

Netcool/OMNIbus  
Version 7 Release 3

## *Event Integration Facility Reference*





Netcool/OMNIbus  
Version 7 Release 3

## *Event Integration Facility Reference*



**Note**

Before using this information and the product it supports, read the information in “Notices” on page 73.

This edition applies to version 7, release 3, modification 1 of IBM Tivoli Netcool/OMNIBus (product number 5724-S44) and to all subsequent releases and modifications until otherwise indicated in new editions.

© **Copyright IBM Corporation 2003, 2011.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

---

# Contents

<b>About this publication</b>	<b>v</b>
Intended audience	v
What this publication contains	v
Publications	vi
Accessibility	viii
Tivoli technical training	viii
Support information	viii
Conventions used in this publication	ix

## Chapter 1. Overview of the Tivoli Event Integration Facility . . . . . 1

Events	1
Adapters	2
Event classes	3
Configuration files	3
Event server	3
Event filtering	6
Rules	6

## Chapter 2. Installing the Tivoli Event Integration Facility . . . . . 9

Preparing for installation	9
Installing	10
Migrating adapters	12

## Chapter 3. Event transport . . . . . 13

Event delivery methods	13
Connection options	13
Transport options	13
Shell script and test options	17
Event reception for applications	20
Sending events through firewalls	21
Event delivery when systems fail	21
Activating the cache	22
Configuring backup servers to deliver events	23
Using the portmapper keywords	24
Configuring a reception application built with the C API	24

## Chapter 4. Building an adapter . . . . . 27

Adapter files	27
Identifying events to monitor	27
Defining the source	28
Defining event classes	28
Selecting event delivery methods	29
Configuring an EIF receiver application for SSL	29
Configuring an EIF client application for SSL	31
Programming the adapter	33
Upgrading existing adapters	34
Configuration file APIs	34
Communications APIs	34
Special considerations for Microsoft Windows	35
Compiling the adapter built with the C API	35
Linking the adapter built with the C API	36

Installing, configuring, and testing the adapter	36
Running adapters built with the Event Integration Facility Java API	37
Configuring adapters for international environments	37

## Chapter 5. Filtering events at the source . . . . . 39

Filtering with configuration files	39
Filtering events when systems fail	40
Regular expressions in filters	41

## Chapter 6. Troubleshooting . . . . . 43

Message logs	43
Trace logs	43
Performance and availability	44
Event reception connection parameters	44
Common problems and scenarios	45
Building and running adapters	45
Making connections to the event server	45
Sending events	46

## Appendix A. Application programming interfaces . . . . . 47

C language API	47
tec_agent_getenv	47
tec_agent_init	47
tec_create{EIF_handle	48
tec_create_handle	49
tec_create_handle_c	50
tec_create_handle_r	51
tec_destroy_handle	52
tec_errno	52
tec_get_event	52
tec_put_event	53
tec_register_callback	53

## Appendix B. Utilities for the C API . . . 55

ed_scan_get_n	55
ed_scan_n	55
ed_sleep	56

## Appendix C. Java language API . . . . . 57

disconnect	57
disconnect(time)	57
getConfigVal	57
onMessage	58
receiveEvent	58
registerListener	59
sendEvent	59
TECAgent	59
TECEvent	60

## Appendix D. Keywords . . . . . 63

<b>Notices</b>	<b>73</b>
Trademarks	75

<b>Index</b>	<b>77</b>
--------------	-----------

---

## About this publication

The Tivoli Event Integration Facility Reference contains reference material for the Event Integration Facility toolkit (EIF).

---

## Intended audience

This guide explains the concepts to effectively develop new adapters or modify existing ones. This book is for developers who have programming knowledge and need to create custom adapters and use the Tivoli Event Integration Facility within their applications. This book is also useful for Tivoli Netcool/OMNIBus administrators who modify configuration files.

Readers should be familiar with the following software:

- Java or C programming languages
- UNIX, Microsoft Windows, or other target operating systems

---

## What this publication contains

This publication contains reference information for the Tivoli Event Integration Facility.

This publication contains the following sections:

- Chapter 1, "Overview of the Tivoli Event Integration Facility," on page 1  
Introduces the Event Integration Facility.
- Chapter 2, "Installing the Tivoli Event Integration Facility," on page 9  
Describes planning, installing and migrating for the Event Integration Facility.
- Chapter 3, "Event transport," on page 13  
Describes methods for sending information to the event server.
- Chapter 4, "Building an adapter," on page 27  
Describes how to build an adapter to help you monitor events.
- Chapter 5, "Filtering events at the source," on page 39  
Describes how to filter events at the source to deal with large numbers of events.
- Chapter 6, "Troubleshooting," on page 43  
Explains how to troubleshoot problems that can arise installing and using the Event Integration Facility.
- Appendix A, "Application programming interfaces," on page 47  
Describes how to use APIs to build custom adapters or applications.
- Appendix D, "Keywords," on page 63  
Contains the keywords common to most adapters.

---

## Publications

This section lists publications in the Tivoli Netcool/OMNIbus library and related documents. The section also describes how to access Tivoli publications online and how to order Tivoli publications.

### Your Tivoli Netcool/OMNIbus library

The following documents are available in the Tivoli Netcool/OMNIbus library:

- *IBM Tivoli Netcool/OMNIbus Installation and Deployment Guide*, SC14-7604  
Includes installation and upgrade procedures for Tivoli Netcool/OMNIbus, and describes how to configure security and component communications. The publication also includes examples of Tivoli Netcool/OMNIbus architectures and describes how to implement them.
- *IBM Tivoli Netcool/OMNIbus Administration Guide*, SC14-7605  
Describes how to perform administrative tasks using the Tivoli Netcool/OMNIbus Administrator GUI, command-line tools, and process control. The publication also contains descriptions and examples of ObjectServer SQL syntax and automations.
- *IBM Tivoli Netcool/OMNIbus Web GUI Administration and User's Guide*, SC14-7606  
Describes how to perform administrative and event visualization tasks using the Tivoli Netcool/OMNIbus Web GUI.
- *IBM Tivoli Netcool/OMNIbus User's Guide*, SC14-7607  
Provides an overview of the desktop tools and describes the operator tasks related to event management using these tools.
- *IBM Tivoli Netcool/OMNIbus Probe and Gateway Guide*, SC14-7608  
Contains introductory and reference information about probes and gateways, including probe rules file syntax and gateway commands.
- *IBM Tivoli Monitoring for Tivoli Netcool/OMNIbus Agent User's Guide*, SC14-7610  
Describes how to install the health monitoring agent for Tivoli Netcool/OMNIbus and contains reference information about the agent.
- *IBM Tivoli Netcool/OMNIbus Event Integration Facility Reference*, SC14-7611  
Describes how to develop event adapters that are tailored to your network environment and the specific needs of your enterprise. This publication also describes how to filter events at the source.
- *IBM Tivoli Netcool/OMNIbus Error Messages Guide*, SC14-7612  
Describes system messages in Tivoli Netcool/OMNIbus and how to respond to those messages.
- *IBM Tivoli Netcool/OMNIbus Web GUI Administration API (WAAPI) User's Guide*, SC22-5403-00  
Shows how to administer the Tivoli Netcool/OMNIbus Web GUI using the XML application programming interface named WAAPI.

### Related publications

The following documents should be consulted when using the IBM® Tivoli Enterprise Console® as event server. They are available in the IBM Tivoli Enterprise Console library:

- *IBM Tivoli Enterprise Console Adapters Guide*, SC32-1242  
Provides information about supported adapters, including how to install and configure these adapters.



- *IBM Tivoli Enterprise Console Installation Guide*, SC32–1233  
Describes how to install, upgrade, and uninstall the IBM Tivoli Enterprise Console product.
- *IBM Tivoli Enterprise Console Command and Task Reference*, SC32–1232  
Provides details about IBM Tivoli Enterprise Console commands, predefined tasks shipped in the task library, and the environment variables that are available to tasks that run against an event.
- *IBM Tivoli Enterprise Console Rule Developer's Guide*, SC32–1234  
Describes how to develop rules and integrate them for event correlation and automated event management.
- *IBM Tivoli Enterprise Console User's Guide*, SC32–1235  
Provides an overview of the IBM Tivoli Enterprise Console product and describes how to configure and use the IBM Tivoli Enterprise Console product to manage events.
- *IBM Tivoli Enterprise Console Warehouse Enablement Pack: Implementation Guide*, SC32–1236  
Describes how to install and configure the warehouse enablement pack for the IBM Tivoli Enterprise Console product and describes the data flow and structures that are used by the warehouse pack.
- *IBM Tivoli Event Console Release Notes*, SC32–1238  
Provides release-specific information that is not available until just before the product is sent to market.
- *IBM Tivoli Enterprise Console Rule Set Reference*, SC32–1282  
Provides reference information about the IBM Tivoli Enterprise Console rule sets.

## 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 Information Center Web site at:

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

**Note:** If you print PDF documents on other than letter-sized paper, set the option in the **File > Print** window that allows Adobe Reader to print letter-sized pages on your local paper.

## Ordering publications

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

<http://www.elink.ibm.link.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, perform the following steps:

1. Go to the following Web site:  
<http://www.elink.ibm.link.ibm.com/publications/servlet/pbi.wss>
2. Select your country from the list and click **Go**. The Welcome to the IBM Publications Center page is displayed for your country.
3. On the left side of the page, click **About this site** to see an information page that includes the telephone number of your local representative.

---

## Accessibility

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

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 some features of the graphical user interface.

---

## Tivoli technical training

For Tivoli technical training information, 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 publication

This publication uses several conventions for special terms and actions and operating system-dependent commands and paths.

### Typeface conventions

This publication 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*

- Citations (examples: titles of publications, diskettes, and CDs)
- Words defined in text (example: a nonswitched line is called a *point-to-point* line)
- Emphasis of words and letters (words as words example: "Use the word *that* to introduce a restrictive clause."; letters as letters example: "The LUN address must start with the letter *L*.")
- New terms in text (except in a definition list): a *view* is a frame in a workspace that contains data
- Variables and values you must provide: ... where *myname* represents....

#### **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 publication uses the UNIX 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. For example, on UNIX systems, the *\$NCHOME* environment variable specifies the path of the Netcool® home directory. On Windows systems, the *%NCHOME%* environment variable specifies the path of the Netcool home directory. 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.

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



---

## Chapter 1. Overview of the Tivoli Event Integration Facility

The Event Integration Facility is a toolkit that expands the types of events and system information that you can monitor. You can use it to develop your own adapters, tailored to your network environment and to your specific needs.

The Event Integration Facility contains:

- An event application programming interface (API) library for use with the Java and C programming languages.
- A debugging function that checks event syntax and sends events to a file.

You can use Tivoli Event Integration Facility to do the following tasks:

- Specify the event information to send to the event server for processing.
- Create an adapter to filter, translate, and then forward event information to the event server.
- Filter and correlate events near the source.
- Create an application that can receive events.

---

### Events

The central unit of information is the event. An event is any significant change in the state of a system resource or application. Events can be generated for problems and for successful completions of tasks.

Events provide many different types of information, for example when a host is down, when someone unsuccessfully tries to log in to a host as an administrator, or when a hard drive is nearly full. Events can also be generated to clear other events.

An event begins as an error message, a trap, or a similar piece of information that is displayed or written to a file. The information provided in an event is dependent on the source. Some sources provide detailed information about an event, while other sources are brief in their descriptions. An adapter converts all events into a format that provides consistency in information, such as the source, location, date, and time of an event.

Some events, such as traps, contain data that you cannot read. In these cases, the adapter must translate the data into a format that you can read so that the information can be further processed. Some sources, such as a system log file, provide data that you can read without machine translation.

Adapters translate event information into a set of attributes. Each attribute is predefined by the adapter and contains the attribute name and the attribute value. The adapter places the appropriate information in each attribute, and then sends the event to the event server.

This following figure illustrates how an event evolves:

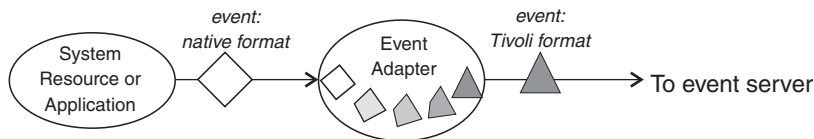


Figure 1. Evolution of an event

The following example shows how log file information is translated into events. In this example, a failed attempt to run the **su root** command on the host oak is written to the system log file. You can read the resulting format:

```
Nov 7 08:51:42 oak su: 'su root' failed for don on /dev/ttyp0
```

The adapter then translates the log file information into an event as follows:

```
Su_Failure:
source=LOGFILE;
origin=oak;
date="Nov 7 08:51:42 ";
host=oak;
sub_source=login;
from_user=don;
tty=/dev/ttyp0
to_user=root;
END
```

#### Related concepts

“Event server” on page 3

## Adapters

Adapters are processes that monitor managed sources. A source is an application such as a database, or a system resource such as disk space. When an adapter receives information from its source, the adapter formats the information and forwards it to the event server.

To monitor a source such as a third-party or custom application, you must use the Event Integration Facility to create an adapter and event classes for each new source.

Adapters monitor sources in the following ways:

- An adapter can receive messages from a source that actively produces messages. For example, adapters can receive messages that are sent by Tivoli software applications.
- An adapter can check a file at configurable intervals if the source updates a file.
- An adapter can poll a system resource or system condition at configurable intervals, and then interpret and forward the resulting information directly to the event server.

---

## Event classes

Event classes are classifications of events. After separating information into event classes, adapters format the information into messages that represent specific instances of event classes. Then they send the information to the event server, which processes the information, along with information received from other adapters.

Event classes can be divided into subclasses to facilitate a further breakdown of information so that more detailed rules can be applied to the event information. Essentially, event classes are an agreement between the adapter and the event server about what information the adapter sends to the event server.

**Note:** Event classes are not the same as Tivoli objects.

### Related concepts

“Defining event classes” on page 28

---

## Configuration files

You can control the behavior of an adapter using the configuration file. A configuration file enables you to specify configuration parameters.

You can specify the following configuration parameters:

- The information that must be sent to the event server.
- The connection interface used by the adapter.
- The host on which the event server is located.
- The event filtering to be performed by the adapter.
- Any information that is unique to a particular adapter.

After information is separated into event classes, the adapter sends the information to the event server for further processing. You can use the configuration file to control the information that the adapter sends to the event server in order to reduce the load on the network.

You do not need to modify multiple instances of an adapter to run in different environments. You have to modify only the configuration files.

### Related reference

Appendix D, “Keywords,” on page 63

---

## Event server

An event server is a central server that handles all events collected by the distributed adapters. The event server creates an entry for each incoming event and evaluates that event against a set of rules to determine if it can respond to the event, or modify the event automatically.

The Probe for Tivoli EIF acts as the event server for the Tivoli Event Integration Facility.

**Note:** For previous versions of EIF, the Tivoli Enterprise Console Server fulfills the same role.

An adapter does not typically provide values for all attributes of a particular event; some attribute values are provided by the event server. The Probe for Tivoli EIF can process any well-formed EIF event. You have the option to define the following attributes:

**\$ClassName**

The class name of the event.

**\$EventSeqNo**

The sequence number of the event.

**\$EventString**

The entire event in a single string.

**\$date** The date of the event in mm/dd/yy format.

**\$EventClass**

The class of the event as assigned by the event source.

**Note:** The Probe for Tivoli EIF uses the event class to identify the format of the message for each event type.

**\$hostname**

The host name of the system.

**\$msg** The content of the message.

**\$peerhost**

The secondary host.

**\$severity**

The severity of the alarm.

**\$source**

The type of application that has created the event.

**\$sub\_origin**

The optional subcategorization of the event origin.

**\$sub\_source**

A detailed description of the source.

Rules describe the actions that are performed when the event server receives a particular system event. In some cases, you can customize other applications to receive events from the event server.

The event server validates incoming events. If an event is valid, the event server assigns a unique identification (ID) and time stamp and stores it in an event database. The event server processes each incoming transaction before processing the next transaction.

