

event.instance_id.transaction_id = "SomeUniqueTransaction";
event.instance_id.size = sizeof("SomeUniqueTransaction") - 1;
event.instance_id.transaction_data = &transaction_data;
```

Context information

A typical environment monitored by ITCAM for Transaction Tracking produces large amounts of tracking data. For this reason, ITCAM for Transaction Tracking aggregates the tracking data. Timings and other statistics are aggregated by their common contextual information.

Contextual information is also used for labeling nodes in the visualization of a transaction, and for linking to domain-specific IBM Tivoli Monitoring applications.

Contextual information is information related to the circumstances in which the transaction event occurred. For example, the name or address of the host on which the event occurred, or the host that caused the event to occur. Similarly, the name of the application from which the transaction originated or the application that is presently processing the transaction are also contextual information. Such information allows users to aggregate response times between hosts, between applications, and so on. It is not, however, instance data, that is, it is not specific to one event, but typically specific to a *flow* of events.

Contextual information is stored in two fields: the vertical context, and horizontal context. Vertical context is intended to contain information about the transaction, application or host where the event occurred. Horizontal context is intended to contain information about the message or interaction between two applications or hosts. For example, the vertical context might contain the host name of the machine on which an event occurred, and the horizontal context might contain the type of HTTP request that caused the event to occur.

For a transaction moving through a physical topology, an event's vertical context (for example, the hostname, physical location, application name) is used to label the individual nodes (that is, the hosts) in the graph. An event's horizontal context (for example, the query type, message queue name) is used to label the edges between those nodes.

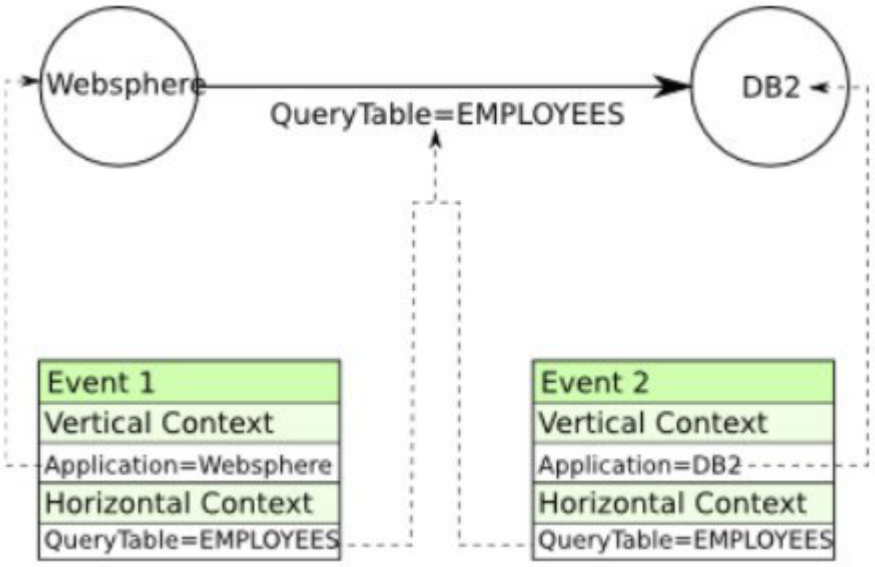


Figure 2. Contextual information in a transaction

Similarly to creating stitching IDs, providing contextual information requires cooperation between the programmers instrumenting the various applications. In particular, the names of the items of information should match where transactions should be grouped by that information. Names are case-sensitive, and aggregation performs a binary equality comparison on them.

ITCAM for Transaction Tracking workspaces also depend on names to provide a further hierarchy of information. The following four Vertical Contexts must be included in every event sent:

ServerName
The server name or address of the machine on which the event occurred. For example, `win001`.

For z/OS users, this must be the Sysplex name and the z/OS host name (SMF id), separated by a forward slash (/). For example, `SYSPLEXQ/MVS1`.

ComponentName
The name of the component in which the event occurred, For example CICS®, Websphere Application Server, MQ: `MQ`.

ApplicationName
The name of the application in which the event occurred. For example, the CICS region name or the MQ Queue Manager name: `CICS001`.

TransactionName
A common identifier for a group of transactions. If this is known by all participants in the transaction, you can easily view aggregate information for all events occurring within transactions of that group. For example, `TXN1`.

The horizontal context distinguishes between interactions at an aggregate level, which is of particular benefit when a mesh of interactions occurs. ITCAM for Transaction Tracking workspaces do not currently display this information, but the interaction data presented is more accurate when provided. Some suggestions for common contextual information to include in events are:

Resource
For applications that query some resource in an external application, such as a message queue or database table, the name of that resource. For example, the queue name or database table name.

SourceHost
The host name of the source of the interaction.

DestinationHost
The host name of the destination of the interaction.

The contexts that the Transaction Collector should aggregate are specified in the Context Mask file. See Appendix E, "Transaction Collector Context Mask," on page 67 for more information on this file.

Blocking events

Blocking events indicate the start of a transaction that causes an application or process to wait for a response.

Transaction Tracking API allows you to provide enough information when creating an event so that the correlation system can determine whether or not the event is part of a synchronous transaction. This can help the system when it calculates the System Time metric of an interaction. A description of the metrics produced by the correlation system is outside the scope of this document.

Specify events as blocked if and only if they are related to the start of a transaction that will cause the code to wait for a response, which is referred to as a *synchronous transaction*. Note that blocking can still occur in an asynchronous transaction; for example, an asynchronous transaction may synchronize at some point, leading to a blocking event being generated.

Java

```
event.setBlocked(true);
```

C/C++

```
event.blocked = 1;
```

Example: blocking events

This example shows how to instrument a partially asynchronous transaction.

In the transaction example shown in Figure 3 on page 22, the client application makes a nonblocking request to the server application, and then continues to perform some computation. When it is ready to block while waiting for the response, it sends a `HERE` event with the `blocked` flag set. This indicates that a previously asynchronous transaction has become synchronous. The `HERE` event type indicates that an event occurs after a transaction starts and before the transaction ends, but is not necessarily related to an interaction between applications.

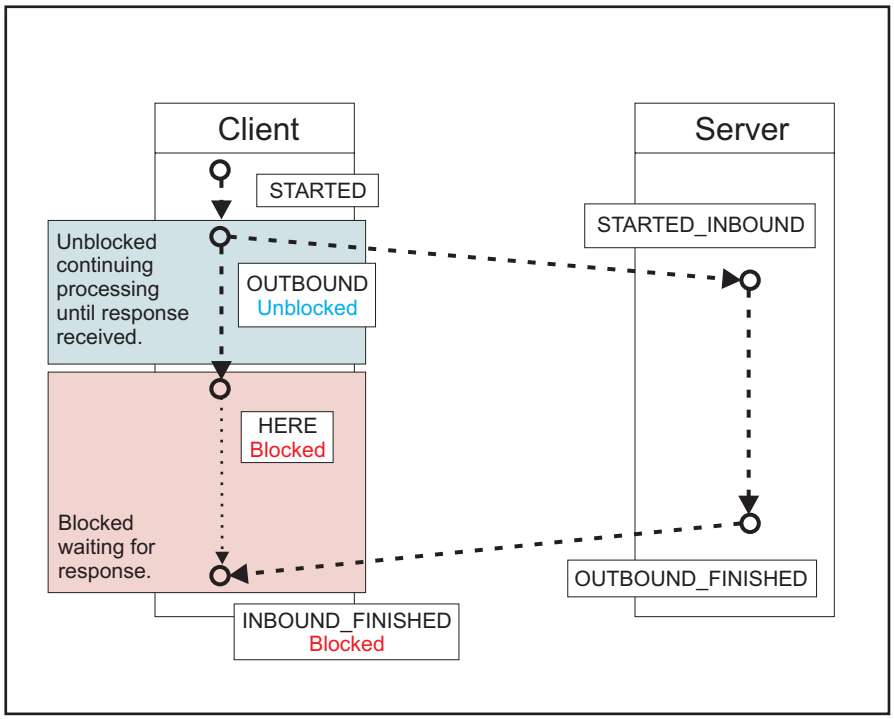


Figure 3. Partially asynchronous transaction

Platform specific issues

Applications on z/OS send event data using EBCDIC with the exception of Java applications.

By default, ITCAM for Transaction Tracking translates all event data from EBCDIC to ASCII. However if some data supplied is binary or ASCII data, this translation must be avoided. The Transaction Tracking API structure includes flags that can be set to stop this translation. See the SCYTSAMP dataset for examples.

Chapter 6. High Level Language reference

This section is for COBOL, C, C++ and PL/I High Level Languages. Transaction Tracking API function names all begin with `cyta` for z/OS, and `tt` for non-z/OS platforms (although `cyta` can also be used).

Functions

Descriptions of ITCAM for Transaction Tracking functions callable from High Level Languages.

Function: check_version

Name: `CYTA_check_version` (z/OS and non-z/OS); `TT_check_version` (non-z/OS)

Purpose: Checks that the header being compiled against matches the library being linked against.

Parameters required: None.

Return codes: See Appendix B, "Return codes," on page 47.

C definition: `int CYTA_check_version(void);`

Notes:

- Call `CYTA_check_version` before any other Transaction Tracking API function, including `CYTA_init`.
- Not available on z/OS.
- Not required, however it can assist developers in cases where structures defined in an older header do not line up with what is expected by a newer library.

Examples:

C:

```
#include <cytapi.h>
rc = CYTA_check_version();
```

Function: init

Name: `CYTA_init` (z/OS and non-z/OS); `TT_init` (non-z/OS)

Purpose: InitializeTransaction Tracking API for High Level Language Callers.

Parameters required: Configuration Block with valid server as described in Appendix A, "Transport address format," on page 45.

Return codes: See Appendix B, "Return codes," on page 47.

C Definition: `int CYTA_init(cyta_config_t *config);`

Notes:

- After `init` has been called, the Configuration Block must not be modified
- Callers must first initialize the Configuration Block to zeroes.
- If the server field is set to none, the default server will be used. See Appendix A, "Transport address format," on page 45 for further information.

- Two separate Transaction Tracking API configurations cannot exist within the same process. The Transaction Tracking API may communicate with only one Transaction Collector which receives all generated events. This is not applicable to z/OS.

Examples:

C:

```
#include <cytapi.h>
  cyta_config_t  configblk;
 memset(&configblk, 0, sizeof(config));
 configblk.server = "tcp:svr.mycompany.com:5455";
  rc = CYTA_init(&configblk); /* Config token in configblk*/
```

COBOL:

```
DATA DIVISION.
Working-Storage Section.
COPY CYTABCON.
COPY CYTABCFG.
01 SERVER  pic x(8) value 'SSN:CYTZ';
01 RC  pic S(9) comp.
PROCEDURE DIVISION.
SET CYTA-CFG-SERVER TO ADDRESS OF SERVER.
CALL "CYTA_init" USING CYTA-CFG-BLOCK RETURNING RC.
```

PL/I:

```
%include CYTAPEVT;            /* Area to hold event blk  */
%include CYTAPCFG;           /* Area to hold Config Blk */
Dcl server Char(8) Init("SSN:CYTZ");
Dcl stgarea      Area;    /* stgarea is 1000 bytes    */
Allocate cytacfg In(stgarea);
cytacsrv = Addr(server);    /* Set Container subsystem  */
Call CYTA_init(cytacfg);    /* Config token in cytacfg  */
```

Function: shutdown

Name: `CYTA_shutdown` (z/OS and non-z/OS); `TT_shutdown` (non-z/OS)

Purpose: Shut down Transaction Tracking API. For z/OS, this function is not required, and is included for compatibility with other platforms only.

Parameters required: Configuration Block initialized with `CYTA_init`.

Return codes: See Appendix B, “Return codes,” on page 47.

C Definition: `int CYTA_shutdown(cyta_config_t *config);`

Example:

C:

```
#include <cytapi.h>
int rc;
cyta_config_t  configblk;
cyta_event_t eventblk;
memset(&configblk, 0, sizeof(configblk));
memset(&eventblk, 0, sizeof(eventblk));
configblk.server = "tcp:svr.mycompany.com:5455";
rc = CYTA_init(&configblk);
/* code here to populate event block     */
rc = CYTA_track(&configblk, &eventblk);
rc = CYTA_shutdown(&configblk);
```

Function: strerror

Name: CYTA_strerror (z/OS and non-z/OS); TT_strerror (non-z/OS)

Purpose: Return a string describing a return code from a Transaction Tracking API function.

Parameters required: Return code from a Transaction Tracking API function.

Output: String describing the error code.

Return codes: None.

C Definition: `const char* CYTA_strerror(int errno);`

Notes: Only available for C and C++.

Examples:

C:

```
#include <cytapi.h>
int rc;
rc = CYTA_init(&configblk);
if (rc > 0)
    printf("Init error: %s\n", CYTA_strerror(rc));
```

Function: time

Name: CYTA_time (z/OS and non-z/OS); TT_time (non-z/OS)

Purpose: Get time now in seconds and microseconds since 00:00:00, Jan 1, 1970.

Parameters required: tt_time_t structure to receive time.

Output: Current wall clock time returned in the tt_time_t structure.

Return codes: None.

C Definition: `cyta_time_t* CYTA_time(cyta_time_t*);`

Notes: This function is only required if a timestamp different from the current timestamp is required on an event. If the timestamp on an event sent to track is zero, the current time is automatically inserted.

Examples:

C:

```
#include <cytapi.h>
 int rc;
 cyta_time_t now;
  rc = tt_time_t;
```

COBOL:

```
DATA DIVISION.
Working-Storage Section.
COPY CYTABEVT.
01 RC  pic S(9) comp.
PROCEDURE DIVISION.
CALL "CYTA_time" USING CYTA-E-SECONDS RETURNING RC.
```

PL/I:

```
%include CYTAPEVT;
Dcl stgarea      Area;    /* stgarea is 1000 bytes */
Allocate cytacfg In(stgarea);
Call CYTA_time(cytaesec);
```

Function: token

Name: CYTA_token (z/OS and non-z/OS); TT_token (non-z/OS)

Purpose: Obtain a fullword token unique across the enterprise for High Level Language Callers.

Parameters required: Configuration Block initialized with CYTA_init.

Output: Fullword unique token returned in area supplied.

Return codes: See Appendix B, "Return codes," on page 47.

C Definition: `int CYTA_token(cyta_config_t *config, cyta_int32_t *token);`

Examples:

C:

```
#include <cytapi.h>
  int rc;
int token;
 cyta_config_t  configblk;
memset(&configblk, 0, sizeof(config));
 configblk.server = "tcp:svr.mycompany.com:5455";
  rc = CYTA_init(&configblk);
 if (rc == TT_SUCCESS)
 rc = CYTA_token(&configblk, &token);
```

COBOL:

```
DATA DIVISION.
Working-Storage Section.
COPY CYTABCON.
COPY CYTABCFG.
01 SERVER  pic x(8) value 'SSN:CYTZ';
01 TOKEN  pic S(9) comp.
01 RC  pic S(9) comp.
PROCEDURE DIVISION.
SET CYTA-CFG-SERVER TO ADDRESS OF SERVER.
CALL "CYTA_init" USING CYTA-CFG-BLOCK RETURNING RC.
CALL "CYTA_token" USING CYTA-CFG-BLOCK TOKEN RETURNING RC.
```

PL/I:

```
%include CYTAPEVT;
 %include CYTAPCFG;
Dcl server Char(8) Init("SSN:CYTZ");
Dcl token Fixed(32);
Dcl stgarea      Area;    /* stgarea is 1000 bytes */
Allocate cytacfg In(stgarea);
cytacsrv = Addr(server);
Call CYTA_init(cytacfg);
Call CYTA_token(cytacfg,token);
```

Function: track

Name: CYTA_track (z/OS and non-z/OS); TT_track (non-z/OS)

Purpose: Send completed event for High Level Language Callers.

Parameters required: Configuration Block initialized with CYTA_init. Completed event block.

Output: Event sent if valid and Transactions operational.

Return codes: See Appendix B, "Return codes," on page 47.

C Definition: `int CYTA_track(cyta_config_t *config, cyta_event_t *event);`

Notes:

- If the time in the event block is zero, the current time is automatically inserted for the event.
- The contents of the event block are unchanged by `track`. Hence the same event block can be used for multiple `track` calls, with only the changed fields being updated

Examples:

C:

```
#include <cytapi.h>
  int rc;
int token;
 cyta_config_t  configblk;
 cyta_event_t eventblk;
memset(&configblk, 0, sizeof(configblk));
 memset(&eventblk, 0, sizeof(eventblk));
 configblk.server = "tcp:svr.mycompany.com:5455";
  rc = CYTA_init(&configblk);
 /* code here to populate event block     */
 rc = CYTA_track(&configblk, &eventblk);
```

COBOL:

```
DATA DIVISION.
Working-Storage Section.
COPY CYTABCON.
COPY CYTABCFG.
COPYT CYTABEVT.
01 SERVER  pic x(8) value 'SSN:CYTZ';
01 TOKEN  pic S(9) comp.
01 RC  pic S(9) comp.
PROCEDURE DIVISION.
SET CYTA-CFG-SERVER TO ADDRESS OF SERVER.
CALL "CYTA_init" USING CYTA-CFG-BLOCK RETURNING RC.
* code here to populate event block
CALL "CYTA_track" USING CYTA-CFG-BLOCK CYTA-EVENT RETURNING RC.
```

PL/I:

```
%include CYTAPEVT;
 %include CYTAPCFG;
 Dcl server Char(8) Init("SSN:CYTZ");
 Dcl token Fixed(32);
 Dcl stgarea      Area;    /* stgarea is 1000 bytes */
 Allocate cytacfg In(stgarea);
 cytacsrv = Addr(server);
 Call CYTA_init(cytacfg);
 /* code here to populate event block   */
 Call CYTA_token(cytacfg,token);
```

C types and structures

Transaction Tracking API provides a set of publicly available data types and structures. The ttapi.h (non-z/OS) and SCYTSAMP CYTAPI (z/OS) include files define many C and C++ data structures that can be used throughout your program. For z/OS, all data types begin with `cyta`, for non-z/OS users, they can start with `tt` or `cyta`.

Basic data types

Transaction Tracking API defines common names for various basic data types:

tt_uint64_t
Unsigned 64-bit integer. (cyta_uint64_t for z/OS)

tt_int64_t
Signed 32-bit integer. (cyta_int64_t for z/OS)

tt_uint32_t
Unsigned 32-bit integer. (cyta_uint32_t for z/OS)

tt_int32_t
Signed 32-bit integer. (cyta_int32_t for z/OS)

tt_uint16_t
Unsigned 16-bit integer. (cyta_uint16_t for z/OS)

tt_int16_t
Signed 16-bit integer. (cyta_int16_t for z/OS)

tt_uint8_t
Unsigned 8-bit integer. (cyta_uint8_t for z/OS)

tt_int8_t
Signed 8-bit integer. (cyta_int8_t for z/OS)

tt_byte_t
Indicates that a memory address is considered as opaque, and may contain any 8–bit value. (cyta_byte_t for z/OS)

tt_config_t (cyta_config_t for z/OS)

Defines generic configuration parameters for Transaction Tracking API.

```
typedef struct TT_CONFIG_T
{
    const char* server;
    tt_uint32_t connect_timeout;
    tt_uint32_t connect_retries;
    tt_uint32_t connect_retry_interval;
    void*       handle;
    const char* token_filename;
    tt_uint16_t queue_size;
} tt_config_t;
```

tt_config_t.server
The address of the Transaction Collector to send events to. The address format is defined in Appendix A, “Transport address format,” on page 45. If this field is set to zero, it is replaced with the default server.

tt_config_t.connect_timeout
Length of time (in seconds) to wait for a TCP/IP connection to the Transaction Collector before timing out. Default is `30000` (30 seconds).

tt_config_t.connect_retries
Number of times to retry a failing TCP/IP connection. If set to zero, ITCAM for Transaction Tracking will retry indefinitely. .

tt_config_t.connect_retry_interval
Interval to wait before retrying TCP/IP connection. Default is `5000` (5 seconds).

tt_config_t.handle
Internal use only.

tt_config_t.token_filename
The filename of the token prefix file. This functionality is reserved for future use.

tt_config_t.queue_size
Maximum number of events to queue while waiting for a TCP/IP connection. Default is `1000`. If more than 1000 events are received and there is no TCP/IP connection to the Transaction Collector, the oldest event is discarded.

Note: If you specify an incorrect transport address when the transport prefix does not refer to an existing transport, such as `tcp` in `tcp:127.0.0.1:5455`, an error is returned.

tt_time_t (cyta_time_t for z/OS)

Describes a point in time relative to the epoch, which is 00:00:00 UTC, January 1, 1970.Transaction Tracking API replaces instances of `tt_time_t` with both zero seconds and zero microseconds with the current time.

```
typedef struct TT_TIME_T
{
    tt_uint32_t sec;
    tt_uint32_t usec;
} tt_time_t;
```

tt_time_t.sec
Seconds component

tt_time_t.usec
Microseconds component

tt_event_t (cyta_event_t for z/OS)

Decribes an event that has occurred in the instrumented application. For example, the event may describe the beginning or completion of a transaction, or the sending or receipt of a request or response.

```
typedef struct TT_EVENT_T
{
    tt_uint32_t         type;
    tt_time_t           timestamp;
    tt_instance_id_t    instance_id;
    tt_association_id_t horizontal_id;
    tt_association_id_t vertical_id;
    tt_values_list_t*   horizontal_context;
    tt_values_list_t*   vertical_context;
    int                 blocked;
    void*               reserved1;
} tt_event_t;
```

tt_event_t.type
The event type. For example, `TT_STARTED_EVENT`, or `TT_OUTBOUND_EVENT`.

tt_event_t.timestamp
The point in time at which the event occurred.

tt_event_t.instance_id
Transaction ID and instance-specific data.

tt_event_t.horizontal_id
Horizontal linking and stitching IDs.

tt_event_t.vertical_id
Vertical linking and stitching IDs.

tt_event_t.horizontal_context
Horizontal context. A NULL-pointer is interpreted as an empty set.

tt_event_t.vertical_context
Vertical context. A NULL-pointer is interpreted as an empty set.

tt_event_t.blocked
Determines the blocked status of the event. Zero equates to unblocked, while a non-zero value equates to blocked.

tt_event_t.reserved1
Reserved for future use.

tt_association_id_t (cyta_association_t for z/OS)

Defines the information that identifies associations between sets of events, that is, the linking and stitching IDs.

```
typedef struct TT_ASSOCIATION_ID_T
{
    tt_uint32_t       caller_type;
    const tt_byte_t*  link_id;
    tt_uint8_t        link_id_size;
    tt_uint16_t       flags;
    tt_values_list_t* stitch_ids;
} tt_association_id_t;
```

tt_association_id_t.caller_type
Caller type for the link ID. This is used to eliminate collisions in link IDs between different callers of Transaction Tracking API.

tt_association_id_t.link_id
Address of the link ID for this event.

tt_association_id_t.link_id_size
Size of the link ID, in 8-bit bytes.

tt_association_id_t.flags
Flags that affect how the association ID is interpreted by Transaction Tracking API. For z/OS, `CYTA_ASSOCIATION_FLAG_LINK_RAW (1)` specifies that the link_id field is binary or ASCII data, and is not to be translated from EBCDIC to ASCII.

tt_association_id_t.stitch_ids
Stitching IDs. A NULL-pointer is interpreted as an empty set.

tt_instance_id_t (cyta_instance_id_t for z/OS)

Defines the information that identifies transaction-specific data.

```
typedef struct TT_INSTANCE_ID_T
{
    const tt_byte_t*  transaction_id;
    tt_uint16_t       size;
    tt_uint16_t       flags;
    tt_values_list_t* transaction_data;
} tt_instance_id_t;
```

tt_instance_id_t.transaction_id
Address of the transaction ID for the event.

tt_instance_id_t.size
Size of the transaction ID for the event, in 8-bit bytes.

tt_instance_id_t.flags
Flags which affect how the instance ID is interpreted by Transaction Tracking API. For z/OS, `CYTA_INSTANCEID_FLAG_RAW (1)` specifies that the `transaction_id` field is binary or ASCII data, and is not to be translated from EBCDIC to ASCII.

tt_instance_id_t.transaction_data
Transaction instance-specific data. A NULL pointer is interpreted as an empty set.

tt_values_list_t (cyta_values_list_t for z/OS)

Defines a singly-linked list of name-value pairs.

```
typedef struct TT_VALUES_LIST_T
{
    struct TT_VALUES_LIST_T* next;
    const char*              name;
    const tt_byte_t*         value;
    tt_uint16_t              size;
    tt_uint16_t              flags;
} tt_values_list_t;
```

tt_values_list_t.next
Pointer to the next item in the list. The last item in the list must have next set to zero.

tt_values_list_t.name
Name portion of the name/value pair. This is expected to be a null-terminated UTF-8 string of size less than or equal to 256 characters (including the null character).

tt_values_list_t.value
Value portion of the name-value pair. This is treated as a binary string. The value does not need to be null-terminated.

tt_values_list_t.size
Size of the value, in 8-bit bytes.

tt_values_list_t.flags
Flags which affect how the name/value pair is interpreted by Transaction Tracking API. Valid flags are:

- `CYTA_VALUELIST_FLAG_NAME_RAW (1)`– for z/OS only. Specifies that the name field is binary or ASCII data, and is not to be translated from EBCDIC to ASCII.
- `CYTA_VALUELIST_FLAG_VALUE_RAW (2)`– for z/OS only. Specifies that the value field is binary or ASCII data, and is not to be translated from EBCDIC to ASCII.

Chapter 7. Java reference

Reference information for the Transaction Tracking API Java wrapper, TTAPI4J, is available separately as Java API documentation (Javadoc).

See the Transaction Tracking API SDK and Javadoc for TTAPI4J that are installed in tusupport/ttapi/doc/ttapi4j as part of the Transaction Collector installation for further information.

Chapter 8. High Level Assembler Reference

This section is for High Level Assembler (HLASM) on z/OS.

HLASM Macro: CYTADFV

Macro to create Name/Value pairs to specify the minimal Vertical Context for an event.

Purpose

Prepare minimum required linked Vertical Context Name/Value pair entries for an event.

Input registers

No requirements.

Output registers

- R0 - 3 used as work register
- R4 - 13 unchanged
- R14 - 15 used as work register

Syntax

Syntax	Description
name	*name*: symbol. Begin name in column 1
`CYTADFV`	One or more blanks must follow `CYTADFV`
`APPL=`*`appl`*	*appl* – Application Name. Constant string in single quotes or pointer to string (register R2 – R12 in brackets or Rx address). Default: Jobname (if batch) or Address Space name (otherwise). Optional.
`APPLLEN=`*`length`*	*length* – Length of Application Name. Decimal constant in single quotes, register (R2-R12 in brackets) or halfword label. Optional if APPL is a label or constant string.
`XLATEA=`*`yes/no`*	If NO, no translation from EBCDIC to ASCII is performed for the application name when sending event to Transaction Collector. Otherwise the application name is translated from EBCDIC to ASCII. Set this value to YES (default) if the application name is an EBCDIC string, NO otherwise. Optional.
`COMPONENT=`*`comp`*	*comp* – Component Name. Constant string in single quotes or pointer to string (register R2 – R12 in brackets or Rx address). Default: BATCH (if batch) or STC (otherwise). Optional.

Syntax	Description
COMPONENTL=*length*	*length* – Length of Component name. Decimal constant in single quotes, register (R2-R12 in brackets) or halfword label. Optional if COMPONENT is a label or constant string
XLATEC=*yes/no*	If NO, no translation from EBCDIC to ASCII is performed for the component name when sending event to Transaction Collector. Otherwise the component name is translated from EBCDIC to ASCII. Set this value to YES (default) if the component name is an EBCDIC string, NO otherwise. Optional.
HOST=*host*	*host* – Host Name. Constant string in single quotes or pointer to string (register R2 – R12 in brackets or Rx address). Default: Sysplex name and z/OS SMFID (separated by a '/' – for example SYSPLEX1/MVS1). Optional.
HOSTL=*length*	*length* – Length of Host Name. Decimal constant in single quotes, register (R2-R12 in brackets) or halfword label. Optional if HOST is a label or constant string.
XLATEH=*yes/no*	If NO, no translation from EBCDIC to ASCII is performed for the host name when sending event to Transaction Collector. Otherwise the host name is translated from EBCDIC to ASCII. Set this value to YES (default) if the host name is an EBCDIC string, NO otherwise. Optional
TXN=*txn*	*txn* – Transaction Name. Constant string in single quotes or pointer to string (register R2 – R12 in brackets or Rx address). Default: (unknown). Optional
TXNL=*length*	*length* – Length of Transaction Name. Decimal constant in single quotes, register (R2-R12 in brackets) or halfword label. Optional if TXNL is a label or constant string.
XLATET=*yes/no*	If NO, no translation from EBCDIC to ASCII is performed for the transaction name when sending event to Transaction Collector. Otherwise the transaction name is translated from EBCDIC to ASCII. Set this value to YES (default) if the transaction name is an EBCDIC string, NO otherwise. Optional.
CHAINTO=*ptr*	*ptr* – Pointer to an existing Vertical Context list that these Name/Value pair entries are to be chained off, or zero if none. Pointer can point to any existing Name/Value pair in an existing Vertical Context list. If set, these Name/Value pairs will be added to the end of the list. Default 0. Optional.
MF=*form*	*form* one of S (inline), L (list form) or (E, list addr) (execute form). Default S. Optional.

Return codes

- none

Notes

- This macro creates four chained Name/Value pairs that provide the minimum Vertical Context required for an event.
- If using execute form, ensure that list address is the list form of the CYTADFV macro, NOT the list form of the CYTANV macro.

Sample