The following example event has been processed by the Probe for Tivoli EIF and the Tivoli Netcool/OMNIBus ObjectServer:

```
Node Alias: 9.42.19.243
TECEventHandle:
Process Required: 0
Prec. Entity ID: 0
First Occurrence: 07/28/2009 02:17:23 PM
Rem. Sec Obj.:
Port: 0
Suppr./Esc1.: Normal
Rem. Root Obj.:
Extended Attributes:
```



Slot: 0  
 Local Node Alias:  
 Managed Status: Managed  
 Alert Group: EVENT  
 Class: TME10tecad  
 Flash: No  
 URL:  
 Summary: hello\_eif\_probe  
 Cause Type: Unknown  
 Count: 1  
 Poll: 0  
 Agent: TEC  
 State Change: 07/28/2009 02:17:23 PM  
 Alert Key: TEC  
 Local Sec. Obj.:  
 TaskList: Not in Task List  
 Local Root Obj.:  
 Group: Public  
 Card:  
 Serial: 1950  
 Manager: tivoli\_eif probe on austin.tivlab.raleigh.ibm.com  
 Event Type: Not Defined  
 Rem. Pri. Obj.:  
 Customer:  
 Expire Time: Not Set  
 Identifier: :TEC:EVENT  
 Prec. Obj. Inst.: 0  
 TECRepeatCount: 0  
 Correlated Notif.:  
 TECFQHostname:  
 Specific Problem:  
 Owner: Nobody  
 Location:  
 Server Name: NCOMS  
 Service:  
 Internal Timestamp: 07/28/2009 02:17:23 PM  
 Node: 9.42.19.243  
 TECStatus:  
 TECDate:  
 Probable Cause: Not Defined  
 Rem. Node Alias:  
 Severity: Indeterminate  
 Last Occurrence: 07/28/2009 02:17:23 PM  
 TECHostname:  
 Grade: 1  
 Server Serial: 1950  
 TECDateReception:  
 Local Pri. Obj.:  
 TECServerHandle:  
 Precision Domain:  
 Type: Problem  
 Event ID:  
 Prec. Serial:  
 Ack: No

The following example event has been processed by the Tivoli Enterprise Console Server:

```

Su_Failure:
  server_handle=1;
  date_reception=784408852;
  event_handle=1;
  source=LOG;
  sub_source=login;
  origin=oak;
  sub_origin='';
  hostname='';
  
```

```

last_modified_time='Nov 07, 1994 08:51:42';
adapter_host='';
status=OPEN;
administrator='';
acl=[admin];
severity=WARNING;
date='Nov 07, 1994 08:51:42';
duration=0;
msg='';
msg_catalog='';
msg_index=0;
num_actions=1;
credibility=0;
repeat_count=0;
cause_date_reception=0;
cause_event_handle=0;
from_user=don;
to_user=root;
END

```

#### Related concepts

“Events” on page 1

---

## Event filtering

Event filtering reduces complexity for console operators and improves response times for complex system errors.

You can filter events with the Tivoli Event Integration Facility by defining filter statements in the configuration file.

#### Related concepts

Chapter 5, “Filtering events at the source,” on page 39

---

## Rules

The event server uses rules to specify and control automatic responses to events throughout an enterprise. If you develop a new adapter, you can write new rules specifically geared toward enhancing the usefulness of the adapter events.

IBM Tivoli Netcool/OMNIBus automations are similar to EIF rules. The automations detect changes in the ObjectServer and respond to these changes automatically, enabling the ObjectServer to process alerts without requiring an operator to take action. For more information about Tivoli Netcool/OMNIBus automations, see the *IBM Tivoli Netcool/OMNIBus Administration Guide* in the Network Availability Management information center at <http://publib.boulder.ibm.com/infocenter/tivihelp/v8r1/index.jsp>

For more information about the Probe for Tivoli EIF rules, see the following publications in the IBM Tivoli Network Availability Management information center at <http://publib.boulder.ibm.com/infocenter/tivihelp/v8r1/index.jsp>:

- The *IBM Tivoli Netcool/OMNIBus Probe and Gateway Guide* at [http://publib.boulder.ibm.com/infocenter/tivihelp/v8r1/topic/com.ibm.netcool\\_OMNIBus\\_probes.doc/welcome\\_genprb.htm](http://publib.boulder.ibm.com/infocenter/tivihelp/v8r1/topic/com.ibm.netcool_OMNIBus_probes.doc/welcome_genprb.htm)
- The *IBM Tivoli Netcool/OMNIBus Probe for Tivoli EIF Reference Guide* at [http://publib.boulder.ibm.com/infocenter/tivihelp/v8r1/topic/com.ibm.netcool\\_OMNIBus.doc/probes/tivoli\\_eif/tivoli\\_eif/wip/concept/tveif\\_intro.html](http://publib.boulder.ibm.com/infocenter/tivihelp/v8r1/topic/com.ibm.netcool_OMNIBus.doc/probes/tivoli_eif/tivoli_eif/wip/concept/tveif_intro.html)

For more information about Tivoli Enterprise Console rules, see the *IBM Tivoli Enterprise Console Rule Developer's Guide* in the IBM Tivoli Enterprise Console information center at  
<http://publib.boulder.ibm.com/infocenter/tivihelp/v3r1/topic/com.ibm.itecruledev.doc/ecodmst.htm>



---

## Chapter 2. Installing the Tivoli Event Integration Facility

You install the Tivoli Event Integration Facility using a stand-alone installation process. Custom adapters created with a previous version of the Event Integration Facility can be migrated only when the SOCKET transport type is used.

---

### Preparing for installation

The Tivoli Event Integration Facility SDK is distributed with IBM Tivoli Netcool/OMNIbus as a compressed file. This file is available on CD, or you can download it from the IBM Passport Advantage® Online Web site as part of the main Tivoli Netcool/OMNIbus download.

**Note:** The same version of EIF, which is suitable for all operating systems, is available for download with all versions of Tivoli Netcool/OMNIbus.

Extract the contents of the installation package into a temporary location. Extract the EIF files to a separate directory from your IBM Tivoli installation directory.

**Note:** You do not use the Tivoli Netcool/OMNIbus installation process to install the Tivoli Event Integration Facility. Instead, you merely extract the compressed EIF files that are bundled with the Tivoli Netcool/OMNIbus files.

**Note:** The Tivoli Netcool/OMNIbus version of EIF is compiled using Microsoft Visual C++ 2005 SP1. Therefore EIF executables such as posteifmsg may require the Microsoft Visual C++ 2005 SP1 Redistributable Package to be installed. The user or application deploying EIF must install the Redistributable Package, because EIF does not have an installer program and is not installed by Tivoli Netcool/OMNIbus. You can download the Redistributable Package from the following locations:

#### For x86 operating systems

<http://www.microsoft.com/downloads/details.aspx?familyid=200b2fd9-ae1a-4a14-984d-389c36f85647>

#### For x64 operating systems

<http://www.microsoft.com/downloads/details.aspx?familyid=EB4EBE2D-33C0-4A47-9DD4-B9A6D7BD44DA>

### Downloading EIF

If you are downloading EIF from the IBM Passport Advantage Online Web site, follow the instructions in the download document for your operating system.

*Table 1. Location of the download documents for all supported Tivoli Netcool/OMNIbus operating systems*

Operating system	Download document location
AIX®	<a href="http://www-01.ibm.com/support/docview.wss?rs=3120&amp;uid=swg24026995">http://www-01.ibm.com/support/docview.wss?rs=3120&amp;uid=swg24026995</a>
HP-UX	<a href="http://www-01.ibm.com/support/docview.wss?rs=3120&amp;uid=swg24026996">http://www-01.ibm.com/support/docview.wss?rs=3120&amp;uid=swg24026996</a>
HP-UX Integrity	<a href="http://www-01.ibm.com/support/docview.wss?rs=3120&amp;uid=swg24026999">http://www-01.ibm.com/support/docview.wss?rs=3120&amp;uid=swg24026999</a>

Table 1. Location of the download documents for all supported Tivoli Netcool/OMNIBus operating systems (continued)

Operating system	Download document location
Linux	<a href="http://www-01.ibm.com/support/docview.wss?rs=3120&amp;uid=swg24026997">http://www-01.ibm.com/support/docview.wss?rs=3120&amp;uid=swg24026997</a>
Linux for System z®	<a href="http://www-01.ibm.com/support/docview.wss?rs=3120&amp;uid=swg24026998">http://www-01.ibm.com/support/docview.wss?rs=3120&amp;uid=swg24026998</a>
Solaris	<a href="http://www-01.ibm.com/support/docview.wss?rs=3120&amp;uid=swg24027000">http://www-01.ibm.com/support/docview.wss?rs=3120&amp;uid=swg24027000</a>
Windows	<a href="http://www-01.ibm.com/support/docview.wss?rs=3120&amp;uid=swg24027001">http://www-01.ibm.com/support/docview.wss?rs=3120&amp;uid=swg24027001</a>

## Installing

You do not use the Tivoli Netcool/OMNIBus installation process to install the Tivoli Event Integration Facility. Instead, you extract the compressed EIF files that are bundled with the Tivoli Netcool/OMNIBus files. The EIF SDK includes the JAR files and C libraries.

### Directory structures

EIF has the file and directory structure described in the following list. In this directory structure, *interp* maps to the following names:

- aix4-r1
- hpux10
- hpuxia
- linux-ix86
- linux-ppc
- linux-s390
- linux-ia64
- os390
- solaris2
- solaris2-ix86
- w32-ix86

#### **EIF.conf**

A sample EIF configuration file.

#### **ed\_diag\_config**

A sample configuration file to enable EIF diagnostics.

#### **EIF.version**

A file containing the current build version of EIF.

#### **bin/interp**

Contains, for each *interp*, the 32-bit command line tool for sending EIF events:

- posteifmsg

#### **bin64/interp**

Contains, for each *interp*, the 64-bit command line tool for sending EIF events:

- posteifmsg

**contrib/interp**

Contains compiled sample C programs (32-bit) for each supported interp.

Sample C-based receivers:

- eifrcv1
- eifrcv2
- eifrcv3
- eifrcv4

Sample C-based senders:

- eifsend1
- eifsend2

**contrib64/interp**

Contains compiled sample C programs (64-bit) for each supported interp:

Sample C-based receivers:

- eifrcv1
- eifrcv2
- eifrcv3
- eifrcv4

Sample C-based senders:

- eifsend1
- eifsend2

**include**

Contains header files (.h) for building adapters. These are common across all interps:

- agent\_comm.h
- tec\_defines.h
- tec\_eif.h
- tec\_eEIF.h

**jars**

Contains all the jar files necessary to use the Java based EIF API:

- evd.jar
- log.jar

**javadoc**

Contains the javadoc for the Java-based EIF API.

**lib/interp**

Contains, for each interp type, the 32-bit static library needed for linking an adapter:

- libeif.a

**lib64/interp**

Contains, for each interp type, the 64-bit static library needed for linking an adapter:

- libeif.a

**samples**

Contains sample adapter source code.

Sample C-based receivers:

- eifrcv1.c

- eifrcv2.c
- eifrcv3.c
- eifrcv4.c

Sample C-based senders:

- eifsend1.c
- eifsend2.c

Sample C-based long running adapter:

- sampleAdapter.c

Sample Java-based long running adapter:

- SampleAdapter.java

---

## Migrating adapters

The new Event Integration Facility is backwardly compatible with earlier versions only when the SOCKET transport type is used.

### Java API

Applications that are migrating to the new Event Integration Facility must include the following new jar files.

- evd.jar
- log.jar

**C API** Applications that are migrating to the new Event Integration Facility must relink any binaries that are dependent on the Event Integration Facility. This means a static link to libeif.a (previously libteceef.a).

### Related concepts

Chapter 4, “Building an adapter,” on page 27



---

## Chapter 3. Event transport

The Tivoli Event Integration Facility provides several methods for sending information to the event server. The system on which an adapter is run must provide a TCP/IP-based interprocess communication facility.

---

### Event delivery methods

Events are delivered by an interprocess communication mechanism. To specify the delivery methods for adapters and applications using the Tivoli Event Integration Facility, modify the configuration file and link to the applicable library. To deliver test events directly from a command prompt, use the command-line interface.

#### Related concepts

“Selecting event delivery methods” on page 29

#### Related reference

“Shell script and test options” on page 17

### Connection options

The connection options are either connection-oriented or connectionless. In situations where you want to send many events, you use the connection-oriented option. In situations where you want to send few events over a period of time, you use the connectionless option.

You can write a single adapter and use it with a connection-oriented or a connectionless delivery method. You can specify a delivery method by modifying the configuration file.

- The default setting for the connection option is connectionless. This method establishes and discards a new connection for each event or group of events.
- For a connection-oriented event delivery, specify `ConnectionMode=C0` in the configuration file. This method keeps the channel to the event server open, which improves the performance when you are sending many events.

#### Related concepts

“Selecting event delivery methods” on page 29

#### Related reference

Appendix D, “Keywords,” on page 63

### Transport options

An application can use the Event Integration Facility API to act as an event sender or an event receiver. The transport options for connections are either SOCKET or Secure Sockets Layer (SSL), and are defined in the EIF configuration file.

#### **SOCKET**

Compatible with both IPv4 and IPv6.

Links to the `libeif.a` library.

Uses standard TCP/IP to establish connections.

#### **SSL**      Compatible with both IPv4 and IPv6.

Links to the `libeif.a` library.

Performs the standard SSL handshake internally.

Can run in FIPS 140-2 mode.

Requires SSL keystores and truststores from the application.

**Note:** The keystores and truststores contain the digital certificates and keys that are required to establish an SSL connection. Use the iKeyman utility provided with GSKit to create these. GSKit is installed in the following IBM Tivoli Netcool/OMNIBus location: `$NCHOME/platform/arch/lib`, with *arch* being your operating system directory.

#### Related concepts

“Federal Information Processing Standard 140–2 (FIPS 140–2) support” on page 17

“Selecting event delivery methods” on page 29

#### Related reference

“Linking the adapter built with the C API” on page 36

## SSL and FIPS 140-2

The Tivoli Event Integration Facility supports the use of the Secure Sockets Layer (SSL) encryption and authentication protocol to send and receive events. In addition, EIF SSL connections can operate in FIPS 140-2 mode, which means the use of FIPS 140-2 approved cryptographic providers.

SSL uses digital certificates for key exchange and authentication. When a client initiates an SSL connection, the server presents the client with a certificate that is signed by a Certificate Authority (CA); that is, a trusted party that guarantees the identity of the certificate and its creator. The server certificate contains the identity of the server, the public key, and the digital signature of the certificate issuer.

To enable FIPS 140-2 mode, edit the EIF configuration file and set the **channel\_nameSSLFIPSMODE** property to 0N. For Java-based EIF applications, you must also configure the Java Runtime Environment for FIPS 140-2 mode. When in FIPS 140-2 mode, all encryption and key generation functions that are required for the secured SSL connections are provided by FIPS 140-2 approved cryptographic providers. If an EIF receiver application has been enabled for FIPS 140-2 mode, all EIF client programs that connect to the receiver must also be FIPS-enabled.

For the Java version of EIF, FIPS 140-2 mode is facilitated by the use of IBM Java Runtime Environment (JRE) 1.4.2 or higher.

For the C version of EIF, FIPS 140-2 mode is facilitated by the use of IBM Global Security Kit (GSKit) version 7.

**Restriction:** SSL support is unavailable in the EIF C API for platforms not supported by GSKit.

Before using SSL in the C version of EIF, ensure that the path to the GSKit libraries appears in the following environment variables:

- On Windows operating systems, the path to the GSKit libraries must appear in the PATH environment variable.

**Note:** The GSKit libraries are located in `%NCHOME%\platform\win32\lib`

- On UNIX and Linux operating systems, the path to the GSKit libraries must appear in the LIBPATH, SHLIB\_PATH, or LD\_LIBRARY\_PATH environment variables.

**Note:** The GSKit libraries are located in `$NCHOME/platform/arch/lib` with *arch* being your operating system directory.