```
    LA    R2,VCONTV
    LH    R3,=AL2(L'VCONTV)
NVNAMC3 CYTANV NAME='Division',Value='Payroll'
    CYTADFV TXN='Txn1',CHAINTO=NVNAMC3,MF=(E,NVNAMC4)
    BR  R14

NVNAMC4 CYTADFV MF=L
```

HLASM Macro: CYTAINIT

Macro to call the CYTA_init function that initializes Transaction Tracking API for HLASM callers.

Purpose

Initialize Transaction Tracking API for HLASM callers, and return a Configuration Token for use by other ITCAM for Transaction Tracking macros.

Input registers

No requirements.

Output registers

- R0 - 3 used as work register
- R1 holds Configuration token if R15 = 0, zero otherwise
- R2 - 13 unchanged
- R14 used as work register
- R15 return code

Syntax

Syntax	Description
name	*name*: symbol. Begin name in column 1
CYTAINIT	One or more blanks must follow CYTAINIT
SUB=*subsystem*	*subsystem* – Container subsystem – constant in single quotes (4 chars), register in brackets, or Rx address
MF=*form*	*form* one of S (inline), L (list form) or (E, list addr) (execute form). Default S. Optional.

Return codes

- As for CYTA_init function, see Appendix B, “Return codes,” on page 47.

Sample

```
   CYTAINIT SUB='CYTZ'
   CYTAINIT SUB=#SUB1
   LA    R5,#SUB1
   CYTAINIT SUB=(R5),MF=(E,INITL)
INITL CYTAINIT MF=L
#SUB1 DC   C'CYTZ'
```

HLASM Macro: CYTANV

Macro to create a Name/Value pair entry.

Purpose

Prepare a Name/Value pair.

Input registers

No requirements.

Output registers

- R0 - 1 used as work register
- R2 - 13 unchanged
- R14 - 15 used as work register

Syntax

Syntax	Description
name	*name*: symbol. Begin name in column 1.
CYTANV	One or more blanks must follow CYTANV.
NAME=*name*	*name* – Name/Value pair name. Constant string in single quotes or pointer to **null terminated** string (register R2 – R12 in brackets or Rx address). Required.
VALUE=*value*	*value* – Name/Value pair value. Constant string in single quotes or pointer to string (register R2 – R12 in brackets or Rx address – value string does not have to be null terminated). Required.
LEN=*length*	*length* – Length of value. Decimal constant in single quotes, register (R2-R12 in brackets) or halfword label. Optional if VALUE is a label or constant string.
XLATEN=*yes/no*	If NO, no translation from EBCDIC to ASCII is performed for the name when sending events to Transaction Collector. Otherwise the name is translated from EBCDIC to ASCII. Set this value to YES (default) if the name is an EBCDIC string, NO otherwise. Optional.

Syntax	Description
XLATEV=*yes*/*no*	If NO, no translation from EBCDIC to ASCII is performed for the value when sending events to Transaction Collector. Otherwise the value is translated from EBCDIC to ASCII. Set this value to YES (default) if the value of the Name/Value pair is an EBCDIC string, NO otherwise. Optional.
CHAINTO=*ptr*	*ptr* – Pointer to an existing Name/Value pair list hat this Name/Value pair entry is to be chained off, or zero if none. Pointer can point to any existing Name/Value pair in an existing Name/Value pair list. If set, this Name/Value pair will be added to the end of the list. Default 0. Optional.
MF=*form*	*form* one of S (inline), L (list form) or (E, list addr) (execute form). Default S. Optional.

Return codes

- none

Notes

- Label is required for inline form of CYTANV.

Sample

```
NVNAMC3 CYTANV NAME='Transaction',VALUE=TXNVAL,     X
          LEN=TXNVALL
    LA    R2,VCONTV
    LA    R3,L'VCONTV
    CYTANV NAME=VCONTN,VALUE=(R2),LEN=(R3),     X
     CHAINTO=NVNAMC3,MF=(E,NVNAMC4)
    BR  R14

NVNAMC4  CYTANV MF=L
VCONTN   DC  'Process'        Name
         DC  X'00'            MUST be null terminated
VCONTV   DC  'ATM'            Value (not null terminated)
TXNVAL   DC  'Txn1'           Value (not null terminated)
TXNVALL  DC  AL2(L'TXNVAL)    Length of value
```

HLASM Macro: CYTATOK

Macro to call the CYTA_token function.

Purpose

Obtain fullword token unique across enterprise for HLASM callers.

Input registers

No requirements.

Output registers

- R0 used as work register
- R1 holds unique token if R15 = 0, zero otherwise
- R2 - 13 unchanged
- R14 used as work register
- R15 return code

Syntax

Syntax	Description
name	*name*: symbol. Begin name in column 1.
CYTATOK	One or more blanks must follow CYTATOK
TOKEN=*token*	*token* – Fullword Configuration token from CYTAINIT macro. Register in brackets, or Rx address. Required.
MF=*form*	*form* one of S (inline), L (list form) or (E, list addr) (execute form). Default S. Optional.

Return codes

- As for CYTA_token function. See Appendix B, "Return codes," on page 47.

Sample

```
    CYTAINIT SUB='CYTZ'
    ST  R1,ETOKEN
    LR  R5,R1
    CYTATOK TOKEN=ETOKEN
    CYTATOK TOKEN=(R5),MF=(E,TOKL)
TOKL   CYTATOK MF=L
ETOKEN  DS  F
```

HLASM Macro: CYTATRAK

Macro to call the CYTA_track function.

Purpose

Send completed event for HLASM callers.

Input registers

No requirements.

Output registers

- R0, R1 used as work register
- R2 - 13 unchanged
- R14 used as work register
- R15 return code

Syntax

Syntax	Description
name	*name*: symbol. Begin name in column 1.

Syntax	Description
`CYTATRAK`	One or more blanks must follow `CYTATRAK`
type	*type* – Type of event. Must be one of: STARTED, HERE, INBOUND, OUTBOUND, FINISHED, STARTED_IN, OUTBOUND_FIN, INBOUND_FIN. Required.
`TIME=`*`time`*	*time* – STCK value for timestamp of event. If not specified, current time is used. Register R2 – R12 in brackets, or Rx address. Optional.
`TXN=`*`transaction`*	*transaction* – Transaction Identifier. Constant string in single quotes or pointer to string (register R2 – R12 in brackets or Rx address). Optional.
`TXNLEN=`*`length`*	*length* – Length of Transaction Identifier. Decimal constant in single quotes, register (R2-R12 in brackets) or halfword label. Optional if TXN is a label or constant string.
`XLATET=`*`yes/no`*	If NO, no translation from EBCDIC to ASCII is performed for the Transaction Identifier when sending events to Transaction Collector. Otherwise the Transaction Identifier is translated from EBCDIC to ASCII. Set this value to YES (default) if the Transaction Identifier is an EBCDIC string, NO otherwise. Optional.
`TXNTXT=`*`list`*	*list* – Pointer to Transaction Context List – a list of Name/Value pointers. Pointer (register R2 – R12 in brackets or Rx address) or zero if none. Default 0. Optional.
`HTYPE=`*`type`*	*type* – Horizontal Caller type. Must be a valid Caller Type or a number between 0 and 255. Default is ANY. Optional.
`HLINK=`*`linkid`*	*linkid* – Horizontal Link. Constant string in single quotes or pointer to string (register R2 – R12 in brackets or Rx address). Optional.
`HLINKL=`*`length`*	*length* – Length of Horizontal Link. Decimal constant in single quotes, register (R2-R12 in brackets) or halfword label. Optional if HLINK is a label or constant string.
`XLATEH=`*`yes/no`*	If NO, no translation from EBCDIC to ASCII is performed for Horizontal Link when sending event to Transaction Collector. Otherwise Horizontal Link is translated from EBCDIC to ASCII. Set this value to YES (default) if Horizontal Link is an EBCDIC string, NO otherwise. Optional.
`HCTXT=`*`list`*	*list* – Pointer to Horizontal Context List – a list of Name/Value pointers. Pointer (register R2 – R12 in brackets or Rx address) or zero if none. Default 0. Optional

Syntax	Description
HSTITCH=*list*	*list* – Pointer to Horizontal Stitch List – a list of Name/Value pointers. Pointer (register R2 – R12 in brackets or Rx address) or zero if none. Default 0. Optional.
VTYPE=*type*	*type* – Vertical Caller type. Must be a valid Caller Type or a number between 0 and 255. Default is ANY. Optional.
VLINK=*linkid*	*linkid* – Vertical Link. Constant string in single quotes or pointer to string (register R2 – R12 in brackets or Rx address). Optional.
VLINKL=*length*	*length* – Length of Vertical Link. Decimal constant in single quotes, register (R2-R12 in brackets) or halfword label. Optional if VLINK is a label or constant string.
XLATEV=*yes/no*	If NO, no translation from EBCDIC to ASCII is performed for Vertical Link when sending events to the Transaction Collector. Otherwise Vertical Link is translated from EBCDIC to ASCII. Set this value to YES (default) if Vertical Link is an EBCDIC string, NO otherwise. Optional.
VCTXT=*list*	*list* – Pointer to Vertical Context List – a list of Name/Value pointers. Pointer (register R2 – R12 in brackets or Rx address) or zero if none. Default 0. Optional.
VSTITCH=*list*	*list* – Pointer to Vertical Stitch List – a list of Name/Value pointers. Pointer (register R2 – R12 in brackets or Rx address) or zero if none. Default 0. Optional.
TOKEN=*token*	*token* – Fullword Configuration token from CYTAINIT macro. Register in brackets, or Rx address. Required.
MF=*form*	*form* one of S (inline), L (list form) or (E, *list addr*) (execute form). Default S. Optional.

Return codes

- As for CYTA_track function. See Appendix B, “Return codes,” on page 47.

Notes

- If using execute form, event block is NOT cleared before event is sent. Any previous event values will not be reset.
- If execute form of CYTATRAK is used with no parameters except TOKEN, then the list address must point to a fully formed event block.
- Valid Caller Types are:
  - ANY – Type 0 – no specified caller type
  - GPS – Type 1 - GPS
  - ARM – Type 2 - ARM
  - WSA – Type 3 – WSA
  - CTG – Type 4 – CICS Transaction Gateway
  - MQ – Type 5 – Websphere MQ

- SOA – Type 6 - SOA
- WR – Type 7 – Web Resources
- CICS – Type 8 - CICS
- IMS™ – Type 9 – IMS

Caller Types can be any of these strings, or any number between 0 and 255. Caller Types 0-199 are reserved by IBM. Caller Types 200-255 are available to users.

Sample

```
CYTAINIT SUB='CYTZ'
ST R1,ETOKEN
LA       R5,VCONTXT
CYTATRAK TYPE=STARTED,VCTXT=VCONTXT,VLINK='LINK1', X
         HLINK=HLINK1,HLINKL=HLINK1L,TOKEN=ETOKEN
CYTATRAK TYPE=HERE,VCTXT=(R5),MF=(E,TRAKL)
* Code to fully populate event EVENT1 here
CYTATRAK TOKEN=ETOKEN,MF=(E,EVENT1)
BR R14
TRAKL CYTATRAK MF=L
EVENT1   CYTAEVNT MF=L
ETOKEN  DS F
#HLINK1  DC    C'HLinkID'
#HLINK1L DC    AL2(L'#HLINK1)
```

Appendix A. Transport address format

This appendix provides definitions of the available addressing schemes for connecting an instrumented application to a Transaction Collector.

The addressing format that Transaction Tracking API uses is modular - multiple transports may be available depending on which platform the Transaction Tracking API is running. There is a subset of transports guaranteed to be available on all platforms. Each transport has a unique string associated with it.

The addresses used by Transaction Tracking API are always prefixed by the unique identifier of the transport to be used, followed by a colon (:). The remainder of the address following the colon, is interpreted in a module-dependant manner as follows:

```
<module>:<address>
```

The following sections describe each module and the formats they define for their addresses.

TCP/IP module (tcp)

The TCP/IP transport supports both IPv4 and IPv6 (where the platform supports IPv6). The module's unique identifier is `tcp`. This format is not supported on z/OS.

Addresses for this module follow a URL-like format:

```
tcp:host:port
```

To allow the use of IPv6 addresses in this format, the host must be enclosed in square brackets (as is done in URLs). For example, to connect to port 5455 on the IPv6 local host, specify the following address:

```
tcp:[::1]:5455
```

The default TCP/IP value is `tcp:127.0.0.1:5455`.

Subsystem module (ssn)

The subsystem format is supported only on z/OS, and specifies the four character subsystem name of the destination Transactions Container.

Addresses for this module are:

```
ssn:subsystem
```

For example, `ssn:SS01`.

The default SSN value is `ssn:CYTZ`. All other fields in the Configuration Block are ignored for Subsystem users.

Appendix B. Return codes

Transaction Tracking API functions return values in a return code – a fullword area.

The return codes are:

- **0** Operation successful
- **10 - 19: z/OS Related Error:**
  - –
  - 10 - Container subsystem not found
  - 11 - Invalid Configuration Token passed
  - 12 - Container subsystem inactive
  - 13 - Dispatcher unavailable
  - 14 - No more storage available
  - 15 - No more ASIDs in ASID pool
  - 16 - No Couriers available
  - 17 - System Error Occurred
  - –
- **20-29: Distributed (non-z/OS) Related Error:**
  - 21 - Invalid
  - 22 - Not Implemented
  - 23 - Transport error
  - 24 - No memory
  - 25 - System error occurred
  - 26 - Logic error occurred
  - 27 - Unavailable, try again.
  - 28 - Timeout
- **100-200 Event Record Invalid:**
  - 100 - Internal Error
  - 101 - Invalid event block address
  - 102 - Invalid event type
  - 103 - Invalid timestamp
  - 104 - Invalid transaction ID length (not positive)
  - 105 - Invalid transaction ID flag
  - 106 - Invalid transaction ID address
  - 110 - Invalid horizontal ID length (not positive)
  - 111 - Invalid horizontal ID flag
  - 112 - Invalid horizontal caller type
  - 113 - Invalid horizontal ID address
  - 120 - Invalid vertical ID length (not positive)
  - 121 - Invalid vertical ID flag
  - 122 - Invalid vertical caller type
  - 123 - Invalid vertical caller address

- 130 - Invalid Name string address in transaction list
- 131 - Invalid Value string address in transaction list
- 132 - Invalid Value string length in transaction list
- 133 - Invalid Name Value flag in transaction list
- 134 - Invalid Name/Value pair address in transaction list
- 140 - Invalid Name string address in horizontal stitch list
- 141 - Invalid Value string address in horizontal stitch list
- 142 - Invalid Value string length in horizontal stitch list
- 143 - Invalid Name Value flag in horizontal stitch list
- 144 - Invalid Name/Value pair address in horizontal stitch list
- 150 - Invalid Name string address in vertical stitch list
- 151 - Invalid Value string address in vertical stitch list
- 152 - Invalid Value string length in vertical stitch list
- 153 - Invalid Name Value flag in vertical stitch list
- 154 - Invalid Name/Value pair address in vertical stitch list
- 160 - Invalid Name string address in horizontal context list
- 161 - Invalid Value string address in horizontal context list
- 162 - Invalid Value string length in horizontal context list
- 163 - Invalid Name Value flag in horizontal context list
- 164 - Invalid Name/Value pair address in horizontal context list
- 170 - Invalid Name string address in vertical context list
- 171 - Invalid Value string address in vertical context list
- 172 - Invalid Value string length in vertical context list
- 173 - Invalid Name Value flag in vertical context list
- 174 - Invalid Name/Value pair address in vertical context list

Appendix C. Samples

Code examples.

z/OS samples provided

On z/OS, examples can also be found in the SCYTSAMP file, as described in Table 5.

Table 5. Samples in the SCYTSAMP library

Language	Member	Description
HLASM	CYTAASM	Sample HLASM program to send events.
	CYTAINIT	Macro to call the CYTA_init function.
	CYTATRAK	Macro to call the CYTA_track function.
	CYTATOK	Macro to call the CYTA_token function.
	CYTAEVNT	Macro to map an event.
	CYTACFG	Macro to map a Configuration Block.
	CYTANVAL	Macro to map a Name/Value pair entry.
	CYTANV	Macro to create a Name/Value pair entry.
	CYTADFV	Macro to create Name/Value pairs to specify the minimal Vertical Context for an event.
COBOL	CYTABCFG	COBOL definitions for a Configuration Block.
	CYTABCON	COBOL constants required to use the Transaction Tracking API.
	CYTABEVT	COBOL definitions for an event.
	CYTABNV	COBOL definitions for a Name/Value pair entry.
	CYTABSMP	Sample COBOL program to send events.
C	CYTACSMP	Sample C program to send events. C header file definitions are in the SCYTH dataset.
Java	CYTAJSMP	Sample Java program to send events.
PL/I	CYTAPCFG	PL/I definitions for a Configuration Block.
	CYTAPEVT	PL/I definitions for an event.
	CYTAPNV	PL/I definitions for a Name/Value pair entry.
	CYTAPSMP	Sample PL/I program to send events.
All	CYTASIDE	Binder Side Deck required by dynamic callers.

C

```
/*================================================================
  Includes
  ================================================================*/
#pragma runopts(POSIX(ON))
#pragma runopts(TRAP(ON,SPIE))
#include <cytapi.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <unistd.h>
```

```
/*================================================================
Mainline Code
================================================================ */
int main(int argc, char **argv)
{
    /* ---------------------------------------------------------------
       Variables
       ------------------------------------------------------------- */
      int rc;                           /* Return code from functions  */
      int hlink1;                       /* Store numeric Hor Link IDs  */
      /* --- Area for Communications Configuration Block -------------- */
     cyta_config_t configblk;           /* Configuration block         */
      /* --- Area for Event Block ------------------------------------- */
     cyta_event_t eventblk;             /* Event block                 */
      /* --- Area for 'Standard' Vertical Context Name/Value Pairs ---- */
     cyta_values_list_t vert1, vert2, vert3, vert4; /* Vertical Context*/
     /* ---------------------------------------------------------------
       Put together 'standard' Vertical Contexts

   These are the minimum Vertical Context needed to display
       event in ITCAM for Txns workspaces. We need:
          HOST - Host Name. Normally Sysplex/SMFID
          COMPONENT - Component - what we are running under -eg.
             BATCH, STC, IMS, CICS, WAS, TSO.
          APPLICATION - Usually Job or Started Task Name
          TRANSACTION - The transaction we are running

       In your program, you will modify the values to suit your
       installation, however the names of the name/value pairs should
       not be changed
       ------------------------------------------------------------- */
    /* --- Hostname ------------------------------------------------- */
     memset(&vert1, 0, sizeof(vert1));
     static char *server = "Sysplex/Host";
     static char *server_lbl = "ServerName";
     vert1.name = server_lbl;
     vert1.value = server;
     vert1.size = strlen(server);
     /* --- ComponentName-------------------------------------------- */
     memset(&vert2, 0, sizeof(vert2));
     static char *component = "STC";
     static char *component_lbl = "ComponentName";
     vert1.next = &vert2;
     vert2.name = component_lbl;
     vert2.value = component;
     vert2.size = strlen(component);
     /* --- ApplicationName ------------------------------------------ */
     memset(&vert3, 0, sizeof(vert3));
     static char *application = "Application";
     static char *application_lbl = "ApplicationName";
     vert2.next = &vert3;
     vert3.name = application_lbl;
     vert3.value = application;
     vert3.size = strlen(application);
     /* --- Transaction (no EBCDIC -> ASCII Translation) -------------- */
     memset(&vert4, 0, sizeof(vert4));
     static int transaction = 254;      /* Value is a number: 254      */
     static char *transaction_lbl = "TransactionName";
     vert3.next = &vert4;
     vert4.name = transaction_lbl;
     vert4.value = &transaction;
     vert4.size = sizeof(transaction);
     vert4.flags = CYTA_VALUELIST_FLAG_VALUE_RAW;/* No translation     */
     /* ---------------------------------------------------------------
       Get Configuration Token
          The server field specifies where the ITCAM for Transactions
          events are to be sent. It must be of the form:
```

```
            SSN:sub
         Where sub is the 4 character subsystem name used by the ITCAM
         for Transactions Collector Started Task
       ------------------------------------------------------------- */
     memset(&configblk, 0, sizeof(configblk));  /* Zero config block   */
     configblk.server = "SSN:CYTZ";     /* Send events to CYTZ subsys  */
     rc = CYTA_init(&configblk);        /* Get the token               */
     printf("(CYTACSMP) CYTAINIT Return Code=%d\n", rc);
     /* -------------------------------------------------------------
       Send a 'Started' Event
          We don't specify a timestamp, so the time now is automatically
          inserted.
       ------------------------------------------------------------- */
     memset(&eventblk, 0, sizeof(eventblk));  /* Zero event block      */
     eventblk.type = CYTA_STARTED_EVENT;      /* Started Event         */
     eventblk.vertical_context = &vert1;      /* Vertical Context Addr */
     eventblk.vertical_id.link_id = "CYTACSMP";  /* Vertical Link ID   */
     eventblk.vertical_id.link_id_size = 8;   /* Link ID Length        */
     rc = CYTA_track(&configblk, &eventblk);  /* Send the event        */
     printf("(CYTACSMP) STARTED Event Return Code=%d\n", rc);
     /* -------------------------------------------------------------
       Send an Outbound Event
       (we only need to specify changed fields - all other fields remain
       from the Started event)
       ------------------------------------------------------------- */
     eventblk.type = CYTA_OUTBOUND_EVENT;     /* Started Event         */
     eventblk.horizontal_id.link_id = "Hlink Value";/* Horizontal ID   */
     eventblk.horizontal_id.link_id_size = 11;/* Link ID Length        */
     rc = CYTA_track(&configblk, &eventblk);  /* Send the event        */
     printf("(CYTACSMP) OUTBOUND Event Return Code=%d\n", rc);
     /* -------------------------------------------------------------
       Send an Inbound Finished Event
       We change the Horizontal Link to the incoming Link ID - this
        is the ID specified by the application sending the response
        in its OUTBOUND event. In this case, the Link ID is a number,
        so we MUST set the flag so it is NOT translated from EBCDIC to
        ASCII.
       ------------------------------------------------------------- */
     eventblk.type = CYTA_INBOUND_FINISHED_EVENT;   /* Event Type      */
     hlink1 = 56;                         /* Horizontal Link ID=56     */
     eventblk.horizontal_id.link_id = &hlink1 /* Horizontal ID         */
     eventblk.horizontal_id.link_id_size = 4; /* Link ID Length        */
     eventblk.horizontal_id.flags = CYTA_ASSOCIATION_FLAG_LINK_RAW;
                                          /* Do NOT xlate from EBCDIC    */
     rc = CYTA_track(&configblk, &eventblk);  /* Send the event        */
     printf("(CYTACSMP) INBOUND FINISHED Event Return Code=%d\n", rc);
 }                                        /* main                        */
```

COBOL

```
CBL   RENT,PGMNAME(LM),LIB,NODYNAM,NODLL
      * ===============================================================
      *
      * Identification Division
      *
      * ===============================================================
       IDENTIFICATION DIVISION.

       PROGRAM-ID. "CYTABSMP".

      * ===============================================================
      *
      * Environment Division
      *
      * ===============================================================
       ENVIRONMENT DIVISION.
```

```
      * ============================================================
      *
      * Data Division
      *
      * ============================================================
       DATA DIVISION.
       Working-Storage Section.
      * ------------------------------------------------------------
      * Constants Needed to Use the ITCAM for Txns API
      * ------------------------------------------------------------
       COPY CYTABCON.

      * ------------------------------------------------------------
      * Area to hold our event block
      * ------------------------------------------------------------
       COPY CYTABEVT.

      * ------------------------------------------------------------
      * Area to hold our Configuration Block
      * ------------------------------------------------------------
       COPY CYTABCFG.

      * ------------------------------------------------------------
      * Area for 'Standard' 4 Vertical Context Name/Value pairs.
      *    These are the minimum Vertical Context needed to display
      *    event in ITCAM for Txns workspaces
      *       HOST - Host Name. Normally Sysplex/SMFID
      *       COMPONENT - Component - what we are running under -eg.
      *          BATCH, STC, IMS, CICS, WAS, TSO.
      *       APPLICATION - Usually Job or Started Task Name
      *       TRANSACTION - The transaction we are running
      *
      *    Each has three variables:
      *       xxx-LBL - the label of the Name/Value Pair, ending in
      *                 nulls.
      *       xxx     - area to hold the actual value
      *       xxx-NV  - area to hold the name/value pair
      *
      *    In your program, you will modify the values to suit your
      *    installation, however the labels should not be changed
      * ------------------------------------------------------------
       01  HOST-LBL               pic x(11) value z"ServerName".
       01  HOST                  pic x(12) value "Sysplex/Host".
       01  HOST-NV                pic x(16).

       01  COMPONENT-LBL          pic x(14) value z"ComponentName".
       01  COMPONENT              pic x(3)  value "STC".
       01  COMPONENT-NV           pic x(16).

       01  APPLICATION-LBL        pic x(16) value z"ApplicationName".
       01  APPLICATION            pic x(11) value "Application".