## IBM Key Management utility (iKeyman)

You configure transport types of either SOCKET or SSL in the EIF configuration file. If you choose SSL, you use iKeyman (provided with GSKit version 7) to generate and manage keys and digital certificates that are required for SSL communication.

**Note:** You can also manage keys and digital certificates from a command-line interface.

### Keystores for EIF receiver applications

Use iKeyman to create keystores for EIF receiver applications.

The keystore of the receiver is a key database, and its default personal certificate will be presented by the EIF receiver to EIF clients during an SSL connection. This certificate can be either a self-signed certificate that you create using iKeyman, or a certificate obtained from and signed by a Certificate Authority (CA). The certificate must be set as the default personal certificate in the key database of the receiver.

**Note:** The Java version of EIF uses the JKS key database format, whereas the C version of EIF uses the CMS format.

Use the EIF configuration keyword (`channel_nameSSLKeystore`) to configure the EIF receiver to use a key database as its keystore. In situations where the EIF receiver application uses the configuration keyword (`channel_nameSSLRequireClientAuthentication=YES`) clients are also required to present a certificate during an SSL connection. The keystore of the receiver must then contain not only the personal certificate of the receiver, but also the default personal certificates of any trusted clients. Import these trusted certificates into the keystore of the receiver using iKeyman.

### Keystores for EIF client applications

Use iKeyman to create keystores for EIF client applications.

The keystore of the client is a key database containing the default personal certificates of any EIF receiver applications that are trusted by the EIF client. These trusted certificates must be imported into the keystore of the client using iKeyman in order for the client to connect to the EIF receiver using SSL.

**Note:** The Java version of EIF uses the JKS key database format, whereas the C version of EIF uses the CMS format.

Use the EIF configuration keyword (`channel_nameSSLKeystore`) to configure an EIF client application to use a key database as its keystore. In situations where an EIF receiver application uses the configuration (`channel_nameSSLRequireClientAuthentication=YES`), clients are required to present a certificate during an SSL connection. The keystore of the client must then contain not only the trusted receiver certificates, but also a separate certificate that will identify the client to the receiver. This new certificate can either be self-signed or signed by a CA, and it must be the default certificate in the keystore of the client.

## Truststore

Truststores are key databases containing certificates that are trusted by an EIF application.

Any EIF application (whether client or receiver), may use a single key database to hold both trusted certificates and the default personal certificate of the application. Such a database would be configured in EIF using the `channel_nameSSLKeystore` keyword.

A Java EIF application, however, also has the option of using one key database to hold trusted certificates, and a second key database to hold the default personal certificate of the application. Use the `channel_nameSSLTruststore` keyword to specify the key database containing the trusted certificates, and the `channel_nameSSLKeystore` keyword to specify the key database containing the default personal certificate of the application.

## Stash file

Applies to EIF applications created using the C API only.

If you require automatic login to gain access to the digital certificates, you can save the password for a key or trust database in encrypted format to a stash file. Whenever the keystore or truststore is accessed, the system checks whether a stash file exists. If found, the file contents are decrypted and used as input for the password.

**Note:** To create and store encrypted passwords for applications created using the Java API, use the `com.tivoli.tec.event_delivery.common.Encryption` script.

## Ciphers

The `SSLCipherList` keyword specifies the ciphers that will be permitted during SSL authentication. The sending and receiving ends of the EIF connection must have at least one cipher in common in order for the connection to succeed. FIPS 140-2 mode restricts the ciphers that are allowed for both C and Java versions of EIF. If no ciphers are specified or restricted, all available ciphers are permitted.

**The following values are valid for C EIF applications:**

`SSL_RC2_CBC_128_CBC_WITH_MD5`  
`SSL_RC2_CBC_128_CBC_EXPORT40_WITH_MD5`  
`SSL_DES_64_CBC_WITH_MD5`  
`SSL_DES_192_EDE3_CBC_WITH_MD5`  
`SSL_NULL_WITH_NULL_NULL`  
`SSL_RSA_WITH_NULL_MD5`  
`SSL_RSA_WITH_NULL_SHA`  
`SSL_RSA_EXPORT_WITH_RC4_40_MD5`  
`SSL_RSA_WITH_RC4_128_MD5`  
`SSL_RSA_WITH_RC4_128_SHA`  
`SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5`  
`SSL_RSA_WITH_DES_CBC_SHA`

**The following values are valid for Java EIF applications:**

Any cipher supported by the IBM JRE is valid.

For a full list of supported ciphers, see the Java Secure Socket Extension (JSSE) IBMJSSE2 Provider Reference Guide for the Java 2 SDK, Standard Edition, Version 5 at:  
<http://www.ibm.com/developerworks/java/jdk/security/50/secguides/jsse2Docs/JSSE2RefGuide.html>

#### **Related concepts**

“Selecting event delivery methods” on page 29

#### **Related tasks**

“Configuring an EIF client application for SSL” on page 31

“Configuring an EIF receiver application for SSL” on page 29

#### **Related reference**

“Shell script and test options”

Appendix D, “Keywords,” on page 63

### **Federal Information Processing Standard 140–2 (FIPS 140–2) support**

Federal Information Processing Standards (FIPS) are standards and guidelines that the National Institute of Standards and Technology (NIST) issues for use in United States federal government computer systems.

The Federal Information Processing Standard 140–2 (FIPS 140–2) defines security requirements for cryptographic modules that are used to protect sensitive information in computer and telecommunication systems.

Tivoli Event Integration Facility uses the FIPS 140–2 approved cryptographic providers, IBMJCEFIPS (certificate 376) or IBMJSSEFIPS (certificate 409), and IBM Crypto for C (ICC) (certificate 384), for cryptography. The certificates are listed on the NIST Web site at <http://csrc.nist.gov/cryptval/140-1/1401val2004.htm>.

The FIPS 140–2 approved cryptographic providers provide both cryptographic functions and Secure Sockets Layer (SSL) data protection, on both client and server applications. When Tivoli Event Integration Facility is running in FIPS 140–2 mode, all encryption and key generation functions are provided by the FIPS 140–2 approved cryptographic modules.

#### **Related concepts**

“Selecting event delivery methods” on page 29

#### **Related reference**

“Transport options” on page 13

## **Shell script and test options**

Several CLI commands and a Java class are available for sending events.

You can send events manually for the following reasons:

- To troubleshoot event delivery problems after installing a new adapter
- To use shell scripts in order to develop adapters
- To test the adapter after you have:
  - Created new event groups and assignments
  - Edited rules
  - Changed the way a server processes events

**Note:** When using a user ID other than Administrator or root, ensure that you have the correct permissions for creating the file specified by the BufEvtPath keyword (as well as the default value when the keyword is not specified).

You can use `postEIFmsg` to post an event to the event server, as demonstrated in the following example:

```
postEIFmsg -S server | -f configuration_file [-m message] [-r severity]
[attribute=value...] class source
```

You can use `com.tivoli.tec.event_delivery.TECAgent` to post an event to the event server, as demonstrated in the following example:

```
postEIFmsg -S server | -f configuration_file [-m message] [-r severity]
[attribute=value...] class source
```

You can use `com.tivoli.tec.event_delivery.common.Encryption` to create an encryption key for SSL communication, as demonstrated in the following example:

```
java -cp <path to evd.jar> com.tivoli.tec.event_delivery.common.Encryption
createkey [-l key length]
```

Valid key lengths are 128, 192 and 256. The default value if you do not specify a value is 128. The `-f` option turns on FIPS 140-2 mode. Both the EIF sender and the EIF receiver must be using FIPS for FIPS 140-2 mode to function.

When creating a stash file for the Java version of EIF, you first create the encryption key file and then the stash file. You use `createKey` to generate the encryption key file, and `encrypt` to generate the stash file. When a stash file is used, an encryption key file is required.

**Note:** For the C version of EIF, use `iKeyman` to create the stash file.

The following command-line script creates an encryption key and saves it to a file:

**Example syntax**

```
java -cp <path to evd.jar>
com.tivoli.tec.event_delivery.common.Encryption createkey [-l key
length] -o output_file_path [-f]
```

**Example script**

```
java -cp ./evd.jar com.tivoli.tec.event_delivery.common.Encryption
createkey -l 128 -o ./mykey
```

The following command line script uses the encryption key to encrypt the password and saves the encrypted data to a stash file:

**Example syntax**

```
java -cp <path to evd.jar>
com.tivoli.tec.event_delivery.common.Encryption encrypt -k
encryption_key_file_path -o output_file_path -d text_to_encrypt
[-f]
```

**Example script**

```
java -cp ./evd.jar com.tivoli.tec.event_delivery.common.Encryption
encrypt -k ./mykey -o ./mypass -d password
```

### Related concepts

“Event delivery methods” on page 13

### Related tasks

“Programming the adapter” on page 33

### Related reference

“SSL and FIPS 140-2” on page 14

“Installing, configuring, and testing the adapter” on page 36

## Setting up the posteifmsg utility on z/OS using USS

Proceed as follows to set up the **postEIFMSG** utility on z/OS® using USS.

1. From the Tivoli Netcool/OMNIBus EIFSDK directory, copy the `bin/os390/postEIFMSG` file to a location where the command will be run.
2. Set up the appropriate `postEIFMSG` configuration file. For example:

```
TransportList=t1
t1_Channels=c1
c1_ServerLocation=<IP_Address_Goes_Here>
c1_Port=<EventServerListeningPortGoesHere>
t1_Type=SOCKET
```

```
BufEvtPath=/tmp/postEIFMSG.cache
```

3. In your USS environment, create the `/etc/Tivoli/codeset` directory. This directory contains the EBCDIC codeset files extracted from the Tivoli Netcool/OMNIBus EIFSDK directory.

**Tip:** If you downloaded the files to another system, use FTP to transfer the EBCDIC codeset file or files in binary mode to your USS system, and copy them to the `/etc/Tivoli/codeset` directory.

The file names correspond to the codepage number associated with the codeset. The following codepages are provided for EBCDIC:

- 37
- 273
- 274
- 277
- 278
- 280
- 282
- 284
- 297
- 424
- 500
- 870
- 875
- 933
- 935
- 937
- 939
- 1025
- 1026
- 1047

- 1112
- 1122
- 1388

The codepages correspond to the EBCDIC codesets used for Western, East European, Middle Eastern, and Asian languages. The default language is English (codeset 1047). The **postEIFmsg** utility uses only the files it requires. Therefore you can transfer all the codeset files to your USS environment.

4. Run the following command to set the **TISDIR** environment variable to the correct value:  
`export TISDIR=/etc/Tivoli` The **postEIFmsg** utility can now access the Tivoli® Management Framework (TMF), locate the codeset file associated with your system environment, and then convert it to UTF-8.
5. Run the **postEIFmsg** command to send events. For example:  
`postEIFmsg -f nameOfConfigFile -m testMessage EVENT TESTIN`

---

## Event reception for applications

In addition to sending events, the Tivoli Event Integration Facility enables other applications to receive, that is, listen for, events.

The Event Integration Facility can instantiate one or more event listeners. Each event listener can have one or more channels. This allows information to flow from multiple sources. You can specify multiple channels, and the event listener listens to all of those channels.

The polling mechanism enables the application to retrieve events synchronously, by using the **get** method of the API. With the non-polling mechanism, the application registers a listener or callback and receives events asynchronously.

Additionally, EIF uses a cache for event reception. If the application has a listener, EIF also registers a listener in the cache. Then it notifies the application and passes the event to the application listener. If the application does not use a listener, the application must request the retrieval of the next event.

The following is an example of a configuration file that enables the application to receive events using sockets:

```
BufferEvents=YES
BufEvtPath=/tmp/eif_socket_rcv.cache
TransportList=t1
t1Type=SOCKET
t1Channels=t_
t_ServerLocation=myserver.com
t_Port=5151
```

The following is an example of a configuration file that enables an application created using the C API to receive events using SSL, with FIPS 140-2 mode enabled:

```
TransportList=t1_
t1_Type=SSL
t1_Channels=c1_
c1_Port=3443

c1_ServerLocation=myserver.com

c1_SSLKeystore=/my/location/gbkeys.kdb
```



```
c1_SSLKeystorePW=password  
c1_SSLEncipherList=SSL_RSA_WITH_3DES_EDE_CBC_SHA
```

```
c1_SSLEnforceMode=ON
```

**Related tasks**

“Activating the cache” on page 22

**Related reference**

Appendix D, “Keywords,” on page 63

---

## Sending events through firewalls

You can send events through firewalls depending on your environment and organizational restrictions regarding firewall security.

To send events through firewalls, use Tivoli Management Framework Firewall Security Toolbox. It collects events and sends them across the firewall using a proxy. To obtain Tivoli Management Framework Firewall Security Toolbox and its documentation, contact Customer Support.

To send events through firewalls in non-Tivoli environments, you configure your firewall to allow arbitrary TCP/IP connections on the port specified for your adapter. Ensure that the firewall allows inbound communication from the computer system hosting the adapter.

**Related reference**

Appendix D, “Keywords,” on page 63

---

## Event delivery when systems fail

To ensure that events are delivered after system failures, the Tivoli Event Integration Facility provides a cache on the adapter or the application receiving events.

This repository stores events and removes those events from the cache when they are delivered. Also, the Event Integration Facility ensures that the same event is not delivered more than once.

To configure your environment for the reliable delivery of events, you activate the cache and specify backup servers to deliver events. You can further avoid delivery failure by specifying a list of servers.

**Related tasks**

“Activating the cache” on page 22

“Configuring backup servers to deliver events” on page 23

**Related reference**

“Installing, configuring, and testing the adapter” on page 36

## Activating the cache

By default, the cache that ensures recovery after system failures stores events. You control the configuration of the cache with keywords in the configuration file.

You can use the following keywords to configure the cache.

*Table 2. Keywords for configuring the cache*

Configuration of the Cache	Keywords
Activation of the cache	BufferEvents
Rate to send events	BufferFlushRate MaxPacketSize
Size of the cache	BufEvtMaxSize

You can configure the cache in the following way:

- Store events in memory only with the BufferEvents keyword.
- Save these buffered events to a permanent file with the BufferEvents keyword set to YES. Also, the BufEvtPath keyword specifies the location of the permanent file.

The second configuration ensures that no events are lost during a system failure.

The following example from a configuration file demonstrates the use of cache keywords:

```
BufferEvents=YES
BufEvtPath=./buffer
MaxPacketSize=130
BufferFlushRate=96
ConnectionMode=C0
```

When connections to the event server fail, events wait in the cache. The Event Integration Facility attempts to reconnect to each of the backup servers in turn. If no servers are available after all retries, the API caches the events until a connection is made.

When enabling recovery features, it is important to determine the importance of performance and reliability. In some environments, the use of the cache can degrade performance depending on its configuration. Therefore you can bypass event caching by setting the BufferEvents keyword to NO.

### Related concepts

“Event delivery when systems fail” on page 21

“Performance and availability” on page 44

### Related tasks

“Configuring backup servers to deliver events” on page 23

“Filtering events when systems fail” on page 40

### Related reference

“Event reception for applications” on page 20

## Configuring backup servers to deliver events

The configuration uses TCP sockets to deliver events. You specify how to contact backup servers by defining the transport channel to be the server name and the port number.

You specify how to contact backup servers by completing the following steps:

1. Create names for connections you want to make to the backup servers. Use the `TransportList` keyword to create this list.
2. Specify either an SSL or a `SOCKET` connection for each of the items in this list by using the `Type` keyword.
3. Specify multiple channels as alternate paths for each type of transport, using the `ServerLocation` keyword.

### Example

The following is a backup example for the C API:

```
TransportList=t1,t2
t1Type=SSL
t1Channels=c1,c2
c1ServerLocation=sslhost1
c1Port=1111

c1_SSLKeystore=/my/location/gbkeys.kdb
c1_SSLKeystorePW=password
c1_SSLKeystoreStashFile=/my/location/gbkeys.sth
c1_SSLEncipherList=SSL_RSA_WITH_3DES_EDE_CBC_SHA

c1_SSLEnableFIPSMode=OFF

c2ServerLocation=sslhost2
c2Port=2222

c2_SSLKeystore=/my/other/location/gbkeys.kdb
c2_SSLKeystorePW=password
c2_SSLEncipherList=SSL_RSA_WITH_3DES_EDE_CBC_SHA
c2_SSLEnableFIPSMode=OFF

t2Type=SOCKET
t2Channels=c3,c4
c3ServerLocation=host1
c3Port=1234
c4ServerLocation=host2
c4Port=5678
```

**Note:** The backup type for both `SOCKET` and `SSL` transport type can be either `SSL` or `SOCKET`. The C API ignores the Java API keywords (`TMEHost`, `TMEUserID`, `TMEPassword`, and `TMEPort`).

### Related concepts

“Event delivery when systems fail” on page 21

“Performance and availability” on page 44

### Related tasks

“Activating the cache” on page 22

## Using the portmapper keywords

The keywords for the portmapper channel enable the receiver applications to register multiple ports under various portmapper program names, and the sending applications to access those registered names.

**Note:** The Event Integration Facility does not support the portmapper keywords for the 64-bit OS/390® library.

When the `channel_namePortMapper` keyword is set to YES, it forces the specified port to be registered with the portmapper for a receiver application. Thus, it ignores the specified port in favor of the portmapper to obtain the correct port for sender code. The API ignores any other value for `channel_namePortMapper`.

The default values for the `channel_namePortMapper`, `channel_namePortMapperNumber`, and `channel_namePortMapperVersion` keywords are the ones used by the event server. If these keywords are not present in the configuration file for the sender application and the portmapper is requested, a connection to the event server is attempted. In a receiver application, the port is only registered if the port is set to zero (0). In this case, the default values are used.

If a port is set to zero (0), EIF uses the `channel_namePortMapper`, `channel_namePortMapperNumber`, and `channel_namePortMapperVersion` keywords. If any of their values are not specified, their default values are used.

If a port is set to a value greater than zero, the portmapper is only used if the `channel_namePortMapper` keyword is set to YES. In this case, the specified port is ignored for the sender side but used for the receiver side. The `channel_namePortMapper` keyword allows the port used for portmapper to be specified. Thus, a port set to zero (0) will pick any available port. On the receiver side, it is advantageous to use the portmapper so that sending applications can connect to the receiver by means of the portmapper.

## Configuring a reception application built with the C API

A reception application built with the C API can be configured with keywords in the configuration file.

Use the following keywords in the configuration file to configure a reception application built with the C API.

### **ActiveConnections=nn**

The number of active connections that the reception process should handle.

The number of possible connections range from 2 to 10000.

A number less than the minimum value sets the value to the minimum value unless that number is zero, which means unlimited connections. A number greater than the maximum value sets the value to the maximum value. Not specifying a value, yields the default value of 128.

Applies to the C API only.

### **ActiveConnectionsSafety=nn**

The percentage of `ActiveConnections` that the number of actual connections permits must be reduced to a specific number before connections can be processed again. This is a threshold value.

Setting `ActiveConnections` limits the numbers of active connections handled by the C Event Integration Facility reception process.

For example, if `ActiveConnections` equals 20 and the `ActiveConnectionsSafety` equals 80, the reception process stops accepting connections when there are 20 connections. The percent can range from 10 to 90. New connections resume when the number of active connections is reduced to 16 (80% of 20) or less.

The default value is 80 and is only used when `ActiveConnections` has been specified.

A number less than the minimum value sets the value to the minimum value. A number greater than the maximum value sets the value to the maximum value.

Applies to the C API only.

### **ConnectionsQueued=nn**

The approximate number of queued connections that the reception process will handle.

The server socket queues up connections that are waiting to be accepted by the reception process. Connection attempts fail after this limit has been exceeded.

Use this option to limit the number of connections that can be queued. The actual number of connections queued can be slightly more than or less than the value of `ConnectionsQueued`. The default is 1000. The range of values can be between 1 and 1000. A number less than the minimum value sets the value to the minimum value. A number greater than the maximum value sets the value to the maximum value.

Applies to the C API only.

Example of a reception application build with the C API receiving events over a SOCKET connection:

```
BufferEvents=YES
BufEvtPath=/tmp/eif_socket_recv.cache
```

```
TransportList=t1
t1Type=SOCKET
t1Channels=t_
t_ServerLocation=myserver.com
t_Port=5151
```

Example of a reception application build with the C API receiving events over an SSL connection:

```
BufferEvents=YES
BufEvtPath=/tmp/eif_socket_recv.cache
```

```
TransportList=t1
t1Type=SSL
t1Channels=t_
t_ServerLocation=myserver.com
t_Port=5151
```

```
t_SSLKeystore=/my/other/location/gbkeys.kdb
t_SSLKeystorePW=password
t_SSLCipherList=SSL_RSA_WITH_3DES_EDE_CBC_SHA
t_SSLFIPMode=OFF
```

**Note:** The procedure to configure a keystore for a reception application is identical to that of a sender application.

---

## Chapter 4. Building an adapter

Before building an adapter, you must identify events to monitor. You then define the event source and event classes and select the method for event delivery. You program the event adapter, and install, configure and test it. You are then ready to run the adapter.

### Related concepts

“Migrating adapters” on page 12

---

## Adapter files

In addition to the header file or Java package, a number of files are related to the adapter. This section lists these files and shows the relationship between them and event processing.

There are several adapter files. In addition to the header file or Java package, the following are files related to the adapter:

### **.conf file**

The configuration file controls filtering and buffering of events. It also controls communications.

### **.rls file**

The rule file applies custom rules to events for filtering, tasks, and other actions. Some rule files are installed on the event server by default. You can optionally specify other rules.

For a detailed description of adapter files, see the *IBM Tivoli Enterprise Console Adapters Guide* in the IBM Tivoli Information Center at <http://publib.boulder.ibm.com/infocenter/tivihelp/v3r1/index.jsp>

### Related reference

Appendix D, “Keywords,” on page 63

---

## Identifying events to monitor

Before you create an adapter, you must decide what types of events you need to monitor.

Review the following factors to help you identify significant events:

- Users of IT resources
- Service level agreements
- Network requirements for down-time and up-time
- Application and network dependencies (for example, databases, e-commerce Web sites, wide area networks, and so forth)
- Performance requirements
- Important servers and network resources

After creating a list of significant events, use this list to define the source.

---

## Defining the source

The method used to retrieve event information depends on the resources.

When you develop a new adapter, you must determine how to gather information about the monitored resource. You must also determine a method for identifying the information that you want to send to the event server. For example, the necessary information about a resource can be gathered from a system log file. Then this information must be formatted, and optionally filtered, before being sent to the event server.

---

## Defining event classes

An important task when you create an adapter is to determine the event classes for the information that you want to monitor. To help you when you write rules to handle the events, you must make event definitions as specific as possible.

The Probe for Tivoli EIF has a set of default rules that map attributes of Event Integration Facility events to columns in alerts.status. You must define these common attributes so that the probe can process them.

**Note:** Event class names must be unique.

The event string is adapter-dependent. You can have as many as required pairs of attributes and values. The following code fragment illustrates the assembly of an event string:

```
"MY_EVENT_CLASS;  
source=_ANY_DEFINED_SOURCE;  
application=myApp11;  
origin=9.179.1.234;  
msg=Hello World;  
END\n\001"
```

In the code fragment, `source` and `application` are application-specific.

For more information about probe rules and attributes, see the IBM Tivoli Netcool/OMNIBus Probe for Tivoli EIF guide in the IBM Tivoli Network Management Information Center at [http://publib.boulder.ibm.com/infocenter/tivihelp/v8r1/topic/com.ibm.netcool\\_OMNIBus.doc/probes/tivoli\\_eif/tivoli\\_eif/wip/concept/tveif\\_intro.html](http://publib.boulder.ibm.com/infocenter/tivihelp/v8r1/topic/com.ibm.netcool_OMNIBus.doc/probes/tivoli_eif/tivoli_eif/wip/concept/tveif_intro.html).

For more information about the default classes hierarchy, see the *IBM Tivoli Enterprise Console Adapters Guide* in the IBM Tivoli Information Center at <http://publib.boulder.ibm.com/infocenter/tivihelp/v3r1/index.jsp>

### Related concepts

“Event classes” on page 3



---

## Selecting event delivery methods

While building an adapter, you also need to decide which event delivery method the adapter uses to communicate with the event server.

You must define the connection and transport types. You do this by editing the appropriate keywords in the EIF configuration file. The following connection types and transport types are available:

### Connection options

- Connection-oriented
- Connectionless

### Transport options

- SOCKET
- SSL
  - Normal SSL mode (non FIPS)
  - FIPS 140-2 mode

If you choose to use the SSL transport type, you must generate security certificates and keys. You must also decide whether to operate the SSL connection in FIPS 140-2 mode. If you do, you enable FIPS 140-2 mode by editing the configuration file.

### Related concepts

“Event delivery methods” on page 13

“Connection options” on page 13

“Federal Information Processing Standard 140–2 (FIPS 140–2) support” on page 17

### Related reference

“Transport options” on page 13

“SSL and FIPS 140-2” on page 14

Appendix D, “Keywords,” on page 63

## Configuring an EIF receiver application for SSL

In order to use Secure Sockets Layer (SSL) communication between EIF receiver applications and client applications, you must configure the receiver application. Receivers require a personal certificate that is either self-signed or signed by a certificate authority (CA).

### Before you begin

To configure an EIF receiver application for SSL, complete the following procedure:

1. Create a key database for the EIF receiver.
  - a. Start the **nc\_ikeyman** utility and create a new key database file.  
For C-based EIF applications, use the CMS key database type.  
For Java-based EIF applications, use the JKS key database type.
  - b. Choose a password for the key database.  
For CMS key databases, you can encrypt the password in a stash file.
2. Obtain a personal certificate in one of the following ways:
  - Use the **nc\_ikeyman** utility to create a new self-signed certificate.
  - Use the **nc\_ikeyman** utility to create a certificate request and submit it to a CA for signing.

- Obtain a signed certificate from a CA using another method.
3. Add the new personal certificate to the receiver key database and configure it to be the default personal certificate.
  4. Create a configuration file for the EIF receiver. Start with the following code:
 

```
TransportList=t1_
t1_Type=SSL
t1_Channels=c1_
c1_Port=[Listening port of the EIF receiver]
c1_ServerLocation=[Hostname or IP address of the EIF receiver]
c1_SSLKeystore=[File location of the receiver key database]
```
  5. Add the password of the receiver key database to the EIF configuration file.
    - To specify the password as plain text, add the following line to the EIF configuration file:
 

```
c1_SSLKeystorePW=password of the receiver key database
```
    - To encrypt the password and are using a CMS key database, you can use the stash file location associated with the key database. Add the following line to the EIF configuration file:
 

```
c1_SSLKeystoreStashFile=File location of the stash file
```
    - To encrypt the password and are using a JKS key database, you can use the Java EIF utility `com.tivoli.tec.event_delivery.common.Encryption` to create an encrypted password and an encryption key that will decode the password. Add the following lines to the EIF configuration file:
 

```
c1_SSLKeystoreStashFile=File containing the encrypted password
c1_SSLKeystoreEncryptionKeyFile=File containing the encryption key
```
  6. Choose the ciphers that the EIF receiver will support.
 

By default, a C-based EIF application will support all ciphers available in the IBM GSKit, and a Java-based EIF application will support all ciphers available in the IBM JRE.

Add the list of cipher names, separated by commas, to the EIF configuration file, using the following format:

```
c1_SSLCipherList=Cipher name 1,Cipher name 2, [...]
```
  7. Enable FIPS 140-2 mode.
 

By default, FIPS 140-2 mode is disabled. To enable FIPS 140-2 mode, add the following line to the EIF configuration file: `c1_SSIFIPSMODE=ON`

For Java-based EIF applications, the Java Runtime Environment must also be configured to use FIPS providers. Add the following entries to the list of security providers in the `lib/java/java.security` file of your JRE installation:

```
security.provider.number=com.ibm.fips.jsse.IBMJSSEFIPSProvider
security.provider.number=com.ibm.crypto.fips.provider.IBMJCEFIPS
```
- Note:** Use *number* to indicate the sequence you want to allocate to each provider in the list.
8. Specify if clients must present a trusted certificate.
 

By default, EIF does not require clients to present a trusted certificate. To add a trusted certificate, add the following line to the EIF configuration file:

```
c1_SSLRequireClientAuthentication=YES
```
  9. For C-based EIF applications, update the required environment variable with the path to the GSKit library files.

#### On Windows® operating systems

PATH

**Note:** The GSKit libraries are located in `%NCHOME%\platform\win32\lib`

## On UNIX and Linux operating systems

LIBPATH, SHLIB\_PATH, or LD\_LIBRARY\_PATH

**Note:** The GSKit libraries are located in \$NCHOME/platform/*arch*/lib with *arch* being your operating system directory.

## What to do next

You must configure the EIF client application for SSL next.

### Related reference

“SSL and FIPS 140-2” on page 14

Appendix D, “Keywords,” on page 63

## Configuring an EIF client application for SSL

In order to use SSL communication between EIF receiver and client applications, you must configure the EIF client application.

## Before you begin

You need to create the EIF receiver's personal certificate before you can configure the client application.

To configure an EIF client application for SSL, complete the following procedure:

1. Create a key database for the EIF client.
  - a. Start the **nc\_ikeyman** utility and create a new key database file.  
For C-based EIF applications, use the CMS key database type.  
For Java-based EIF applications, use the JKS key database type.
  - b. Choose a password for the key database.  
For CMS key databases, you can encrypt the password in a stash file.
2. Add the personal certificate of the EIF receiver to the client key database.
  - a. Use the **nc\_ikeyman** utility to export the default personal certificate from the receiver key database.
  - b. Use the **nc\_ikeyman** utility to import the certificate into the client key database.
3. Create a configuration file for the EIF client. Start with the following:

```
TransportList=t1_
t1_Type=SSL
t1_Channels=c1_
c1_Port=[Listening port of the EIF receiver]
c1_ServerLocation=[Hostname or IP address of the EIF receiver]
c1_SSLKeystore=[File location of the receiver key database]
```
4. Add the password of the client key database to the EIF configuration file.
  - To specify the password as plain text, add the following line to the EIF configuration file:  
`c1_SSLKeystorePW=password of the key database`
  - To encrypt the password and are using a CMS key database, you can use the stash file location associated with the key database. Add the following line to the EIF configuration file:  
`c1_SSLKeystoreStashFile=File location of the stash file`
  - To encrypt the password and are using a JKS key database, you can use the Java EIF utility `com.tivoli.tec.event_delivery.common.Encryption` to create an encrypted password and an encryption key that will decode the password. Add the following lines to the EIF configuration file:

- `c1_SSLKeystoreStashFile=File containing the encrypted password`  
`c1_SSLKeystoreEncryptionKeyFile=File containing the encryption key`
5. Specify the ciphers that are required by the EIF receiver. If the receiver supports all ciphers, omit this step.  
 Add the list of cipher names, separated by commas, to the EIF configuration file, using the following format:  
`c1_SSLCipherList=Cipher name 1,Cipher name 2, [...]`
  6. If the EIF receiver has been enabled for FIPS 140-2 mode, enable FIPS 140-2 mode for the client as well. Otherwise, do not enable 140-2 mode for the client. By default, FIPS 140-2 mode is disabled. To enable FIPS 140-2 mode, add the following line to the EIF configuration file: `c1_SSLEFIPSMODE=ON`  
 For Java-based EIF applications, the Java Runtime Environment must also be configured to use FIPS providers. Add the following entries to the list of security providers in the `lib/java/java.security` file of your JRE installation:  
`security.provider.number=com.ibm.fips.jsse.IBMJSSEFIPSProvider`  
`security.provider.number=com.ibm.crypto.fips.provider.IBMJCEFIPS`
- Note:** Use *number* to indicate the sequence you want to allocate to each provider in the list.
7. If the EIF receiver requires clients to present a trusted certificate, complete the following steps:
    - a. Obtain a trusted certificate. You can use the **nc\_ikeyman** utility to create a new self-signed certificate, or use the **nc\_ikeyman** utility to create a certificate request and submit it to a CA for signing. Alternatively, you can obtain a signed certificate from a CA using another method.
    - b. Add the new certificate to the client key database and configure it to be the default certificate.
    - c. Add the new certificate to the receiver key database, but do not configure it to be the default certificate.
  8. For C-based EIF applications, update the required environment variable with the path to the GSKit library files.