01  APPLICATION-NV         pic x(16).

       01  TRANSACTION-LBL        pic x(16) value z"TransactionName".
       01  TRANSACTION            pic s9(9) binary value 254.
       01  TRANSACTION-NV         pic x(16).

      * ------------------------------------------------------------
      * Outbound and Inbound Horizontal Link IDs
      *    For your organisation, specify unique values here. But
      *    for this example, constant values will be used
      *    HLINK-OUT - Outgoing Horizontal Link (string)
      *    HLINK-IN  - Incoming Horizontal Link (number)
      * ------------------------------------------------------------
       01  HLINK-OUT                   pic x(11) value 'HLINK VALUE'.
       01  HLINK-IN                    pic s9(9) binary value 56.
```

```
* -------------------------------------------------------------
* Vertical Link ID
*    This value should be unique for every work unit. We will
*    use a constant in this example.
* -------------------------------------------------------------
 01  VLINK                      pic x(8) value 'CYTABSMP'.

* -------------------------------------------------------------
* String Specifying Destination for Events
*    This must be of the form SSN:sub - sub is the ITCAM for
*    Transactions Container subsystem.
* -------------------------------------------------------------
 01  SERVER                     pic x(8) value 'SSN:CYTZ'.

* -------------------------------------------------------------
* Definition used to insert a one byte length field
*    LINK-LEN - Halfword Link Name length
*    LINK-LEN-BYTE - One byte Link Name length
* -------------------------------------------------------------
 01  LINK-LEN                   pic s9(3) binary.
 01  LINK-LEN-STR               redefines LINK-LEN.
     02   filler                pic x(1).
     02   LINK-LEN-BYTE         pic x(1).

* -------------------------------------------------------------
* Fullword to hold return code from CYTA_track
* -------------------------------------------------------------
 01  RC                         pic S9(9) comp.

 Linkage Section.
* -------------------------------------------------------------
* Map Name/Value Pair Entry
* -------------------------------------------------------------
 COPY CYTABNV.

* =============================================================
*
* Procedure Division
*
* =============================================================
 PROCEDURE DIVISION.
     DISPLAY "(CYTABSMP) Entry".

* -------------------------------------------------------------
* Initialize Our Event Block
* -------------------------------------------------------------
     INITIALIZE CYTA-EVENT REPLACING ALPHANUMERIC BY x"00".

* -------------------------------------------------------------
* Setup a Name/Value Pair for Host
*    1. Address the Name/Value pair
*    2. Initialise the Name/Value pair to nulls
*    3. Set the Name pointer
*    4. Set the Value pointer
*    5. Set the Value length
* -------------------------------------------------------------
     SET ADDRESS OF CYTA-NV-LIST TO ADDRESS OF HOST-NV.
     INITIALIZE CYTA-NV-LIST REPLACING ALPHANUMERIC BY x"00".
     SET CYTA-NV-NAME-POINTER TO ADDRESS OF HOST-LBL.
     SET CYTA-NV-VALUE-POINTER TO ADDRESS OF HOST.
     MOVE LENGTH OF HOST TO CYTA-NV-VALUE-LENGTH.

* -------------------------------------------------------------
* Setup a Name/Value Pair for Component
*    1. Host Name/Value pair chains to Component Name/Value pair
*    2. (steps as for Host Name/Value Pair)
```

```
*    7. Stop EBCDIC->ASCII transaction of Department ID (as it
*       is a number, not a string)
* -------------------------------------------------------------
     SET CYTA-NV-NEXT-POINTER TO ADDRESS OF COMPONENT-NV.
     SET ADDRESS OF CYTA-NV-LIST TO ADDRESS OF COMPONENT-NV.
     INITIALIZE CYTA-NV-LIST REPLACING ALPHANUMERIC BY x"00".
     SET CYTA-NV-NAME-POINTER TO ADDRESS OF COMPONENT-LBL.
     SET CYTA-NV-VALUE-POINTER TO ADDRESS OF COMPONENT.
     MOVE LENGTH OF COMPONENT TO CYTA-NV-VALUE-LENGTH.

* -------------------------------------------------------------
* Setup a Name/Value Pair for Application
* -------------------------------------------------------------
     SET CYTA-NV-NEXT-POINTER TO ADDRESS OF APPLICATION-NV.
     SET ADDRESS OF CYTA-NV-LIST TO ADDRESS OF APPLICATION-NV.
     INITIALIZE CYTA-NV-LIST REPLACING ALPHANUMERIC BY x"00".
     SET CYTA-NV-NAME-POINTER TO ADDRESS OF APPLICATION-LBL.
     SET CYTA-NV-VALUE-POINTER TO ADDRESS OF APPLICATION.
     MOVE LENGTH OF APPLICATION TO CYTA-NV-VALUE-LENGTH.

* -------------------------------------------------------------
* Setup a Name/Value Pair for Transaction
*    Note that the Transaction Value is a number, so we
*    set the flags so that NO EBCDIC to ASCII translation
*    will be performed.
* -------------------------------------------------------------
     SET CYTA-NV-NEXT-POINTER TO ADDRESS OF TRANSACTION-NV.
     SET ADDRESS OF CYTA-NV-LIST TO ADDRESS OF TRANSACTION-NV.
     INITIALIZE CYTA-NV-LIST REPLACING ALPHANUMERIC BY x"00".
     SET CYTA-NV-NAME-POINTER TO ADDRESS OF TRANSACTION-LBL.
     SET CYTA-NV-VALUE-POINTER TO ADDRESS OF TRANSACTION.
     MOVE LENGTH OF TRANSACTION TO CYTA-NV-VALUE-LENGTH.
     MOVE CYTA-DONT-TR-VALUE-FROM-EBCDIC TO CYTA-NV-FLAGS.

* -------------------------------------------------------------
* Call CYTA_init to get Configuration Token
*    1. Specify server string - if this is omitted, the
*       default is: SSN:CYTZ
*    2. Call CYTA_init
* -------------------------------------------------------------
     SET CYTA-CFG-SERVER TO ADDRESS OF SERVER.

     CALL "CYTA_init" USING CYTA-CFG-BLOCK RETURNING RC.
     DISPLAY "(CYTABSMP) CYTA_init Return Code=" RC.

* -------------------------------------------------------------
* Send a STARTED event
*    1. Move in event type
*    2. Move in Vertical link ID
*    3. Move in Vertical link length
*    4. (Don't specify time, so it is automatically inserted)
*    5. Specify Vertical Context that we've built
*    6. Call the API (statically linked).
* -------------------------------------------------------------
     MOVE CYTA-STARTED TO CYTA-E-TYPE.
     SET CYTA-E-VERT-LINK-ID TO ADDRESS OF VLINK.
     MOVE LENGTH OF VLINK TO LINK-LEN.
     MOVE LINK-LEN-BYTE TO CYTA-E-VERT-LINK-LENGTH.
     SET CYTA-E-VERT-CONTEXT-LIST TO ADDRESS OF HOST-NV.

     CALL "CYTA_track" USING CYTA-CFG-BLOCK CYTA-EVENT
        RETURNING RC.
     DISPLAY "(CYTABSMP) STARTED Event Return Code=" RC.

* -------------------------------------------------------------
* Send an OUTBOUND event
*    1. Move in event type
```

```
      *    2. (No need to change Vertical Link or Context)
      *    3. Specify Horizontal Link (program at other end will
      *       need to specify this on it's INBOUND event).
      *    4. Specify Horizontal Link length
      *    5. Call the API
      * -------------------------------------------------------------
           MOVE CYTA-OUTBOUND TO CYTA-E-TYPE.
           SET CYTA-E-HORZ-LINK-ID TO ADDRESS OF HLINK-OUT.
           MOVE LENGTH OF HLINK-OUT TO LINK-LEN.
           MOVE LINK-LEN-BYTE TO CYTA-E-HORZ-LINK-LENGTH.

           CALL "CYTA_track" USING CYTA-CFG-BLOCK CYTA-EVENT
              RETURNING RC.
           DISPLAY "(CYTABSMP) OUTBOUND Event Return Code=" RC.

      * -------------------------------------------------------------
      * Send an INBOUND FINISHED event
      *    1. Move in event type
      *    2. (No need to change Vertical Link or Context)
      *    3. Specify Horizontal Link (program at other end will
      *       need to specify this on it's OUTBOUND event).
      *    4. Specify Horizontal Link length
      *    5. As Horizontal Link is a number, stop conversion from
      *       EBCDIC
      *    6. Call the API
      * -------------------------------------------------------------
           MOVE CYTA-INBOUND-FINISHED TO CYTA-E-TYPE.
           SET CYTA-E-HORZ-LINK-ID TO ADDRESS OF HLINK-IN.
           MOVE LENGTH OF HLINK-IN TO LINK-LEN.
           MOVE LINK-LEN-BYTE TO CYTA-E-HORZ-LINK-LENGTH.
           MOVE CYTA-DONT-TR-FROM-EBCDIC TO CYTA-E-HORZ-LINK-FLAGS.

           CALL "CYTA_track" USING CYTA-CFG-BLOCK CYTA-EVENT
              RETURNING RC.
           DISPLAY "(CYTABSMP) INBOUND FINISHED Event Return Code=" RC.

      * -------------------------------------------------------------
      * And we're done
      * -------------------------------------------------------------

           GOBACK.
```

Java

```
import ttapi4j.ServerFactory;
import ttapi4j.Server;
import ttapi4j.Event;
import ttapi4j.InstanceID;

public class CYTAJSMP
{
    public static void main(StringÝ¨ args) throws Exception
    {
        System.out.println("(CYTAJSMP) Entry");

        /* --- Get Configuration Token - Sending to Subsys CYTZ  ---- */
        Server s = ServerFactory.getServer("ssn:CYTZ");

        /* --- Create Started Event  ------------------------------- */
        Event e = s.createEvent();
        e.setType(Event.Type.STARTED);
        e.getVerticalID().setLinkID("CYTAJSMP");
        e.getVerticalContext().put("ServerName","Sysplex/Host");
        e.getVerticalContext().put("ComponentName", "USS Shell");
        e.getVerticalContext().put("ApplicationName", "Application");
        e.getVerticalContext().put("TransactionName", "CYTAJSMP");
        s.track(e);
```

```
        /* --- Create Outbound Event  ------------------------------ */
        e.getHorizontalID().setLinkID("Hlink Value");
        e.setType(Event.Type.OUTBOUND);
        s.track(e);

        /* --- Create Inbound Finished Event  ---------------------- */
        e.setType(Event.Type.INBOUND_FINISHED);
        s.track(e);

        /* --- Cleanup and Exit ------------------------------------ */
        s.close();
        System.out.println("(CYTAJSMP) Exit");
    }
}
```

PL/I

```
%PROCESS Limits(Extname(15)) Source Arch(1);
%PROCESS Default(Linkage(Optlink) Nullsys);
%PROCESS Margins(2,72) Rules(IBM) System(MVS);
 CYTAPSMP: Procedure Options(main);

 /*====================================================================

  Storage Definitions

  ====================================================================*/
 /* ------------------------------------------------------------------
    Include Structures for ITCAM for Transactions
    ------------------------------------------------------------------ */
 %include CYTAPEVT;
 %include CYTAPCFG;
 %include CYTAPNV;

 /* ------------------------------------------------------------------
    Declarations for 'Standard' 4 Vertical Context Name/Value pairs.
       These are the minimum Vertical Context needed to display
       an event in ITCAM for Txns workspaces:
          Host        - Host Name. Normally Sysplex/SMFID
          Component   - Component - what we are running under, like:
             BATCH, STC, IMS, CICS, WAS, or TSO.
          Application - Usually Job or Started Task Name
          Transaction - The transaction we are running

       We're defining three variables each:
          xxxlbl  - the label of the Name/Value Pair, ending in
                    nulls.
          xxxnam  - area to hold the actual value
          xxxaddr - address of name/value pair

       In your program, you will modify the values to suit your
       installation, however the labels should not be changed
    ------------------------------------------------------------------ */
 Dcl hostaddr    Pointer;
 Dcl hostlbl     Char(11) Varyingz Init('ServerName');
 Dcl hostnam     Char(12) Init('Sysplex/Host');

 Dcl compaddr    Pointer;
 Dcl complbl     Char(14) Varyingz Init('ComponentName');
 Dcl compnam     Char(3)  Init('STC');

 Dcl appladdr    Pointer;
 Dcl applbl      Char(16) Varyingz Init('ApplicationName');
 Dcl applnam     Char(11) Init('Application');

 Dcl tranaddr    Pointer;
```

```
Dcl tranlbl     Char(16) Varyingz Init('TransactionName');
Dcl trannam     Fixed(32) Binary Unsigned Init(254);

/* ------------------------------------------------------------------
   Outbound and Inbound Horizontal Link IDs
      For your organisation, specify unique values here. But
      for this example, constant values will be used
      hlinkout  - Outgoing Horizontal Link (string)
      hlinkin   - Incoming Horizontal Link (number)
   ------------------------------------------------------------------ */
Dcl hlinkout    Char(11) Init("Hlink Value");
Dcl hlinkin     Fixed(32) Binary Unsigned Init(56);

/* ------------------------------------------------------------------
   Vertical Link ID
      This value should be unique for every work unit. We will
      use a constant in this example.
   ------------------------------------------------------------------ */
Dcl vlink       Char(8) Init("CYTAPSMP");

/* ------------------------------------------------------------------
   Communications Server
      This string specifies where the ITCAM for Transactions Events
      are to be sent. It must be of the form:
         SSN:sub
      Where sub is the 4 character subsystem name used by the ITCAM
      for Transactions Collector Started Task
   ------------------------------------------------------------------ */
Dcl server      Char(8) Init("SSN:CYTZ");

/* ------------------------------------------------------------------
   Area that will hold Event Block and all Name/Value Pairs
   ------------------------------------------------------------------ */
Dcl stgarea     Area;                /* stgarea is 1000 bytes long  */