#### On Windows® platforms

PATH

**Note:** The GSKit libraries are located in `%NCHOME%\platform\win32\lib`

#### On UNIX and Linux platforms

LIBPATH, SHLIB\_PATH, or LD\_LIBRARY\_PATH

**Note:** The GSKit libraries are located in `$NCHOME/platform/arch/lib` with *arch* being your operating system directory.

#### Related reference

“SSL and FIPS 140-2” on page 14

Appendix D, “Keywords,” on page 63

---

## Programming the adapter

To program an adapter, you implement the interfaces and the preferred settings for the configuration file. You decide whether to define attribute values in the configuration file or the adapter code. You then compile and, if required, link the adapter.

### Before you begin

#### Sample adapter code

Sample adapter code for the C and Java languages can be found in the EIFSDK directory.

#### Data transfer APIs

Data is transferred to the event server by assembling an event string. The `tec_put_event` function or `sendEvent` method sends the string to the event server.

#### Compiling the adapter

Adapters can be built using either the C or Java API.

Adapters built using the Event Integration Facility are not thread-safe and cannot be multithreaded.

#### Compiling the adapter built with the Java API

To compile a Java source that uses the Event Integration Facility Java API, import `evd.jar` into your source file and ensure that `evd.jar` and `log.jar` are available on the compilation class path.

The Event Integration Facility provides a C API and a Java API to communicate with the event server.

*Table 3. APIs and their behaviors*

Tasks	Java API	C API
Access configuration files and read keyword data	<b>TECAgent</b> <b>getConfigVal</b>	<b>tec_agent_init</b> <b>tec_agent_getenv</b>
Establish and close communications with the event server	<b>TECAgent</b> <b>disconnect</b>	<b>tec_create_handle</b> <b>tec_destroy_handle</b>
Send and receive events	<b>sendEvent</b> <b>receiveEvent</b> <b>registerListener</b> <b>onMessage</b>	<b>tec_put_event</b> <b>tec_get_event</b> <b>tec_register_callback</b> <b>tec_event_callback</b>
<b>Note:</b> The locale is set independently of the Event Integration Facility, and calling the <b>tec_create_handle</b> API does not change the locale.		

You use the functions and methods to handle the configuration files, communications, and data transfer.

**Related concepts**

Appendix A, “Application programming interfaces,” on page 47

**Related reference**

Appendix D, “Keywords,” on page 63

“Shell script and test options” on page 17

## Upgrading existing adapters

To upgrade existing adapters, link your adapters to the `libeif.a` library.

When upgrading existing adapters, relink your Event Integration Facility adapter code with the `libeif.a` stub library. If you receive an undefined symbol error message, it means that the EIF function that you have been using is no longer supported. Update your code and compile and link again.

**Related concepts**

Appendix A, “Application programming interfaces,” on page 47

**Related reference**

“Linking the adapter built with the C API” on page 36

## Configuration file APIs

The first task performed by the APIs is accessing information from the configuration files.

To enable the `tec_agent_getenv` function or `getConfigVal` method, you first call the initialization API (`tec_agent_init` or `TECAgent`). The initialization API reads in configuration information that is used by all subsequent functions or methods.

**Related reference**

Appendix D, “Keywords,” on page 63

## Communications APIs

The communications APIs provide a mechanism to communicate with the event server. The handle is specified when sending an event.

Regardless of the type of transport mechanism, use a single call to `tec_create_handle` (or `TECAgent`) to establish communications with the server. The following code example instantiates `TECAgent`, passing as parameters the configuration file, the delivery mode, and the error-reporting mechanism:

```
public TECAgent(reader configStream, int deliveryMode, int oneway)
```

Call the `tec_destroy_handle` function or `disconnect` method when you no longer want to communicate with the event server. This call is optional, because the channel automatically disconnects when the adapter exits.

## Special considerations for Microsoft Windows

An adapter built for Microsoft Windows operating systems must initialize the Winsock before calling `tec_create_handle`.

The following example of a Microsoft Windows adapter initializes Winsock before calling `tec_create_handle`:

```
#ifdef WIN32
#include <winsock.h>
    WSADATA wsaData;

    if ((rc = WSStartup(MAKEWORD(1, 1), &wsaData)) != 0) {
        printf("error %d starting winsock.dll\n", rc);
        exit(1);
    }

    else
        printf("Winsock initialized successfully...\n");
#endif
```

An application must call the `WSStartup` function to initialize Winsock, regardless of which version of Winsock is being used.

`WSStartup` initializes `Winsock2.dll` and a `WSADATA` structure that contains the details of the Winsock implementation. When an application or DLL has finished using `Winsock2.dll`, it must call `WSACleanup` to enable `Ws2.dll` to free any resources for the application. For every call to `WSStartup`, there must be a call to `WSACleanup`.

If successful, `WSStartup` returns 0. After `WSStartup` returns, an application cannot call `WSAGETLastError` to determine the error value.

## Compiling the adapter built with the C API

To compile a source file that uses the Event Integration Facility C API, you need to include the `tec_eef.h` header file.

When compiling the sample adapter on Windows, you must specify the `PC` flag as a compiler argument to avoid compile-time errors.

To link the sample adapter, you must use the `NODEFAULTLIB:LIBC.LIB` option when linking the adapter on Windows. This allows the linker to avoid conflicts with the default libraries.

If there are multiple `wsock32.lib` files, you must use the `FORCE:MULTIPLE` option when compiling the Windows adapter. This forces the compiler to pick one file to eliminate compile-time errors. In conjunction with the `FORCE:MULTIPLE` option, the `INCREMENTAL:NO` option must also be used.

### Example

The following example shows how to compile and link a sample adapter using the options mentioned above:

```
unsecure: adapter.c \
$ (CC) -nologo -Ze -W3 -MD -DUNSECURE -D_WIN32 -DWIN32 -DPC \
-Ic:/Tivoli/include/w32-ix86/TME/TEC -Id:/msdev/include -FosampleAdapter.obj
-c sampleAdapter.c \
```

```
slashes link -subsystem:console -L. -Ld:/msdev/lib -Lc:/Tivoli/lib/w32-ix86 \
-out:sampleAdapter.exe sampleAdapter.obj msvcrt.lib libeif.a \
libsunrpc.a -NODEFAULTLIB:LIBC.LIB -INCREMENTAL:NO -FORCE:MULTIPLE wsock32.lib
```

## Linking the adapter built with the C API

These tables list the libraries required to link adapters developed with the C API.

*Table 4. Libraries for adapters developed with the Event Integration Facility C API*

Libraries	Provided by	More details
libeif.a	Tivoli Event Integration Facility	None
libdl.a	Operating system	Not on Windows and HP-UX
libpthreads.a	Operating system	For adapters on AIX
libpthread.a	Operating system	For adapters on Linux
libnsl.a	Operating system	For adapters on the Solaris Operating Environment
libsocket.a		
libthread.a		
libsunrpc.a	Tivoli Event Integration Facility	For adapters on Windows
Standard C libraries	Operating system	None

### Related reference

“Transport options” on page 13

“Upgrading existing adapters” on page 34

---

## Installing, configuring, and testing the adapter

After assembling all the files for the adapter, you need to install, configure, and test the adapter before running it.

After installing the adapter, you can also add other features to the adapter. For example, you can modify the adapter to enable the following features:

- Event filtering
- Event caching for failure and recovery
- Rules (see the *IBM Tivoli Enterprise Console Rule Developer's Guide*)

Before installing your new adapter in a production environment, test at a minimum the following functionality:

- Does the adapter successfully install?
- Are events mapped to the appropriate event classes?
- Are events arriving and displaying on the console, if applicable?
- Are events correctly filtered?

### Related concepts

“Event delivery when systems fail” on page 21

### Related reference

“Shell script and test options” on page 17



---

## Running adapters built with the Event Integration Facility Java API

The Event Integration Facility Java API depends on other classes to accomplish its tasks. In addition to setting up the appropriate environment using the **setup\_env** commands, you must add the path to the Java executable file to your library path environment variable.

The following tables list the Java jar files and libraries required to run an adapter that is built using the Event Integration Facility Java API.

To run an adapter that uses the Java Event Integration Facility API, you must add the required libraries to your CLASSPATH environment:

*Table 5. Libraries required for adapters developed with the Event Integration Facility Java API*

Required Libraries	Provided by	More details
evd.jar log.jar	Tivoli Event Integration Facility	None.
ibmjssefips.jar ibmjsseprovider2.jar	Tivoli Event Integration Facility	Optional, for SSL support only.

### Related reference

Appendix D, “Keywords,” on page 63

---

## Configuring adapters for international environments

The event server can receive events in both UTF-8 encoding or the encoding of the event server host. The event server automatically determines the type of encoding (UTF-8 or non-UTF-8) of an event by evaluating a particular flag in the event data.

For all adapters using SOCKET and SSL transport types, use UTF-8 encoding to send events to the event server.

**Note:** If the adapter is sending events to an event server host running a Tivoli Enterprise Console version earlier than version 3.7, the format files in localization directories must remain in English.

### Related reference

Appendix D, “Keywords,” on page 63



---

## Chapter 5. Filtering events at the source

One of the problems associated with event management is working with the high volume of events that devices can generate. You can address this problem by filtering events at the source.

A high volume of events can be generated, for example, if a router has dropped below a key performance threshold, such as the amount of time to return a ping. Typically, the router would be set to generate an event every 30 seconds until an operator has found and addressed the underlying problem. This would cause redundant events to flood the event console, impacting the problem-solving process.

Tivoli Event Integration Facility addresses this problem with two powerful techniques for analyzing, summarizing, and distributing the incoming event information:

### **Filtering with configuration files**

By using configuration files on adapters and gateways, you can filter events based on matches to event classes.

This is the simpler and limited option in terms of capabilities.

When defined correctly, configuration files optimize event management by minimizing the number of events that each operator must monitor.

### **Related concepts**

“Event filtering” on page 6

---

## Filtering with configuration files

Normally, an adapter sends all events to the event server. You can optionally list the events that the adapter can or cannot send to the event server by using the Filter and FilterCache keywords. Similarly, you can modify the configuration file to filter events.

A configuration file can contain as many filter entries as needed. You specify the event class and information such as the origin, severity, or any other attribute and value pair that is defined for the event class. Depending on how you specify the Filter and FilterMode keywords, filtered events are either sent to the event server or discarded.

Define event filters as follows:

- To send all events to the event server (the default behavior):
  1. Set FilterMode to OUT.
  2. Do not specify any Filter statements.
- To send specific events to the event server:
  1. Set FilterMode to IN.
  2. Create Filter statements to match the specific events that you want sent.
- To discard all events:
  1. Set FilterMode to IN.
  2. Do not specify any Filter statements.

- To discard specific events:
  1. Set FilterMode to OUT (the default value).
  2. Create Filter statements to match the specific events that you want discarded.

## Example

The following example shows two filters. The first filter suppresses all events with the class disk\_event. The second filter suppresses all events with the class Su\_Success from the IP address 126.32.2.14.

```
#
# Event Filters
#
Filter:Class=disk_event
Filter:Class=Su_Success;origin=126.32.2.14
```

### Related reference

Appendix D, “Keywords,” on page 63

## Filtering events when systems fail

When an adapter is unable to connect to the event server, it sends the events to a file if the BufferEvents keyword is set to YES.

You can filter events sent to a cache file, similar to filtering events for the event server, by using the FilterCache keyword.

The following procedures describe how to filter events with the FilterCache and FilterMode keywords, when the event server is unavailable:

- To cache specific events:
  1. Set FilterMode to IN.
  2. Set BufferEvents to YES (the default value).
  3. Create Filter and FilterCache statements to match the specific events that you want cached.
- To discard specific events:
  1. Set FilterMode to OUT.
  2. Create Filter and FilterCache statements to match the specific events that you want discarded.
- To cache all events (the default behavior):
  1. Set FilterMode to OUT.
  2. Set BufferEvents to YES.
  3. Do not specify any FilterCache statements.

**Note:** All events are discarded when the configuration is as follows:

1. FilterMode is set to IN.
2. No FilterCache statements are specified.

### Related tasks

“Activating the cache” on page 22

## Regular expressions in filters

You can use Tcl regular expressions in filtering statements.

The format of a regular expression is:

re: *'value\_fragment'*

A regular expression is zero or more branches, separated by a vertical bar (|). A regular expression matches anything that matches one of the branches.

A branch is zero or more pieces that are concatenated. It matches a match for the first, followed by a match for the second, and so forth.

A piece is an atom possibly followed by an asterisk (\*), a plus sign (+), or a question mark (?). An atom followed by an asterisk (\*) matches a sequence of 0 or more matches of the atom. An atom followed by a plus sign (+) matches a sequence of 1 or more matches of the atom. An atom followed by a question mark (?) matches a match of the atom, or the null string.

An atom is a regular expression in parentheses (matching a match for the regular expression), a range, a period (.) (matching any single character), a caret (^) (matching the null string at the beginning of the input string), a dollar sign (\$) (matching the null string at the end of the input string), a backslash (\) followed by a single character (matching that character), or a single character with no other significance (matching that character).

A range is a sequence of characters enclosed in brackets [ ]. A range matches any single character from the sequence. If the sequence begins with a caret (^), it matches any single character not from the rest of the sequence. If two characters in the sequence are separated by minus sign (-), this represents the full list of ASCII characters between them. For example, the range [0-9] matches any decimal digit. To include a literal right bracket (]) in the sequence, use the right bracket (]) as the first character, following a possible caret (^). To include a literal minus sign (-), use the minus sign (-) as the first or last character.

A sample program named `regtest` and a sample input file named `regtest.data` are provided in the EIFSDK directory for testing regular expressions. The sample is provided for each operating system in the `<eifsdk>/bin/$INTERP` directory.

### Restriction:

1. The regular expression code is a slightly altered version of code originally written by Henry Spencer, Copyright 1986 by the University of Toronto. It is not derived from licensed software. Permission is granted to anyone to use this software for any purpose on any computer system, and to redistribute it freely, subject to the following restrictions:
  - a. The author is not responsible for the consequences of use of this software, even if they arise from defects in it.
  - b. The origin of this software must not be misrepresented, either by explicit claim or by omission.
  - c. Altered versions must be plainly marked as such, and must not be misrepresented as being the original software.
2. The Event Integration Facility uses an exception to the Tcl regular expression syntax. The backslash character (\) in the Event Integration Facility indicates

that the following literal character is the character to filter for, not some special character such as a tab. For example, \t means the tab character in Tcl, but means t in Event Integration Facility.

The following example shows a `Filter` statement with a regular expression. This filter statement matches all events with a class name that starts with `TEC_`

```
Filter:Class=re:'TEC_.*'
```

The following example shows a `FilterCache` statement with a narrower range. This filter statement matches all events with a class name that starts with `TEC_` and has a severity of critical:

```
FilterCache:Class=re:'TEC_.*';severity=CRITICAL
```

For more information about Tcl regular expressions, see a Tcl user's guide.

---

## Chapter 6. Troubleshooting

You can troubleshoot problems that can arise installing and using the Tivoli Event Integration Facility. The logs and tracing utilities can help you determine the source of problems when they occur. Before systems fail, you can configure the API to insure the delivery of events. Additionally, you can improve the performance of event delivery with API configurations.

---

### Message logs

In problem-solving situations, you need to understand how to interpret messages and what actions you can take to resolve a problem. You use the message log files to troubleshoot problems in your environment.

To generate log messages for the Java API, you specify keywords in the configuration file.

Use the `LogLevel` and `LogFileName` keywords to specify the amount and destination of messages.

For the C API, you specify the `ed_diag_config_file` keyword.

#### **Related reference**

Appendix D, “Keywords,” on page 63

---

### Trace logs

Trace logs assist you in determining why a problem is occurring. Trace logging captures information about the operating environment when the Tivoli Event Integration Facility fails to operate as intended.

Customer Support personnel use the information captured by trace loggers to trace a problem to its source or to determine why an error occurred. These tools are not enabled by default. Because trace messages are intended for Customer Support, they are generally written to a file that can be viewed for later examination.

To generate trace messages for the Java API, you specify keywords in the configuration file.

Use the `TraceLevel` and `TraceFileName` keywords to specify the amount and destination of tracing.

For the C API, you specify the `ed_diag_config_file` keyword.

#### **Related reference**

Appendix D, “Keywords,” on page 63

---

## Performance and availability

The Tivoli Event Integration Facility can control performance and availability for event processing.

To prevent overloading with event delivery, you can add timers for event delivery, and specify the maximum number of events in the cache.

When using timers, the Event Integration Facility notifies the event receiver that there are cached events. This occurs when the timer expires or when the maximum number of events is exceeded. Therefore the receiver can process the whole cache contents at once.

Additionally, you can increase the availability of events by setting up backup servers.

### Related tasks

“Activating the cache” on page 22

“Configuring backup servers to deliver events” on page 23

## Event reception connection parameters

The reception process creates a server socket that listens on a specified port. The listening process marks a connection-mode socket as accepting connections and limits the number of outstanding connections in the queue of the listening socket to the value specified by the `ConnectionsQueued` value.

The implementation can include incomplete connections in the queue subject to the queue limit. Implementations can limit the length of the queued listening socket by specifying `ConnectionsQueued`.

TCP/IP allows data to be sent and the connection to be closed by the sender before the receiving application can receive the data. The reception process eventually accepts the connection and processes the data. `ConnectionsQueued` allows these types of connections to be limited. Connection attempts are refused when TCP/IP can no longer queue up connections.

The number of active connections handled by the reception process can be limited by the `ActiveConnections` keyword. This prevents the reception process from accumulating too many connections that consume system resources.

If all of the initial connections are `ConnectionOriented` and the `ActiveConnections` have been reached, then no more connections are accepted by the reception process. However, connections are still being considered by the TCP/IP up to the amount of `ConnectionsQueued` has been reached and if all `ActiveConnections` are connection oriented then the state remains unchanged until the number of active connections is reduced to the amount specified by the percentage of `ActiveConnections` through the use of `ActiveConnectionsSafety`.

**Note:** The connections that were handled by TCP/IP (`ConnectionsQueued`) are discarded when the program ends. The connections that are in the `ActiveConnections` list are all closed and any data, residing on the connections, is discarded.



---

## Common problems and scenarios

When using the Tivoli Event Integration Facility you can encounter a number of common problems and scenarios.

### Building and running adapters

A number of scenarios can occur when there are problems building, compiling, and running the adapter.

#### My adapter on a Windows operating system has compile-time errors

**Cause** There are multiple `wsock32.lib` files.

**Remedy**

Use the `FORCE:MULTIPLE` option when compiling the Windows NT adapter.

This forces the compiler to pick one file.

#### I changed the configuration file, but my changes do not take effect

**Cause** There could be improperly specified statements in the configuration file.

**Remedy**

Review the following for the changed keywords:

- Check for typos in the spelling of keywords.
- Ensure that any blank spaces are enclosed in single quotation marks.

#### I received a LOG0014E error

**Cause** The system cannot find the file specified by the `LogFileName` keyword.

**Remedy**

Correct the path name specified by the `LogFileName` keyword.

### Making connections to the event server

A number of scenarios can occur when there are problems connecting to the event server.

#### I received a connection error when I use `postEIFMSG`

**Cause** The error indicates that you might be using a user ID other than Administrator or root. Therefore your ID does not have the correct permissions to create and write the file specified by the `BufEvtPath` keyword.

**Remedy**

Ensure that you have the correct permissions for creating the file specified by the `BufEvtPath` keyword.

## Sending events

A number of scenarios can occur when there are problems sending events to the event server.

### **My adapter is not sending all the events to the event server.**

**Cause** There is either a communication problem between the adapter and the event server, or there is a problem internal to the adapter code.

#### **Remedy**

Send events to a file instead of directly to the event server. Then, verify the event delivery.

The following steps describe how to do this:

1. Set the TestMode keyword to YES in the adapter configuration file.
2. Specify the file to receive the events with the ServerLocation keyword.
3. Restart the adapter.
4. Review the file specified by the ServerLocation keyword and check to see if all the events appear there.

If all events appear in the file, then there is a communication problem with the event server.

If the events are missing from the file, then there is problem internal to the adapter. Check the adapter code.

**Note:** When you complete testing of the adapter, reset the TestMode keyword, so that events are sent to the event server and no longer directed to a file.

### **I am using the posteifmsg commands to send events, but they are not arriving at the event server**

**Cause** The events are being sent to the cache on the adapter or gateway, because of one of the following situations:

- A portmapper is in use on the sending side, but a portmapper is not in use on the event server. For Windows systems, there is no portmapper daemon.
- A non-valid port, host name, or event server was specified.

#### **Remedy**

Resolve the non-valid name or port issues.

---

## Appendix A. Application programming interfaces

The APIs for Tivoli Event Integration Facility are defined by C language functions and Java methods and help you to build a custom adapter or an application that receives events.

### Related tasks

“Programming the adapter” on page 33

### Related reference

“Upgrading existing adapters” on page 34

---

## C language API

In order to build a custom adapter in the C language, you must have a C compiler.

### tec\_agent\_getenv

The `tec_agent_getenv` function retrieves the value of the variable contained in the configuration file.

#### Synopsis

```
char *tec_agent_getenv(char *keyword)
```

#### Arguments

##### keyword

The keyword variable to retrieve.

#### Examples

```
#include "tec_eef.h"
char *serverLoc=tec_agent_getenv("ServerLocation")
```

#### Return codes

Returns a pointer to a string which is the value of the variable. Do not free this pointer. Returns a NULL if the keyword does not appear in the configuration file.

### tec\_agent\_init

The `tec_agent_init` function is an initialization function that reads the configuration file and caches the information.

**Note:** The `tec_agent_init` function is the first function called to initialize the Tivoli Event Integration Facility module. Only call the `tec_agent_init` function once for each adapter, and call it before any of the other functions.

#### Synopsis

```
int tec_agent_init(char *cfgfile)
```

#### Arguments

**cfgfile** The full path to the configuration file.

## Examples

```
#include "tec_eeif.h"
tec_agent_init("config");
```

## Return codes

Returns 0 if successful.

## tec\_create{EIF\_handle

The `tec_create{EIF_handle` function establishes a handle for sending events to the event server or receiving events from a source with the first argument specifying a configuration file. A handle is created using the configuration information specified in the configuration file. This function is similar to `tec_create_handle`.

## Synopsis

```
tec_handle_t tec_create{EIF_handle(char *cfgfile, int oneway,delivery_mode mode)
```

## Arguments

**cfgfile** The full path to the configuration file.

### **oneway**

Used for managed node adapters only, and is used to designate whether calls to `tec_put_event` returns exceptions to the caller in the event of failure. A value of one (1) means that exceptions, if any, are not returned to the caller of `tec_put_event` because the caller does not wait for a response from the `oserv` process. A value of zero (0) means that exceptions are returned to the caller because the caller waits for the `oserv` process to return the success or failure of the method.

**mode** The possible values are the following:

- submission
- reception

Use the submission mode when the handle is used for transmitting. Use the reception mode when the handle is used for receiving.

## Examples

```
#include "tec_eeif.h"
if((th =tec_create{EIF_handle("config",0,submission))==NULL){
    fprintf(stderr,"%s:tec_create_handle failed errno=%d \n ",
        progname,tec_errno);
    exit(1);
}
```

## Return codes

A handle to an internal data structure. The handle is used in calls to other API functions.

## tec\_create\_handle

The `tec_create_handle` function establishes a handle for sending events to the event server.

### Synopsis

```
tec_handle_t tec_create_handle(char *location, unsigned short port,  
int oneway, tec_delivery_type type)
```

### Arguments

#### location

The host name or host protocol address of the event server.

#### oneway

Used for managed node adapters only, and is used to designate whether calls to `tec_put_event` returns exceptions to the caller in the event of failure. A value of one (1) means that exceptions, if any, are not returned to the caller of `tec_put_event` because the caller does not wait for a response from the `oserv` process. A value of zero (0) means that exceptions are returned to the caller because the caller waits for the `oserv` process to return the success or failure of the method.

**port** The port the event server listens on.

**type** The possible values are the following:

- `connection_less`
- `connection_oriented`
- `use_default`

The `use_default` value reads the setting from the configuration file for the `ConnectionMode` keyword and sets up a connectionless handle if the `ConnectionMode` keyword is not specified.

### Examples

```
#include "tec_eelf.h"  
if((th = tec_create_handle(tec_server, port, oneway, type)) == NULL) {  
    fprintf(stderr, "%s: tec_create_handle failed errno=  
        progname, tec_errno);  
    exit(1);  
}
```

### Return codes

A handle to an internal data structure. The handle is used in calls to `tec_put_event`. If the location is `NULL`, the configuration file `ServerLocation` entry is used to derive the location. If the port is zero, the `ServerPort` entry is used, if any; or else the portmapper is queried for the port on which the event server listens.

## tec\_create\_handle\_c

The `tec_create_handle_c` function establishes a handle for sending events to the event server or receiving events from a source with the first argument specifying a configuration file. A handle is created using the configuration information specified in the configuration file. This function is similar to `tec_create_handle`.

### Synopsis

```
tec_handle_t tec_create_handle_c (char *cfgfile, char *location, unsigned short  
port, int oneway, tec_delivery_type type, delivery_mode mode)
```

### Arguments

**cfgfile** The full path to the configuration file.

#### location

The host name or host protocol address of the event server.

**mode** The possible values are the following:

- submission
- reception

Use the submission mode when the handle is used for transmitting. Use the reception mode when the handle is used for receiving.

#### oneway

Used for managed node adapters only, and is used to designate whether calls to `tec_put_event` returns exceptions to the caller in the event of failure. A value of one (1) means that exceptions, if any, are not returned to the caller of `tec_put_event` because the caller does not wait for a response from the `oserv` process. A value of zero (0) means that exceptions are returned to the caller because the caller waits for the `oserv` process to return the success or failure of the method.

**port** The port the event server listens on for non-TME versions.

**type** The possible values are the following:

- connection\_less
- connection\_oriented
- use\_default

The `use_default` value reads the setting from the configuration file for the `ConnectionMode` keyword and sets up a connectionless handle if the `ConnectionMode` keyword is not specified.

### Examples

```
#include "tec_eef.h"  
if((th  
=tec_create_handle_c("config","tecserver.com",5529,0,connection_less,submission))  
==NULL){  
    fprintf(stderr,"%s:tec_create_handle failed errno=%d \n ",  
            progname,tec_errno);  
    exit(1);  
}
```

### Return codes

A handle to an internal data structure. The handle is used in calls to `tec_put_event`. If the location is `NULL`, the configuration file `ServerLocation` entry is used to derive the location. If the port is zero, the `ServerPort` entry is used, if any; or else

the portmapper is queried for the port on which the event server listens.

## tec\_create\_handle\_r

The `tec_create_handle_r` function establishes a handle for sending events to the event server with the first argument specifying a configuration file. A handle is created using the configuration information specified in the configuration file. Similar to `tec_create_handle`.

### Synopsis

```
tec_handle_t tec_create_handle_r (char *cfgfile, char *location, unsigned short  
port, int oneway, tec_delivery_type type)
```

### Arguments

**cfgfile** The full path to the configuration file.

**location**

The host name or host protocol address of the event server.

**oneway**

Used for managed node adapters only, and is used to designate whether calls to `tec_put_event` returns exceptions to the caller in the event of failure. A value of one (1) means that exceptions, if any, are not returned to the caller of `tec_put_event` because the caller does not wait for a response from the `oserv` process. A value of zero (0) means that exceptions are returned to the caller because the caller waits for the `oserv` process to return the success or failure of the method.

**port** The port the event server listens on for non-TME versions.

**type** The possible values are the following:

- `connection_less`
- `connection_oriented`
- `use_default`

The `use_default` value reads the setting from the configuration file for the `ConnectionMode` keyword and sets up a connectionless handle if the `ConnectionMode` keyword is not specified.

### Examples

```
#include "tec_eeif.h"  
th = tec_create_handle_r("config", "localhost", 1234, 0, use_default)) == NULL) */
```

### Return codes

A handle to an internal data structure. The handle is used in calls to `tec_put_event`. If the location is `NULL`, the configuration file `ServerLocation` entry is used to derive the location. If the port is zero, the `ServerPort` entry is used, if any; or else the portmapper is queried for the port on which the event server listens.

## tec\_destroy\_handle

The `tec_destroy_handle` function destroys the handle to the event server created by `tec_create_handle`, `tec_create_handle_c`, `tec_create{EIF_handle`, and `tec_create_handle_r` and closes any established connections.

### Synopsis

```
void tec_destroy_handle (tec_handle_t th)
```

### Arguments

**th**        The tec handle returned from a call to a `create_handle` function.

### Examples

```
#include "tec_eEIF.h"
tec_destroy_handle(th);
```

## tec\_errno

When a function returns an error, the `tec_errno` function is set to the appropriate error code.

### Synopsis

```
extern int tec_errno
```

## tec\_get\_event

The `tec_get_event` function allows an application to receive events. It receives events from the configured transport, on demand. The data returned can contain more than one event. Use the `ed_scan_n` utility to determine the number of events. The memory allocated for the event must be freed.

### Synopsis

```
long tec_get_event (tec_handle_t th, unsigned char ** event_message);
```

### Arguments

**th**        The event server handle returned from a call to a `create_handle` function.

**event\_message**

Contains the event data of the message received from a transport.

### Examples

```
#include "tec_eEIF.h"
char *event;
long event_len;
int rc;

event=NULL;
event_len = tec_get_event(th, &event);
if (event && event_len)
{
    n =ed_scan_n (event,event_len);
}
if (event)
    free(event)
```

### Return codes

Returns the length of the event message. Returns 0 (zero) when no events are available.



## tec\_put\_event

The `tec_put_event` function sends an event to the event server.

**Note:** If buffering is enabled in the configuration file, events are placed into a buffer and then sent on a separate thread. If a call to `tec_put_event` is followed by a call to `tec_destroy_handle`, the handle might be destroyed before the event can be sent. To prevent this from happening, use the `ed_sleep` utility function after `tec_put_event` to allow time for the event to be sent:

```
ed_sleep(0,100);
```

### Synopsis

```
long tec_put_event (tec_handle_t th, char *event)
```

### Arguments

**event** The character string representing the event.

**th** The event server handle returned from a call to a `create_handle` function.

### Examples

```
#include "tec_eelf.h"
if ( tec_put_event(th, event_string) == -1 {
    fprintf(stderr, "%s: tec_put_event failed, errno=,
        progname, tec_errno);
    exit(1);
}
```

### Return codes

Returns the number of bytes sent to the event server, other applications listening for events, or the cache file. A zero return means the event is filtered out. A negative return indicates an error.

## tec\_register\_callback

The `tec_register_callback` function allows an application to receive events through an upcall. The application registers a callback, passing as a parameter the method that handles the received events.

### tec\_event\_callback syntax

The syntax for `tec_event_callback` is as follows:

```
int (*tec_event_callback)(tec_handle_t h, unsigned char *msg, long _msg_len);
```

The data returned can contain more than one event. Use the `ed_scan_n` utility to determine the number of events returned. The memory allocated for the event must be freed.

The `tec_event_callback` function returns `-1` or zero (`0`). A zero indicates that there were no errors and that the event has been processed. A `-1` indicates that there was a problem processing the event and to not remove it from the cache if one is configured.