/*====================================================================

 Main Program

 ====================================================================*/

/* ------------------------------------------------------------------
   Setup Name/Value Pair for Host
   ------------------------------------------------------------------ */
Allocate cytanval In(stgarea);        /* Allocate storage           */
hostaddr = cytanvalp;                 /* Save the address           */
cytannam = Addr(hostlbl);             /* Name                       */
cytanvl  = Addr(hostnam);             /* Value                      */
cytanvll = Length(hostnam);           /* Value Length               */

/* ------------------------------------------------------------------
   Setup Name/Value Pair for Component
   ------------------------------------------------------------------ */
Allocate cytanval In(stgarea);        /* Allocate storage           */
compaddr = cytanvalp;                 /* Save the address           */
cytannam = Addr(complbl);             /* Name                       */
cytanvl  = Addr(compnam);             /* Value                      */
cytanvll = Length(compnam);           /* Value Length               */
hostaddr->cytannxt = compaddr;        /* Chain off Host Pair        */

/* ------------------------------------------------------------------
   Setup Name/Value Pair for Application
   ------------------------------------------------------------------ */
Allocate cytanval In(stgarea);        /* Allocate storage           */
appladdr = cytanvalp;                 /* Save the address           */
cytannam = Addr(appllbl);             /* Name                       */
cytanvl  = Addr(applnam);             /* Value                      */
```

```
cytanvll = Length(applnam);           /* Value Length               */
compaddr->cytannxt = appladdr;        /* Chain off Host Pair        */

/* ------------------------------------------------------------------
   Setup Name/Value Pair for Transaction
   NB: Because Transaction is a number, we set the flag so that this
      value is NOT translated from EBCDIC to ASCII.
   ------------------------------------------------------------------ */
Allocate cytanval In(stgarea);        /* Allocate storage           */
tranaddr = cytanvalp;                 /* Save the address           */
cytannam = Addr(tranlbl);             /* Name                       */
cytanvl  = Addr(trannam);             /* Value                      */
cytannvx = '1'B;                      /* Do NOT xlate Value         */
cytanvll = 4;                         /* Value Length               */
appladdr->cytannxt = tranaddr;        /* Chain off Host Pair        */

/* ------------------------------------------------------------------
   Get our Configuration Token
   ------------------------------------------------------------------ */
Allocate cytacfg In(stgarea);         /* Allocate stg for cfg block  */
cytacsrv = Addr(server);              /* Server                     */
Call CYTA_init(cytacfg);
Display ('(CYTAPSMP) CYTAINIT Return Code=' || PLIRETV());

/* ------------------------------------------------------------------
   Send a Started Event
   ------------------------------------------------------------------ */
Allocate cytaevnt In(stgarea);        /* Allocate storage for event  */
cytaetyp = cytaesta;                  /* Started Event Type         */
cytaevli = Addr(vlink);               /* Vertical Link ID           */
cytaevll = Length(vlink);             /* Vertical Link Length       */
cytaevcn = hostaddr;                  /* Vertical Context Start     */
Call CYTA_track(cytacfg, cytaevnt);
Display ('(CYTAPSMP) STARTED Event Return Code=' || PLIRETV());

/* ------------------------------------------------------------------
   Send an Outbound Event
   (we only need to specify changed fields - all other fields remain
   from the Started event)
   ------------------------------------------------------------------ */
cytaetyp = cytaeob;                   /* Oubound Event Type         */
cytaehli = Addr(hlinkout);            /* Horizontal Link ID         */
cytaehll = Length(hlinkout);          /* Horizontal Link Length     */
Call CYTA_track(cytacfg, cytaevnt);
Display ('(CYTAPSMP) OUTBOUND Event Return Code=' || PLIRETV());

/* ------------------------------------------------------------------
   Send an Inbound Finished Event
   ------------------------------------------------------------------ */
cytaetyp = cytaeibf;                  /* Oubound Event Type         */
cytaehli = Addr(hlinkin);             /* Horizontal Link ID         */
cytaehll = 4;                         /* Horizontal Link Length     */
cytaehnx = '1'b;                      /* Link ID is a number, so do  */
                                      /* NOT translate from EBCDIC.  */
Call CYTA_track(cytacfg, cytaevnt);
Display ('(CYTAPSMP) INBOUND FINISHED Event Return Code=' ||
        PLIRETV());

/* ------------------------------------------------------------------
   Free up our storage
   ------------------------------------------------------------------ */
stgarea = Empty();                    /* Free all storage           */

End CYTAPSMP;
```

HLASM

```
*====================================================================*
* Main Program
*====================================================================*
CYTAASM  RSECT
CYTAASM  AMODE 31
CYTAASM  RMODE ANY
         BAKR  R14,0
         LR    R12,R15
         USING CYTAASM,R12

*--------------------------------------------------------------------*
* Setup workarea
*--------------------------------------------------------------------*
         STORAGE OBTAIN,LENGTH=@WORKL,ADDR=(R1)
         ST    R1,8(R13)              New savearea ptr back to caller
         ST    R13,4(R1)              Old save area ptr in new
         LR    R13,R1
         USING WORK,R13
         MVC   SAVEAREA+4,=C'F1SA'    Show we are using linkage stack

*--------------------------------------------------------------------*
* Setup Vertical Context
*    Your event needs some basic values here, so we use the CYTADFV
*    to fill them in. We chain these values of our own IDNum entry.
*    Note that IDNum is a number, so we don't translate it from EBCDIC
*    to ASCII before sending it down.
*--------------------------------------------------------------------*
         LA    R1,#IDNUM
         CYTANV NAME='IDNum',VALUE=(R1),LEN=#IDNUML,XLATEV=NO,         x
               MF=(E,NAMVALV1)
         CYTADFV TXN='CYTAASM',CHAINTO=NAMVALV1,MF=(E,NAMVALVC)

*--------------------------------------------------------------------*
* Get a Configuration Token - we will use the subsystem CYTZ.        *
*--------------------------------------------------------------------*
         CYTAINIT SUB='CYTZ'          Get Token
         LTR   R15,R15                If successful
         BNZ   LEAVEX
         ST    R1,TTOKEN                 Save it

*--------------------------------------------------------------------*
* Always start with a STARTED Event
* The Vertical Link should be the same for all events in this work
* unit until a FINISHED event is sent.
* Set the entire event block to zeroes before filling it in.
*--------------------------------------------------------------------*
         XC    EVENTBLK(@EVENTBLK),EVENTBLK
         CYTATRAK STARTED,                                          X
               VCTXT=NAMVALV1,                                      X
               TOKEN=TTOKEN,                                        X
               VLINK='CYTAASM',                                     X
               MF=(E,EVENTBLK)
         LTR   R15,R15                If not successful
         BNZ   LEAVEX                 Exit

*--------------------------------------------------------------------*
* Send an Outbound Event - this indicates that we've sent something
* to someone else.
* Remember we need a Horizontal Link and/or a Horizontal Stitch
* here - and the program on the other end also needs to specify the
* same link/stitch on their Inbound event.
* Remember also that this event remembers all the values from the
* previous CYTATRAK call. So we only need to specify the Event Type,
* and the values we are overriding (in this case, HLINK).
*--------------------------------------------------------------------*
         CYTATRAK OUTBOUND,HLINK='CYTAOUT',TOKEN=TTOKEN,MF=(E,EVENTBLK)
```

```
         LTR   R15,R15                If not successful
         BNZ   LEAVEX                 Exit

*--------------------------------------------------------------------*
* (often a program would wait here for something to come back)
*--------------------------------------------------------------------*

*--------------------------------------------------------------------*
* Send an Inbound Finished Event - this is two events (Inbound and
* Finished) rolled into one.
* We also need a Horizontal Link and/or a Horizontal Stitch here, but
* not the one we sent - the one that the program at the other end has
* specified.
* Remember that we also finish with a FINISHED event.
*--------------------------------------------------------------------*
         CYTATRAK INBOUND_FIN,HLINK='CYTABACK',TOKEN=TTOKEN,           X
               MF=(E,EVENTBLK)
         LTR   R15,R15                If not successful
         BNZ   LEAVEX                 Exit

*--------------------------------------------------------------------*
* Return to Caller
*--------------------------------------------------------------------*
LEAVE    DS    0H
         LR    R4,R15                 Save return code
         STORAGE RELEASE,LENGTH=@WORKL,ADDR=(R13) Release workarea
         LR    R15,R4
         PR                           Return to caller

*--------------------------------------------------------------------*
* Error Routine - Error occurred. Exit with return code
*--------------------------------------------------------------------*
LEAVEX   DS    0H
         B     LEAVE

*====================================================================
*
*  PROGRAM CONSTANTS AND LITERALS
*
*====================================================================
#IDNUM   DC    F'45'
#IDNUML  DC    AL2(L'#IDNUM)
         LTORG

*====================================================================
*
*  Mapping Macros and DSECTs
*
*====================================================================
*--------------------------------------------------------------------*
* Workarea
*--------------------------------------------------------------------*
WORK     DSECT
SAVEAREA  DS    18F                   Savearea
STCKTIME  DS    D                     STCK Timestamp

TTOKEN    DS    F                     Fullword for Config Token
NAMVALVC  CYTADFV MF=L                Vertical Ctxt Nam/Val Pairs
NAMVALV1  CYTANV MF=L                 Extra Vert Context Nam/Val pair
EVENTBLK  CYTATRAK MF=L               Event Block
@EVENTBLK EQU  *-EVENTBLK             Length of Event Block

@WORKL    EQU   *-WORK                Length of Workarea

*--------------------------------------------------------------------*
* Register Equates
*--------------------------------------------------------------------*
```

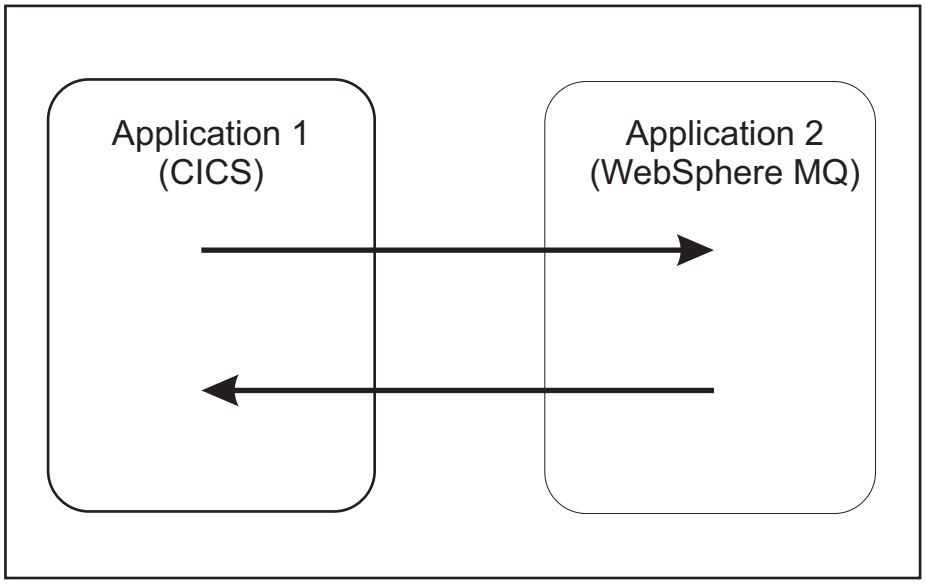
```
R0        EQU   0
R1        EQU   1
R2        EQU   2
R3        EQU   3
R4        EQU   4
R5        EQU   5
R6        EQU   6
R7        EQU   7
R8        EQU   8
R9        EQU   9
R10       EQU   10
R11       EQU   11
R12       EQU   12
R13       EQU   13
R14       EQU   14
R15       EQU   15

          END
```

Appendix D. kto_stitching file

The kto_stitching.xml file defines how horizontal and vertical stitching occurs for events. It consists of definitions for Stitch Pairs.

Consider the following example. A CICS region and a Websphere MQ application communicate to each other. A monitor in CICS and a monitor in Websphere MQ both send Transactions events.



For Transactions to be able to stitch the CICS and Websphere MQ events together, it must know which Stitching name/value pairs must be matched. For example, CICS and WebSphere® MQ may send a ServerName stitching name/value pair. The Stitch Pair entry in kto_stitching.xml tells the Transaction Collector that a CICS event (type 8) ServerName field must match a Websphere MQ event (type 5) ServerName field for two events to be eligible for stitching. The kto_stitching.xml file resides in the Transaction Collector directory.

Format

kto_stitching.xml is an XML file with the following format:

```
TTEMA Stitching
  Stitch Criteria
   StitchPair
    StitchName
  Stitch Priorities
```

TTEMA Stitching defines the beginning of Transactions stitching definitions. It has no fields.

Stitch Criteria defines the beginning of a list of Stitch Pairs. It has no fields.

StitchPair defines global parameters for the Stitch Pair entry. It has the following fields:

- **Name** – name of the Stitch Pair – can be any string.
- **horizontal** – `true` if this pattern is for matching horizontal stitches. Default: `false`.
- **vertical** – `true` if this pattern is for matching vertical stitches. Default: `false`.
- **reflective** – `true` if this pattern is to be applied for messages sent both ways. If false, then this Stitch Pair will only apply to events sent from the first caller to the second caller. Default: `false`.

StitchNameList defines the beginning of a list of stitching pairs to be matched. It has the following fields:

- **caller** – caller type for the following pairs. Must be a valid link type, that is, a number between 0 and 255. See the Building and Event section for a list of link types.

StitchName defines a pair to be matched. It has the following fields:

- **name** – the name of a stitching name/value pair. Remember that this is case sensitive.

Example

The following is an example of a kto_stitching.xml file:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<TTEMAStitching>
        <StitchCriteria>
                <StitchPair name="MQCICS" horizontal="true" reflective="true">
                        <StitchNameList caller="5">
                                <StitchName name="ServerName" />
                                <StitchName name="QMgrName" />
                                <StitchName name="ObjName" />
                                <StitchName name="MsgId" />
                                <StitchName name="CorrelId" />
                                <StitchName name="PutDate" />
                                <StitchName name="PutTime" />
                        </StitchNameList>
                        <StitchNameList caller="8">
                                <StitchName name="ServerName" />
                                <StitchName name="qmgr" />
                                <StitchName name="rslvdQueue" />
                                <StitchName name="msgId" />
                                <StitchName name="corrId" />
                                <StitchName name="putdate" />
                                <StitchName name="puttime" />
                        </StitchNameList>
                </StitchPair>
        </StitchCriteria>
</TTEMAStitching>
```

This example file defines how ITCAM for MQ Tracking and ITCAM for CICS events will be stitched. In this example, *all* the fields in Table 6 must match fully for a stitch to occur.