### Synopsis

```
void tec_register_callback(tec_handle_t th, tec_event_callback *fn)
```

## Arguments

- th**      The event server handle returned from a call to a `create_handle` function.
- fn**      The function to be called when the event arrives.

## Examples

```
#include "tec_eeif.h"
int on_message (tec_handle_t th, unsigned char *event, long event_len)
{
    long n;
    if (event && event_len)
    {
        int i;
        char *ev;
        long len;
        long idx = 1;
        n = ed_scan_n (event,event_len);
        for(i=0;i<n;i++,idx++)
        {
            ev =(char *)ed_scan_get_n ((char *)event, idx, event_len, &len);
            free (ev);
        }
    }
    return 0;
}
tec_register_callback(th,on_message)
```

---

## Appendix B. Utilities for the C API

Tivoli Event Integration Facility provides a number of utilities for the C API.

---

### ed\_scan\_get\_n

Events received through the reception API can contain more than one event. Use the `ed_scan_get_n` utility to get the *n*'th event from the packet.

#### Synopsis

```
char * ed_scan_get_n (char *packet, long index, long packet_len, long *result_len);
```

#### Arguments

**packet** The pointer returned by `tec_get_event` or passed to the callback when `tec_register_callback` is used.

**index** The *n*'th element in the packet starting at 1.

**Packet\_len**

The maximum length of the packet to search.

**Result\_len**

Contains the resulting length of the packet.

#### Examples

```
#include "tec_eeif.h"
char *package
char *ev;
int i;
long n,len,idx=1;
package=tec_get_event(th);
n = ed_scan_n (package, strlen (package));
for (i = 0; i < n; i++, idx++)
{
    ev =(char *) ed_scan_get_n ((char *) package, idx, strlen (package),&len);
    free (ev);
}
```

#### Return codes

Returns a pointer to a newly allocated buffer containing the requested event. This pointer must be freed.

---

### ed\_scan\_n

Events received through the reception API can contain more than one event. You use the `ed_scan_n` utility to determine the number of events contained in the packet.

#### Synopsis

```
long ed_scan_n (char *packet, long packet_len );
```

## Arguments

**packet** The pointer returned by `tec_get_event` or passed to the callback when `tec_register_callback` is used.

### Packet\_len

The maximum length of the packet to search.

## Examples

```
#include "tec_eeif.h"
char *package
char *ev;
int i;
long n,len,idx=1;
package=tec_get_event(th);
n = ed_scan_n (package, strlen (package));
for (i = 0; i < n; i++, idx++)
{
    ev =(char *) ed_scan_get_n ((char *) package, idx, strlen (package),&len);
    free (ev);
}
```

## Return codes

Returns the number of events, zero (0) if none are found, and -1 when an error occurs.

---

## ed\_sleep

The `ed_sleep` utility pauses running for the specified duration, and allows threads to switch. This utility is called by managed node adapters in the main loop to release the execution of the internal threads.

## Synopsis

```
int ed_sleep (long seconds, long millis);
```

## Arguments

**millis** Specifies the amount of milliseconds.

### seconds

Specifies the amount of seconds.

## Examples

```
#include "tec_eeif.h"
/* pause for 3.5 seconds */
ed_sleep (3, 500);

/* sleep duration is zero but yields briefly so other threads can run */
ed_sleep (0, 0);
```

## Return codes

Returns zero (0).

---

## Appendix C. Java language API

To build an adapter in Java, you must have the Java 1.3.1 (or higher) compiler. The Java API is provided in Jar files.

---

### disconnect

The disconnect function closes any open connection to the event server.

#### Synopsis

```
disconnect()
```

#### Examples

```
public synchronized void disconnect()
```

---

### disconnect(time)

The disconnect(time) function flushes the cache and then closes any open connection to the event server.

#### Synopsis

```
disconnect (max_seconds_to_wait)
```

#### Arguments

**Note:** For all values, control is returned to the caller as soon as the cache is emptied.

- < 0     Wait forever or until the cache is emptied. If the server is down, this routine will not return.
- = 0     Send events from the cache while the connection remains up. This could take a long time if the cache is large.
- >= 0    Stop emptying the cache after the specified number of seconds.

#### Examples

```
public synchronized void disconnect(int max_seconds_to_wait)
```

---

### getConfigVal

The getConfigVal function retrieves the value of a variable contained in the configuration file.

#### Synopsis

```
getConfigVal (String key)
```

**key**     Specifies the configuration keyword label.

#### Examples

```
public String getConfigVal(String key)
```

## Return codes

Returns the string value associated with *key* in the configuration file used to initialize the TECAgent function.

If *key* is not in the configuration file, null is returned. For keywords such as Filter, which have multiple values, only the last value specified in the configuration files is returned.

---

## onMessage

The onMessage function handles the events received asynchronously from the API.

### Synopsis

onMessage (String *event*)

### Arguments

**event** A string contained in the event, which is returned to the application.

### Examples

```
public void onMessage( String event )
```

## Return codes

The application returns true if the received event was processed with success, indicating to the API to remove the event from the cache or persistent log, or both to avoid resending the event in the future.

The application returns false if the event was not processed with success, indicating to the API to resend the event to the event server for further processing. In this case, the event is not removed from the cache or persistent log, or both.

---

## receiveEvent

The receiveEvent function enables an application to receive events synchronously. It receives events from all the event servers specified in the configuration file used to initialize the TECAgent function.

### Synopsis

String receiveEvent()

### Examples

```
public synchronized String receiveEvent()
```

## Return codes

Returns event data that was received by this API. The delimiter TECEvent.TECAD\_EVENT\_END\_CHAR (A) separates the events returned in this string.

### Related reference

“registerListener” on page 59

---

## registerListener

The registerListener function registers the calling application as a listener, enabling the asynchronous reception of events. You must pass an object that implements the IEventProcessing interface as a parameter.

### Synopsis

registerListener (IEventProcessing)

### Arguments

#### IEventProcessing

The application class that implements the IEventProcessing callback.

### Examples

```
public void registerListener (IEventProcessing)
```

### Related reference

“receiveEvent” on page 58

---

## sendEvent

This function sends events to the event servers specified in the configuration file used to initialize the TECAgent function. You must pass a serialized TECEvent as a parameter. If the BufferEvents=YES keyword is specified, the events are cached and persistent before they are sent.

### Synopsis

sendEvent (String *event*)

### Arguments

**event** Event data to be sent to the event server. If *event* is non-null, it must be at least TECEvent.MIN\_EVENT\_LEN characters long, or sendEvent returns immediately with an error.

### Examples

```
public synchronized int sendEvent(String event)
```

### Return codes

Returns the number of bytes sent to the event server, other applications listening for events, or the cache file. A zero return means the event is filtered out. A negative return indicates an error.

---

## TECAgent

The TECAgent function accesses the configuration file and sets the transport mechanism. It is the top-level object that enables the sending and receiving of events to and from the event server.

### Synopsis

TECAgent (Reader configStream, int *deliveryMode*, boolean oneway)

## Arguments

### **configStream**

Object that reads the configuration keywords.

### **deliveryMode**

Specifies the delivery mode. The values are SENDER\_MODE and RECEIVE\_MODE.

### **oneway**

Used for connections to TME adapters on managed nodes. Designates whether calls to sendEvent() returns exceptions to the caller in the event of failure. A value of one (1) means that exceptions are not returned to the caller; a value of zero (0) means that exceptions are returned to the caller.

## Examples

```
public TECAgent(Reader configStream, int deliveryMode, boolean oneway)
```

## Return codes

An exception is raised if the TECAgent cannot be created.

---

## TECEvent

The TECEvent function encapsulates the code for parsing event definitions into a class name and attribute=value pairs.

## Synopsis

```
TECEvent()  
init(String event)
```

## Arguments

**event** The event string to be parsed. Here are some examples of valid event strings:

```
Class1;msg='text.';hostname=artemis;source=TEC;END  
Class2;END  
Class3;msg=theMessage;END
```

A valid event string has the following form:

```
ID SEMICLN ( ID = (STRING | VALUE | EMPTY_STRING) SEMICLN ) * "END"[CTRL_A]
```

The tokens for the event string grammar are explained as follows:

```
SEMICLN := ";"
```

```
EQUALS := "="
```

```
CNTRL_A := "\001"
```

ID := Any non-empty sequence of characters from the set

a-z, A-Z, 0-9, \_, -, .

containing at least one character from

a-z, A-Z

STRING :=

Begins and ends with either single quotes or double quotes. Any embedded quotes that are the same as the quotes being used to delimit the string must be escaped with the same quote character. For example:

- 'embedded single(')' would be written 'embedded single(')'



- "embedded double(") would be written "embedded double("")"
- "embedded single(')" would be written "embedded single(')"

**Note:** STRING tokens cannot contain the NUL character ('\000') or control-A ('\001')

VALUE :=

Any non-empty sequence of characters excluding the following:

- all ASCII control characters (" - ")
- the space character (" ")
- the single quote ("'")
- the equal sign ("=")
- the semi-colon (";")

EMPTY\_STRING := This token represents an empty string. Quotes are not needed for this value.

The first ID token is the class name of the event. The sequence of instances of "ID=(STRING|VALUE);" specifies the slot/value pairs and "END" marks the end of the event. The terminating character ^A is optional. Each ID used as a slot name must be unique with respect to all the other IDs used as slot names and the slot name cannot be "END". There can be an arbitrary amount of whitespace (characters " ", "\t", "\r", "\n") before and after any of the tokens in an event string, with the following exception. If a terminating ^A is present, nothing can appear after it. Examples of valid event strings are as follows:

```
Class1;
    msg='embedded quote ''.' ;
    hostname=artemis;
END
Class2;END^A
Class3; msg = theMessage ; END
Class4;
    msg='Here''s a newline
    rest of msg';
END
```

## Examples

```
public boolean init(String event);
```

## Return codes

The return code for the init() call is true when the event string is parsed successfully, and false if it is not.



---

## Appendix D. Keywords

A configuration file can contain keywords. You use these keywords in order to configure your system.

### Format

Keywords use the following format:

`keyword=value`

Type each keyword on a separate line. Do not use blank spaces in keyword statements unless enclosed in single quotation marks.

**Note:** Adapters do not issue error messages for misspelled keywords or keywords set to a value that is not valid.

**Note:** Not all keywords apply to all adapters, and some adapters have additional keywords specific to them.

A configuration file can contain the following keywords, which are common to most adapters.

#### **BufEvtMaxSize=kilobytes**

Specifies the maximum size, in kilobytes, of the adapter cache file.

The default value is 64 and the minimum size is 8. File sizes below this level are ignored, and 8 KB is used. There is no upper limit for the file size.

This keyword is optional.

The cache file stores events on disk when they cannot be sent to the event server and the BufferEvents keyword is set to YES.

**Note:** If the cache file already exists, you must delete the file for keyword changes to take effect.

#### **BufEvtPath=pathname**

Specifies the full path name of the adapter cache file.

The default path name is `cache.dat`.

This is a required keyword when the BufferEvents keyword is set to YES.

**Note:** If more than one application on the same system uses the Event Integration Facility, ensure that each application has unique values for the path name.

#### **BufferEvents=YES | MEMORY\_ONLY | NO**

Specifies how event buffering is enabled.

**YES** Stores events in the file specified by the BufEvtPath keyword.

#### **MEMORY\_ONLY**

Buffers events in memory.

**NO** Does not store or buffer events.

If UseStateCorrelation=YES and BufferEvents=YES, the API also stores events in files that are specified with the BufEvtPath keyword. The StateCorrelationMaxFileSize and StateCorrelationTotalSize keywords control the size and number of files.

The value is not case-sensitive. The default value is YES. This keyword is optional.

**BufferFlushRate=events\_per\_minute**

Specifies the number of events that are sent per minute. Once the adapter has recovered the lost connection, and there are events in the buffer, the events are sent at this rate per minute. The default value is zero (0); consequently all events are sent in one burst.

This keyword is optional.

**ConnectionMode=connection\_oriented | connection\_less**

Specifies the connection mode to use to connect to the event server. The default value is connection\_less.

**connection\_oriented**

A connection is established at adapter initialization and is maintained for all events sent. A new connection is established only if the initial connection is lost. The connection is discarded when the adapter is stopped. This option can be abbreviated to co or C0.

**connection\_less**

A new connection is established and discarded for each event or group of events that is sent.

This keyword is optional.

**ed\_diag\_config\_file=./diag\_config**

For the C API only, this file generates log and trace messages. The ./diag\_config file must be present, or else logging and tracing do not occur.

To enable logging, specify error or warning in the ./diag\_config file.

To enable tracing, specify trace0, trace1, or trace2 in the ./diag\_config file.

Each level of logging or tracing also includes all levels below it. For example, when you specify warning logging, error logging is automatically enabled.

**Note:** Be aware that increasing the level of tracing produces a large trace output.

When the Event Integration Facility restarts, the API truncates the /tmp/tec\_ed trace file. To avoid truncation, change the Truncate\_on\_restart line from true to false.

This keyword is optional.

**Filter** Works with the FilterMode keyword to determine how events are filtered.

An event matches a Filter statement when each attribute=value pair in the Filter statement is identical to the corresponding attribute=value pair in the event.

A Filter statement must contain the event class, and optionally can include any other attribute=value pair that is defined for the event class.

The format of a filtering statement is as follows:

`Filter:Class=class_name;[attribute=value;...;attribute=value]`

Each statement must be on a single line. The attribute=value pair is case sensitive.

This keyword is optional.

### **FilterCache**

Works with the FilterMode and Filter keywords to determine which events are stored in the cache when events cannot be sent successfully to the event server. To store events in the cache, you must set BufferEvents=YES. An event matches a FilterCache statement when each attribute=value pair in the FilterCache statement is identical to the corresponding attribute=value pair in the event.

A FilterCache statement must contain the event class (class\_name) and can include any attribute=value pair that is defined for that event class. The format of a filtering statement is as follows:

`FilterCache:Class=class_name;[attribute=value;...;attribute=value]`

Each statement must be on a single line. The attribute=value pair is case sensitive. You must specify the Filter keyword, when you use the FilterCache keyword. Additionally, the FilterCache statement must specify the same class or subset of classes that the Filter statement specifies.

This keyword is optional.

**Note:** When using the FilterCache keyword with endpoint adapters, you must set the filtering statements at both locations to the same specifications.

### **FilterMode=IN | OUT**

Specifies whether events that match a Filter or FilterCache statement are sent to the event server (FilterMode=IN) or discarded (FilterMode=OUT).

The default value is OUT. The valid values are IN or OUT, without regard for case. If you set FilterMode=IN, you must have one or more Filter and FilterCache statements defined.

This keyword is optional.

### **FQDomain= YES | NO | *fully.qualified.domain.suffix***

Specifies how the adapter should set the value of the fqhostname attribute of events sent to the event server. This attribute is used to specify the fully qualified host name of the originating host. Possible values for this keyword are:

**YES** The adapter attempts to determine the fully qualified host name. If this is successful, the fqhostname attribute is set to this value; if not, the attribute has a null value. This the default, unless the FQDomain keyword has been specified.

**NO** The fqhostname attribute has a null value. This is the default value if the FQDomain keyword is not specified.

*fully.qualified.domain.suffix*

This value is appended to the host name to set the fqhostname slot.

**Note:** This keyword is valid only for the OpenView, SNMP, UNIX log file, and Windows event log adapters.

**LogFile***Name=pathname*

Specifies the full path name of the log file for the Java API. The default location for the file is \$TIVOLIHOME/tec/eif.log.

The default value for the path is eif.log.

If you specify a non-valid path name, the API returns the following error:

```
LOG0014E Unable to open the handler output file <filename>.
java.io.FileNotFoundException: <filename>
(The system cannot find the path specified)
```

This keyword is optional.

**LogLevel***=level*

Specifies whether the Java API generates log messages or not. By default, no messages are generated.

Specify ALL to generate messages. If you specify any other value or no value, the API does not generate messages.

This keyword is optional.

**MaxPacketSize***=bytes*

Specifies the number of bytes to be sent at the rate specified by the BufferFlushRate keyword.

The default value is zero (0), where one event is sent at a time. This keyword is optional.

**NO\_UTF8\_CONVERSION=YES | NO**

Specifies whether Tivoli Event Integration Facility encodes event data in UTF-8.

When this keyword is set to YES, the Event Integration Facility does not encode event data in UTF-8. The data is assumed to already be in UTF-8 encoding when passed to the Event Integration Facility. It does, however, prepend the flag indicating that the data is in UTF-8 encoding if the flag does not exist at the beginning of the event data.

This keyword is optional. The default value for this keyword is NO.

**Pre37Server=YES | NO**

Specifies whether the adapter sends events in the encoding of the event server host or in UTF-8 encoding.

Event server host versions earlier than the Tivoli Enterprise Console 3.7 product do not support UTF-8 encoding of events.

The following values are not case-sensitive:

**YES** Disables UTF-8 encoding and allows the adapter to communicate with event server host versions earlier than the Tivoli Enterprise Console 3.7 product. When this keyword is set to YES, you must also specify the Pre37ServerEncoding keyword.

**NO** The adapter sends events in UTF-8 encoding.

This keyword is optional. The default value is NO.

**Pre37ServerEncoding***=encoding*

Determines which language to use when a non-TME adapter communicates with a non-UTF-8 event server host (versions earlier than the Tivoli Enterprise Console 3.7 product).

This keyword is active only when the Pre37Server keyword is set to YES.

This keyword is optional.

**RetryInterval=***timeout*

When `ConnectionMode=connection_oriented`, and the connection to the event server is lost, an adapter waits the specified number of seconds before again attempting to connect to the primary or secondary servers, or to buffer the events. While the adapter is waiting for the expiration of this interval, no new events are processed by the adapter.

This option allows an adapter to send all events to the primary event server even if the primary event server is stopped briefly, such as when loading a new rule base. If you use this keyword to wait for restarting an event server, set the value for a period of time longer than necessary for the event server to be stopped and then restarted.

This keyword is optional. The default value is 120 seconds.

**ServerLocation=***host*

Specifies the name of the host on which the EIF receiver is installed. The value of this field must be either `host_name` or `IP_address`.

**Note:** Use the dotted format for `IP_address`.

The `ServerLocation` keyword is optional and not used when the `TransportList` keyword is specified.

**Note:** The `ServerLocation` keyword defines the path and name of the file for logging events, instead of the event server, when used with the `TestMode` keyword.

The `ServerLocation` keyword can contain up to eight values, separated by commas. The first location is the primary event server, while others are secondary servers to be used in the order specified when the primary server is down.

The default value is `localhost`.

**ServerPort=***number*

Specifies the port number on a non-TME adapter only on which the event server listens for events.

Set this keyword value to zero (0), the default value, unless the portmapper is not available on the event server, which is the case if the event server is running on a Microsoft Windows system or the event server is a Tivoli Availability Intermediate Manager (see the following note). If the port number is specified as zero (0) or it is not specified, the port number is retrieved using the portmapper.

**Note:** Portmapper is not supported for reception of events from non-TME adapters at the Tivoli Enterprise Console gateway. If your non-TME adapter is sending events to this gateway, then you must code the `ServerPort` keyword to match the value in the `gwr_ReceptionPort` keyword in the Tivoli Enterprise Console gateway configuration file.

The `ServerPort` keyword is optional and not used when the `TransportList` keyword is specified.

The `ServerPort` keyword can contain up to eight values, separated by commas. For adapters that send events to a UNIX event server, use the default value of 0 (only one value of 0, even if multiple UNIX event servers are specified with the `ServerLocation` keyword). For adapters that

send events to a Windows event server or a Tivoli Availability Intermediate Manager, specify one value for each event server defined with the `ServerLocation` keyword.

The `ServerPort` keyword is optional when the event server is running on the UNIX operating system, but mandatory when running on Windows operating system. It is not used when the `TransportList` keyword is specified.

**TestMode=YES | NO**

Specifies whether test mode is turned on or off.

When `TestMode=YES`, the `ServerLocation` keyword specifies the file to which events are logged, instead of being sent to the event server. Valid values are `YES` and `NO`, without regard to case. The default is `NO`.

This keyword is optional.

**TraceFileName=pathname**

Specifies the full path name of the trace file for the Java API.

This keyword is optional.

The default location of the file is `$TIVOLIHOME/tec/eif.trc`.

If you specify an invalid path name, the API returns the following error:

```
LOG0014E Unable to open the handler output file <filename>.
java.io.FileNotFoundException: <filename>
(The system cannot find the path specified)
```

**TraceLevel=level**

Specifies whether the Java API generates trace messages or not. By default, no messages are generated.

Specify `ALL` to generate messages. If you specify any other value or no value, the API does not generate messages.

This keyword is optional.

**TransportList=type\_name,...**

Specifies the user-supplied names of the transport mechanisms, separated by commas. When a transport mechanism fails for sender applications, the API uses the following transport mechanisms in the order specified in the list. For receiving applications, the API creates and uses all the transport mechanisms.

**Note:** This keyword is supported only for Solaris, AIX, Linux, and Windows adapters. It is not supported for other adapters.

This keyword is optional. If it is specified, the transport type and channel for each `type_name` must be specified using the `Type` and `Channels` keywords:

**type\_nameType=SOCKET | SSL**

Specifies the transport type for the transport mechanism specified by the `TransportList` keyword.

This keyword is required.

The server and port for each `channel_name` are specified by the `ServerLocation` and `Port` keywords.



*type\_name***Channels=channel\_name,...**

Specifies the user-supplied names of the channels for the transport mechanism specified by the **TransportList** keyword, separated by commas.

This keyword is required.

**Depending on the Type (SOCKET or SSL), you use the following keywords:**

*channel\_name***Port=number**

Specifies the port number on which the transport mechanisms server listens for the specified channel, as set by the **Channel** keyword.

When this keyword is set to zero (0), the portmapper is used. This keyword is required.

*channel\_name***PortMapper=YES | NO**

Enables the portmapper for the specified channel.

This optional keyword is valid only when the transport type is set to **SOCKET**.

*channel\_name***PortMapperName=name**

If the portmapper is enabled, specifies the name of the portmapper. This optional keyword is valid only when the transport type is set to **SOCKET**.

*channel\_name***PortMapperNumber=rpc\_id**

Specifies the ID registered by the remote procedure call. This optional keyword is valid only when the transport type is set to **SOCKET**.

*channel\_name***PortMapperVersion=version\_number**

If the portmapper is enabled, specifies the version of the portmapper. This optional keyword is valid only when the transport type is set to **SOCKET**.

*channel\_name***ServerLocation=server[region]**

Specifies the hostname or IP address of the server and region on which the server for transport mechanisms is located for the specified channel.

The channel is set by the **Channel** keyword. This keyword is required.

*channel\_name***SSLKeystore=pathname**

Specifies the path to the keystore containing the keys and certificates that will be used during SSL authentication.

This keyword is required when the transport **Type** keyword is set to **SSL**.

*channel\_name***SSLKeystorePW=password**

Specifies the password of the keystore defined in *channel\_name***SSLKeystore**.

This keyword or the *channel\_name***SSLKeystoreStashFile** keyword is required when the transport **Type** keyword is set to **SSL**.

*channel\_name***SSLKeystoreStashFile=pathname**

Specifies the pathname to the stash file which contains the

password for the keystore defined in the *channel\_nameSSLKeystore* keyword.

This keyword or the *channel\_nameSSLKeystorePW* keyword are required when the transport type is set to SSL.

*channel\_nameSSLKeystoreEncryptionKeyFile=pathname*

Specifies the pathname to the file which contains the encryption key which can be used to decode the data contained in *channel\_nameSSLKeystoreStashFile*.

This keyword is only recognized by the Java API and is required only when *channel\_nameSSLKeystoreStashFile* has been specified.

*channel\_nameSSLTruststore=pathname*

Specifies the path to the keystore containing the keys and certificates that will be trusted by the Event Integration Facility during SSL authentication.

This keyword is only recognized by the Java API. If the keyword is not specified, the file defined in *channel\_nameSSLKeystore* will be used as the trust store.

*channel\_nameSSLTruststorePW=password*

Specifies the password for the keystore defined in *channel\_nameSSLTruststore*.

This keyword is recognized only by the Java API.

*channel\_nameSSLTruststoreStashFile=pathname*

Specifies the pathname to the stash file which contains the password for the keystore defined in *channel\_nameSSLTruststore*.

This keyword is recognized only by the Java API.

*channel\_nameSSLTruststoreEncryptionKeyFile=pathname*

Specifies the pathname to the file which contains the encryption key which can be used to decode the data contained in the *channel\_nameSSLTruststoreStashFile* file.

This keyword is recognized only by the Java API and is required only when the *channel\_nameSSLTruststoreStashFile* keyword has been specified.

*channel\_nameSSLCipherList=cipher1,cipher2,cipher3,...*

Specifies the ciphers that will be permitted during SSL authentication, delimited by commas.

If the list is not specified, all available ciphers will be permitted.

*channel\_nameSSLFIPSMODE=ON | OFF*

Specifies whether the SSL handshake will operate in FIPS 140-2 mode or not. The default value is OFF.

**Note:** FIPS mode restricts the types of ciphers that can be used.

*channel\_nameSSLRequireClientAuthentication=YES | NO*

Specifies whether an Event Integration Facility application

acting as a server will require a client to present a certificate during an SSL handshake.

This keyword has no effect when specified in the client configuration, only in the server configuration.

The default value is N0.

#### **Related concepts**

“Configuration files” on page 3

“Connection options” on page 13

“Selecting event delivery methods” on page 29

“Configuration file APIs” on page 34

“Configuring adapters for international environments” on page 37

“Message logs” on page 43

“Trace logs” on page 43

#### **Related tasks**

“Programming the adapter” on page 33

“Filtering with configuration files” on page 39

“Configuring an EIF receiver application for SSL” on page 29

“Configuring an EIF client application for SSL” on page 31

#### **Related reference**

“Sending events through firewalls” on page 21

“Event reception for applications” on page 20

“SSL and FIPS 140-2” on page 14

“Adapter files” on page 27

“Running adapters built with the Event Integration Facility Java API” on page 37



---

## 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 wish 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  
958/NH04  
IBM Centre, St Leonards  
601 Pacific Hwy  
St Leonards, NSW, 2069  
Australia

IBM Corporation  
896471/H128B  
76 Upper Ground  
London SE1 9PZ  
United Kingdom

IBM Corporation  
JBF1/SOM1  
294 Route 100  
Somers, NY, 10589-0100  
United States of America

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 document 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.

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.

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.

#### COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

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

---

## Trademarks

These terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

AIX  
IBM  
Netcool  
OS/390  
Passport Advantage  
System z  
Tivoli  
Tivoli Enterprise Console  
TME  
z/OS  
zSeries

Adobe, Acrobat, Portable Document Format (PDF), PostScript, and all Adobe-based trademarks are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, other countries, or both.

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

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

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation 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.





---

# Index

## Numerics

64-bit libraries 1

## A

accessibility viii

adapters

building 33

overview 27

built with the C API

compiling 35

linking to libraries 36

C APIs 47

compiling 33

configuring 36

event classes 28

event delivery methods 29

files 27

identifying events 27

installing 36

internationalization 37

Java API 57

migrating 12

multithreaded 33

overview 2

required Java API files 37

sample code 33

testing 36

upgrading 34

Windows considerations 35

API utilities

C

ed\_scan\_get\_n 55

ed\_scan\_n 55

ed\_sleep 56

APIs

C 47

tec\_agent\_getenv 47

tec\_agent\_init 47

tec\_create{EIF\_handle 48

tec\_create\_handle\_c 49

tec\_create\_handle\_r 51

tec\_destroy\_handle 52

tec\_errno 52

tec\_get\_event 52

tec\_put\_event 53

tec\_register\_callback 53

communications 34

configuration files 34

Java

disconnect 57

disconnect(time) 57

getConfigVal 57

onMessage 58

receiveEvent 58

registerListener 59

sendEvent 59

TECAgent 59

TECEvent 60

APIs (*continued*)

overview 47

attributes

overview 1

audience v

availability

events 44

## B

backup

servers configuration 23

BAROC files 3

## C

C API

reception application

configuration 24

utilities 55

cache 21

activating 22

certificate 14

ciphers 14

CLI commands 17

communications

APIs 34

configuration file

keywords 63

configuration files

APIs 34

filtering events 39

connection parameters

event reception 44

connections

connection-oriented delivery 13

connectionless delivery 13

content summary v

conventions, typeface ix

## D

daemon

portmapper 24

directory structure 10

disconnect 57

disconnect(time) 57

downloading

EIF 9

## E

ed\_scan\_get\_n 55

ed\_scan\_n 55

ed\_sleep 56

education

see Tivoli technical training viii

environment variables, notation ix

event classes 28

event delivery

connection options 13

selecting methods 29

transport options 13

Event Integration Facility

overview 1

what's new

64-bit libraries 1

FIPS 140-2 1

IPv6 support 1

SSL 1

transport types 1

event servers 3

event transport

overview 13

events 40

availability 44

classes 3

defining classes 28

delivery methods 13

delivery when systems fail 21

example 3

filtering 6, 39

firewalls 21

flow 27

identification 3, 27

monitoring 28

reception

example 20

reception connection parameters 44

state correlation 39

translated 1

## F

files

root.baroc 3

filtering

configuration files 39

regular expressions 41

state correlation 39

filtering events

system failures 40

filtering when system fails 40

filters

event filtering overview 6

FIPS 140-2 1, 14

introduction 17

firewalls

sending events 21

functions

disconnect 57

disconnect(time) 57

getConfigVal 57

onMessage 58

receiveEvent 58

registerListener 59

sendEvent 59

tec\_agent\_getenv 47

tec\_agent\_init 47

tec\_create{EIF\_handle 48

functions (*continued*)

- tec\_create\_handle 49
- tec\_create\_handle\_c 50
- tec\_create\_handle\_r 51
- tec\_destroy\_handle 52
- tec\_errno 52
- tec\_get\_event 52
- tec\_put\_event 53
- tec\_register\_callback 53
- TECAgent 59
- TECEvent 60

## G

getConfigVal 57

## I

iKeyman 13, 14

installation

- migration 12
- overview 9
- preparing 9

installing 10

IPv6 support 1

## J

Java API

- overview 57

## K

keystore 14

keywords

- configuration file 63
- configuration files 3
- list 63

## L

libeif.a library 13

logs

- messages 43
- trace 43

## M

manuals vi

message logs 43

migrating adapters 12

monitoring

- events 28

## O

online publications vi

onMessage 58

ordering publications vi

## P

performance

- preventing overloading 44

portmapper daemon 24

problem determination

- troubleshooting 43

publications vi

## R

receiveEvent 58

recovery

- systems 21

registerListener 59

regular expressions

- filtering 41

rules 6

## S

sendEvent 59

SOCKET 13

SSL 1, 13, 14

stash file 14

support information viii

system failures 21

## T

Tcl regular expressions

- filtering events 41

tec\_agent\_getenv 47

tec\_agent\_init 47

tec\_create{EIF\_handle 48

tec\_create\_handle 49

tec\_create\_handle\_c 50

tec\_create\_handle\_r 51

tec\_destroy\_handle 52

tec\_errno 52

tec\_get\_event 52

tec\_put\_event 53

tec\_register\_callback 53

TECAgent 59

TECEvent 60

Tivoli software information center vi

Tivoli technical training viii

trace logs 43

training, Tivoli technical viii

transport

- options 13

transport types 1

troubleshooting

- building and running adapters 45
- common scenarios 45
- connection error when using
- postEIFmsg 45
- event reception connection
- parameters 44
- events sent error
- general 46
- using postEIFmsg 46
- message logs 43
- overview 43
- trace logs 43

truststore 14

typeface conventions ix

## U

utilities

- C API 55
- ed\_scan\_get\_n 55
- ed\_scan\_n 55
- ed\_sleep 56

## V

variables, notation for ix

## W

Windows

- special considerations 35





Printed in the Republic of Ireland

SC14-7611-00