Table 6. Field matching

ITCAM for MQ Tracking	ITCAM for CICS
ServerName	ServerName
QMgrName	qmgr
ObjName	rslvdQueue
MsgId	msgId
CorrelId	corrId
PutDate	putdate

Table 6. Field matching (continued)

ITCAM for MQ Tracking	ITCAM for CICS
PutTime	puttime

Event flows in both directions (MQ to CICS and CICS to MQ) are covered by this rule.

Appendix E. Transaction Collector Context Mask

The Transaction Collector performs aggregation of events based on context.

For example, all events must specify the ServerName in the vertical context. The Transaction Collector aggregates all events by ServerName, and this is displayed in the Transactions workspaces.

The contexts that the Transaction Collectors aggregate are specified in a Context Mask file. This file is a text file that resides in the Transaction Collector directory. The format of this file is as follows:

```
# Comments begin in column 1. Do not use blank spaces.
# Don't leave blank lines
#
# Use the Compare statement to tell the Transactions Collector to # aggregate.
# On the line following, specify the name of the context to aggregate on.
# Finish the statement with a * in column 1.
# For example:
compare
ApplicationName
*
# The above statement tells Transactions Collector to
# aggregate on ApplicationName context values. You
# cannot use wildcards – specify the complete context
# name.
#
# You can also use the ignore statement. This tells the
# Transactions Collector not to aggregate. For example:
ignore
UserID
*
# The above statement tells the Transactions Collector NOT
# to aggregate on UserID.
# This is the default – all context values are ignored
# unless specified in a compare statement.
```

Specify the name of the Context Mask file during installation of the Transaction Collector, and if reconfigured, in the **Transaction Collector Configuration** dialog box.

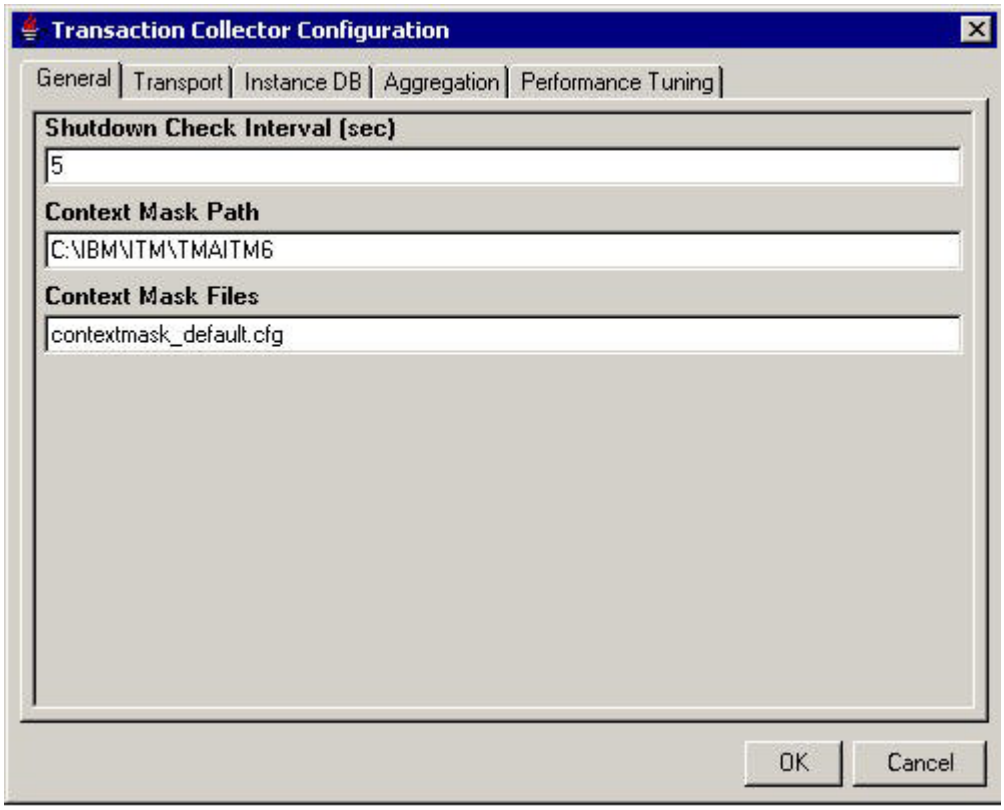


Figure 4. Transaction Collector Configuration dialog box

In Figure 4, the Context Mask file name and location is C:\IBM\ITM\TMAITM6\contextmask_default.cfg. This is the default Context Mask supplied with ITCAM for Transaction Tracking. More than one file can be specified in the **Context Mask Files** field: separate each file name with a semi-colon (;).

Appendix F. Accessibility

Accessibility features help users with physical disabilities, such as restricted mobility or limited vision, to use software products successfully.

The major accessibility features in this product enable users to do the following:

- Use assistive technologies, such as screen-reader software and digital speech synthesizer, to hear what is displayed on the screen. Consult the product documentation of the assistive technology for details on using those technologies with this product.
- Operate specific or equivalent features using only the keyboard.
- Magnify what is displayed on the screen.

In addition, the product documentation was modified to include the following features to aid accessibility:

- All documentation is available in both HTML and convertible PDF formats to give the maximum opportunity for users to apply screen-reader software.
- All images in the documentation are provided with alternative text so that users with vision impairments can understand the contents of the images.

Navigating the interface using the keyboard

Standard shortcut and accelerator keys are used by the product and are documented by the operating system. Refer to the documentation provided by your operating system for more information.

Magnifying what is displayed on the screen

You can enlarge information on the product windows using facilities provided by the operating systems on which the product is run. For example, in a Microsoft Windows environment, you can lower the resolution of the screen to enlarge the font sizes of the text on the screen. Refer to the documentation provided by your operating system for more information.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing 2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who want to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
2Z4A/101
11400 Burnet Road Austin,
TX 78758
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at http://www.ibm.com/legal/copytrade.shtml.

Adobe®, the Adobe logo, PostScript®, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Microsoft, Windows, Windows NT®, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java, JavaScript™, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Glossary

agent Software installed on systems you want to monitor. The agent collects data about an operating system, a subsystem, or an application.

aggregate
An average of all response time detected by the monitoring software over a specific time period.

alert A message indicating an event has occurred. Alerts have associated predefined severity levels. They are listed in order of increasing severity: **Informational**, **Warning**, and **Critical**. Classifying alerts into severity levels provides the ability to focus on the most severe problems first.

application
Represents the business process being monitored, such as Lotus Notes or SAP.

application pattern
Determines what transactions to monitor and how to group them.

ARM An application that passes information about subtransactions between business applications across a network. The Application Response Measurement is designed to instrument a unit of work, such as a business transaction, that is sensitive to performance time. The transaction is something that must be measured and monitored and for which a corrective action can be taken. See http://www.opengroup.org/tech/management/arm/ for further information.

ARM response codes
The ARM response codes have the following definitions

0 The ARM call was successful.
-249 An unspecific error occurred while processing an ARM call.
-1000 Invalid parameter passed to ARM.
-1001 Transaction name passed to ARM was an empty string.
-1002 Transaction Name passed to ARM contains too many characters.
-1003 Error performing character set conversion.
-1100 Application ID passed to ARM is not valid.
-1101 Output application ID passed to ARM is not valid. (may be null)
-1102 Output application handle passed to ARM is not valid. (may be null)
-1200 Transaction ID passed to ARM is not valid.
-1201 Transaction ID passed to ARM is not valid. (may be null)
-1202 Transaction status passed to ARM is not valid.
-1203 Output Transaction ID passed to ARM is not valid. (may be null)
-1204 Output Transaction Handle passed to ARM is not valid. (may be null)
-1300 Metric ID passed to ARM is not valid.
-1301 Metric Format passed to ARM is not valid.
-1302 Metric Usage passed to ARM is not valid.
-1303 Metric Unit passed to ARM is not valid.
-1304 Output Metric ID passed to ARM is not valid. (may be null)
-1305 Metric name is not valid, must not start with ARM:
-1400 Correlator passed to ARM is not valid.
-1401 Flag number passed to ARM is zero. It is 1 (`app trace flag`) or 2 (`agent trace flag`).
-1402 Flag number passed to ARM is not valid.
-1500 Charset passed in to ARM is not supported.
-1501 An error occurred that was not predefined by ARM.
-2000 Arm engine is not currently running.
-2001 Communication is not possible from `libarm` to the ARM engine.
-2002 Communication is not possible from the ARM engine to `libarm`.
-50000 Transaction is not registered.
-50001 Application is not registered.
-50002 Registered application ID is not valid.
-50003 Started transaction handle is not valid.

Additional Java error codes:

-15001 Loading of the Native Library failed, `libarmjni4` not found
-15051 Size of identity names array does not match size of values array
-15052 Identity name index out of bounds
-15053 Identity name array is null
-15054 Identity value index out of bounds
-15055 identity value array is null

-15056 context name index out of bounds
-15057 context name array is null
-15101 Identity Properties is not valid
-15151 `ApplicationDefinition` is null
-15152 `ApplicationDefinition` is not valid
-15153 context value index out of bounds
-15154 context value array is null
-15201 Application object is null
-15202 Transaction Definition is null
-15203 Context value index out of bounds
-15204 context values array is null
-15205 Transaction context sub-buffer will not be created
-15206 Transaction is inactive
-15207 Transaction is active
-15208 Unblock called without block
-15208 Unbind called without bind
-15251 `ApplicationDefinition` is null
-15252 `ApplicationDefinition` is not valid
-15253 Identity Properties is not valid
-15301 Metric Group definition index out of bounds
-15302 Metric Group definition array is null
-15351 Metric group index out of bounds
-15352 Metric group array is null
-15353 Metric group array entry is null
-15354 Metric group array is not valid
-15355 Metric group array and `MetricGroupDefinition` mismatch
-15356 `MetricGroupDefinition` is null
-15357 `MetricGroupDefinition` is not valid
-15401 `ApplicationDefinition` is null
-15402 `ApplicationDefinition` is not valid
-15451 `ArmMetricDefinition` is null
-15452 `ArmMetricDefinition` is not valid
-15501 `ArmMetricDefinitionGroup` is not valid
-15551 `ArmTransactionWithMetricsDefinition` is not valid
-15601 `ArmTransactionWithMetricsDefinition` is not valid
-15651 `ArmMetricString` string is too long
-15701 Internal Error, `Arm40Token` constructor array is null.
-15702 Internal Error, `Arm40Token` Invalid input array
-15703 Internal Error, `Arm40Token` Invalid Offset
-15704 Internal Error, `Arm40Token` length error

attribute
Attributes specify a condition for a situation to monitor so it can provide an appropriate alert. For example, you can create situations that monitor for alerts with a specific severity. When the attribute alert values match the values specified in situations, the managed objects associated with the situations change appearance, alerting you to problems. Attributes are organized into attribute groups.

attribute group
A set of related attributes that can be combined in a data view or a situation. When you open the view or start the situation, data samples of the selected attributes are retrieved. The data samplings from an attribute group return either a single row of data or multiple rows. Each type of monitoring agent has a set of attribute groups. The software displays attributes groups in either a table or chart view.

availability
The successful execution of a monitored transaction over a specified period of time.

client
Represents an end user system that accesses a particular application to perform a transaction. The client is the system that initiated the request (or transaction).

client pattern
A method to define which clients to monitor by host name or IP address and to group them for reporting. Client patterns define groupings (based on IP and host name) of your end users (or clients) by location, or functional unit. You can use the client pattern to see how your application is performing from your customers' perspective.

client time
The time it takes to process and display the Web page on a browser.

composite aggregation algorithm
Used by the Transaction Reporter to track composite applications.

condition
A condition consists of an attribute, an operator (such as greater than or equal to), and a value. A condition can be read as `If - system condition - compared to - value - is true`.

configuration
(1) The manner in which the hardware and software of an information processing system are organized and interconnected. (2) The computers, devices, and programs that make up a system, subsystem, or network.

context
The means used to group the tracking data as part of a transaction flow.

current status
Indicates that data was gathered in the last 5 minutes.

data collector
The monitoring component that recorded the transaction data.

Data Collector plug-in
A combination of a Transaction Tracking API (Transaction Tracking Application Programming Interface) and its supporting files which, when installed on a domain, enables an application to transmit tracking data to a Transaction Collector.

data interval
A time period in minutes for the summary data record

down time
See also mean time to recovery.

edge
The point when the software first detects the transaction.

event A significant occurrence to a task or system that can be detected by a situation. Events include completion or failure of an operation, a user action, or the change in state of a process. The event causes the situation to become true and generates an alert.

failure
An individual instance of a transaction that did not complete correctly. See also incident.

horizontal
Associating tracking data between applications in a domain.

horizontal context
Identifies a transaction flow within a transaction and is used to group interactions based on the application supplying the tracking data.

host A computer that is connected to a network (such as the Internet or an SNA network) and provides an access point to that network. Also, depending on the environment, the host can provide centralized control of the network. The host can be a client, a server, or both a client and a server simultaneously.

HTTP response codes
Monitors for the receipt of the specified HTTP response codes. When a threshold has an HTTP response code transaction status, the software notifies you when the specified response codes are received from the Web server during monitoring.

400 Bad Request	This is a common response code that is generated when a URL is not found. This occurs when a server cannot be contacted or the resource does not exist. For example, if the Web server hosting `www.google.com` is down for service or a component upgrade and a user tries to view it, the user sees a 404 response. Often, a 404 response is presented in a user-friendly way by the Web browser. Microsoft Internet Explorer, for example, presents a special error page for a 404 response.
401 Unauthorization	410 Gone
402 Payment Required	411 Length Required
403 Forbidden	412 Precondition Failed
404 Not Found	413 Request Entity Too Large
405 Bad Method	414 Request-URI Too Long
406 None Acceptable	415 Unsupported Media Type
407 Proxy Authentication Required	416 Requested Range Not Satisfiable
408 Request Timeout	417 Expectation Failed

409 Conflict	
500 Internal Server Error	A common response code generated when the server is unable to service a request due to an internal server error. For example, the following sequence of events might produce this error: 1. A developer writes and deploys an application on `http://tests.mySoftwareDesignTestCases.org.` 2. The code on the accessed page divides by zero at some point, causing an internal application error. 3. The server returns a 500 response when it recognizes this condition.
502 Bad Gateway	504 Gateway Timeout
503 Service Unavailable	505 HTTP Version Not Supported

HTTP transaction
A single HTTP request, such as clicking a link, and an associated response, such as displaying a page.

incident
A failure or set of consecutive failures over a period of time without any successful transactions. An incident represents a period of time when the service was unavailable, down, or not functioning as expected.

instance
A single transaction or subtransaction.

instance algorithm
Used by the Transaction Reporter to track composite applications.

interval
The number of seconds that have elapsed between one sample and the next. A sample is the data that the product collects for the server.

load time
Time elapsed between the user's request and completion of the Web page download.

linking
Tracking transactions within the same domain or from collectors of the same type.

managed system
A particular operating system, subsystem, or application where a Tivoli Enterprise Monitoring Agent is installed and running.

Management Information Base (MIB)
(1) In the Simple Network Management Protocol (SNMP), a database of objects that can be queried or set by a network management system. (2) A definition for management information that specifies the information available from a host or gateway and the operations allowed.

mean time between failures (MTBF)
The average time in seconds between the recovery of one incident and the occurrence of the next one. This is also known as uptime.

mean time to recovery (MTTR)
The average number of seconds between an incident and service recovery; also known as downtime.

metrics aggregation
Used by the Transaction Collector to summarize tracking data using vertical linking and stitching to associate items for a particular transaction instance. Ensures that all appropriate tracking data is aggregated.

MIB See Management Information Base.

monitor
An entity that performs measurements to collect data about performance, availability, reliability, or other attributes of applications or the systems on which the applications rely. These measurements can be compared to predefined thresholds. If a threshold is exceeded, administrators can be notified, or predefined automated responses can be performed.

monitoring schedule
A schedule that determines on which days and at what times the monitors collect data.

Navigator
The left pane of theTivoli Enterprise

Portal window. The Navigator Physical view shows your network enterprise as a physical hierarchy of systems grouped by platform. You can also create other views to create logical hierarchies grouped as you specify, such as by department or function.

network time
Time spent transmitting all required data through the network.

over time interval
The number of minutes the software aggregates data before writing out a data point.

pattern
Patterns group the data into manageable pieces.

platform
The operating system the managed system is using, such as OS/390 and Linux. The Navigator physical mapping places the platform level under the enterprise level.

probe A schedule that determines on which days and at what times the monitors collect data.

profile element
An element or monitoring task belonging to a user profile. The element defines what is to be monitored and when.

query A combination of statements used to search a repository for systems that meet certain criteria. The query object is created in a query library.

realm A password-protected area of a Web site.

request
See transaction.

response time
Time elapsed between the user's request and the completion of the transaction.

round-trip response time
The time it takes to complete the entire page request. Round-trip time includes server time, client, and network and data transfer time.

robotic scripts
Are recordings of typical customer transactions that collect performance data. The performance data helps determine whether a transaction is performing as expected and exposes problem areas of the Web and application environment. Robotic scripts include: CLI Playback, Rational Robot GUI or VU, Mercury LoadRunner, Rational Performance Tester (RPT).

SAF See Store and Forward.

schedule
Determines how frequently a situation runs with user-defined start times, stop times, and parameters.

SDK Software Development Kit.

server Represents the physical system that is running the application process. It is the system that processed the request (transaction).

server time
The time it takes for a Web server to receive a requested transaction, process it, and respond to it.

service
A set of business processes (such as Web transactions) that represent business critical functions that are made available over the internet.

service level agreement (SLA)
A contract between a customer and a service provider that specifies expectations about the level of service, particularly availability, performance, and other measurable objectives.

service level classifications
Rules that are used by monitors to evaluate how well a monitored service is performing. The results form the basis for service level agreements. See also service level agreement.

service recovery
The time it takes for the service to recover from being in a failed state.

situation
A set of conditions that, when met, creates an event. An example of a situation is: `IF - CPU usage - > - 90% - TRUE`. The expression `CPU usage > 90%` is the situation condition.

SLA See service level agreement.

status Describes the state of a transaction at a particular point in time, such as whether it failed, was successful, or slow.

stitching
Tracking transactions between domains or from different types of collectors.

Store and Forward (SAF)
A file that stores all information if the system is unavailable to forward the information. As soon as the system is available again, the information in the SAF file is processed and forwarded to the appropriate system.

subtransaction
An individual step (such as a single page request or logging on to a Web application) in the overall recorded transaction. See transaction. When a transaction is considered together with its subtransactions, it is called a *parent* transaction. The subtransactions are *children* of the parent transaction.

summary data
Display details about the response times and volume history, as well as total times and counts of successful transactions for the whole application.

summary interval
The number of hours data is stored on the agent for display in the Tivoli Data Warehouse workspaces.

summary status
An amount of time (default is 8 hours) to collect data on the Tivoli Enterprise Monitoring Agent

Tivoli Data Warehouse
Stores historical data collected from agents in your environment. The data warehouse is located on a DB2, Oracle, or Microsoft SQL database. To collect information to store in this database, you must install the Warehouse Proxy agent. To perform aggregation and pruning functions on the data, install the Warehouse Summarization and Pruning agent.

Tivoli Enterprise Console (TEC)
Synchronizes the status of situation events that are forwarded to the event server. When the status of an event is updated because of Tivoli Enterprise Console rules or operator actions, the update is sent to the monitoring server, and the updated status is reflected in both the Situation Event Console and the Tivoli Enterprise Console event viewer.

Tivoli Enterprise Monitoring Agent (TEMA)
This is a product-specific agent providing data for IBM Tivoli Monitoring. The monitoring agents are installed on the systems or subsystems you want to monitor. These monitoring agents collect and distribute data to a monitoring server.

Tivoli Enterprise Monitoring Server (TEMS)
This is the host data management component for IBM Tivoli Monitoring. It acts as a collection and control point for alerts received from the agents, and collects their performance and availability data. There are two types of monitoring servers: hub and remote.

Tivoli Enterprise Portal (TEP or portal)
A Java-based user interface for viewing and monitoring the enterprise. It provides two modes of operation: desktop and browser.

thresholds
Customizable values for defining the acceptable tolerance limits (maximum, minimum, or reference limit) for a transaction. When the measured value of the resource is greater than the maximum value, less than the minimum value, or equal to the reference value, the software records an incident.

tracking data
Information emitted by composite applications when a transaction instance occurs.

Tracking Data Store
Stores tracking data provided by the Data Collector plug-in.

transaction
An exchange that accomplishes a particular action or result. A transaction can occur between a workstation and a program, two workstations, or two programs. A recorded transaction consists of one or more subtransactions. When a transaction is considered together with its subtransactions, it is called a *parent* transaction. The subtransactions are *children* of the parent transaction.

transaction flow
The common path through a composite application taken by similar transaction instances.

transaction instance
Interactions between composite applications in response to an external stimuus such as a request.

transaction pattern
The pattern for specifying the name of specific transactions that you want to monitor. Patterns define groupings of transactions that map to business applications and business transactions.

Transaction Collector
Stores the tracking data from multiple Data Collector plug-ins and computes aggregates. Also called TTAS monitoring agent.

Transaction Reporter
Stores the aggregated data from the Transaction Collector and sends this data to the Tivoli Enterprise Portal workspaces.

Transaction Tracking API (Transaction Tracking Application Programming Interface or TTAPI)
Installed on supported domains, the Transaction Tracking API enables an application to transmit tracking data to a Transaction Collector.

Transactions Base
Component of ITCAM for Transaction Tracking for z/OS consisting of the Transactions Container and Transactions Dispatcher.

Transactions Container
z/OS started task (STC) that provides basic functionality for ITCAM for Transaction Tracking.

Transactions Dispatcher
Code that processes event queues on z/OS within the Transactions Container and sends events to the Transaction Collector on Windows or UNIX.

Transactions MQ Container
z/OS started task (STC) that provides basic functionality for MQ events.

Transactions MQ Courier
Code that processes event queues on z/OS and sends events to the Transaction Collector on Windows or UNIX.

Transactions MQ Dispatcher
Code that runs within the Transactions MQ Container and provides an operator command interface.

Transactions MQ Exits
MQ exit code that creates events for MQ and sends these events to the Transactions MQ Dispatcher using the Transaction Tracking API.

Transactions TEMA
See Transaction Reporter.

TTAS TEMA
See Transaction Collector.

trend Data graphed over time to show a general tendency in timings or attributes.

UNIX epoch
00:00:00 UTC, January 1, 1970.

uptime
See Mean Time Between Failure.

URL Universal Resource Locator. The unique address for a file accessible through the Internet. Such a file might be a Web page (usually the home page), an image file, or a program such as a Java applet or servlet. The URL comprises the protocol used to access the file, a domain name that identifies a specific computer on the Internet, and a path name that specifies that file's location on that computer.

user profile (ISM)
An entity such as a department or customer for whom services are being performed.

vertical
Associating individual tracking data within an application within a domain.

vertical context
Used within an application or group of applications to distinguish one transaction flow from another. The vertical context enables Transaction Tracking to group individual transactions as part of a flow, label a node in a topology map, and link to an IBM® Tivoli Monitoring application.

view A logical table that consists of data generated by a query. A view is based on an underlying set of base tables, and the data in a view is determined by a SELECT statement that is run on the base tables.

volume
The number of requests.

workspace
The viewing area of the Tivoli Enterprise Portal window, excluding the Navigator.

Workspaces contain the table views to obtain information about, manage, and utilize log files more effectively. The table views and graphs in each workspace report attribute information you are monitoring. You can use them to perform the following tasks:

- Investigate attribute information relating to a change in state
- Monitor system performance to identify bottlenecks and to evaluate tuning decisions
- Select the most effective threshold values for situations you create

A workspace also might contain other views, such as a notepad pane, a browser session, an event console, or a take action pane that provides the ability to send commands to the operator console.

Index

A

accessibility 69
aggregation 18
Application context 19
applications, instrumenting 13
association IDs 15

B

binding on z/OS 8
blocking events 21
  example 21
books, see publications ix, x

C

compiling 8
Component context 19
contextual information 19
conventions, typeface xi
creating events 7
CYTADFV 35
CYTAINIT 37
CYTANV 38
CYTATOK 39
CYTATRAK 40

D

data types 27
debugging errors 10
detecting errors 10
directory names, notation xii

E

environment variables 10
environment variables, notation xii
environment, preparing to install 5
errors
  debugging 10
  handling 10
  logging 10
event data structures 27
events
  blocking 21
  blocking, example 21
  characteristics 13
  context 19
  correlation 18
  creating and sending 7
  invalid 10
  types 14
  workspaces 19
  z/OS 22
example
  blocking events 21
executing 8

F

FINISHED event type 14
functions, high level language 23

H

HERE event type 14
high level language
  functions 23
  reference 23
HLASM 35
horizontal linking 15
Host context 19

I

identifiers
  instance 18
  uniqueness 15
INBOUND event type 14
INBOUND_FINISHED event type 14
include files 8
instrumenting
  applications 13
  synchronous transactions 15
introduction 1

J

Java API reference 33
Javadoc for TTAPI4J 33

K

KBB_RAS1 environment variable 10
KBB_RAS1_LOG environment variable 10
KBB_VARPREFIX environment variable 10

L

languages, supported 3
linking IDs 15
linking on distributed platforms 8
logging
  environment variables 10
  errors 10

M

manuals, see publications ix, x

N

notation
  environment variables xii
  path names xii
  typeface xii
notation *(continued)*

O

online publications, accessing x
ordering publications x
OUTBOUND event type 14
OUTBOUND_FINISHED event type 14

P

path names, notation xii
platforms, supported 3
preparing environment 5
program requirements 8
publications ix
  accessing online x
  ordering x

R

reference, high level language 23
running Transaction Tracking API 8

S

sample
  CYTADFV 35
  CYTAINIT 37
  CYTANV 38
  CYTATOK 39
  CYTATRAK 40
sending events 7
shutting down Transaction Tracking API 7
STARTED event type 14
STARTED_INBOUND event type 14
stitching IDs 15
support xi
supported
  languages 3
  platforms 3
synchronous transactions 21

T

Tivoli software information center x
transaction
  data 18
  ID 18
  synchronous 15, 21
Transaction context 19
tt_association_id_t data structure 27
tt_event_type_t data structure 27
tt_instance_id_t data structure 27
tt_time_t data structure 27
tt_values_list_t data structure 27

TTAPI4J 33
typeface conventions xi

V

variables, notation for xii
vertical stitching 15

Z

z/OS
  binding 8
  event data 22

Printed in USA

SC23-9755-00