VisualAge Smalltalk

Tivoli® Connection Guide and Reference

Version 5.5

Note

Before using this document, read the general information under “Notices” on page v.

August 2000

This edition applies to Version 5.5 of the VisualAge Smalltalk products, and to all subsequent releases and modifications until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product. The term “VisualAge,” as used in this publication, refers to the VisualAge Smalltalk product set.

Portions of this book describe materials developed by Object Technology International Inc. of Ottawa, Ontario, Canada. Object Technology International Inc. is a subsidiary of the IBM® Corporation.

If you have comments about the product or this document, address them to: IBM Corporation, Attn: IBM Smalltalk Group, 621-107 Hutton Street, Raleigh, NC 27606-1490. You can fax comments to (919) 828-9633.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. 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 of the intellectual property rights of IBM may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY, USA 10594.

IBM may change this publication, the product described herein, or both.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

- IBM
- International Business Machines
- VisualAge®

The following terms are trademarks of Tivoli Systems Inc.:

- Tivoli
- Tivoli Enterprise
- Tivoli Enterprise Console™
- Tivoli Management Framework™

Windows® is a trademark of Microsoft® Corporation.

Chapter 1. Introduction to Tivoli Connection for Windows

Distributed applications and the rise of network computing create a complex environment where business applications require resources from the network, databases, servers, and the internet. Where resources and applications are business-critical, they must be managed to ensure their availability. The Tivoli Management Framework provides an applications management solution that covers the life cycle of an application, from deployment to monitoring and administration.

With the Tivoli Connection feature, you can develop VisualAge applications that can be managed by Tivoli Enterprise products. By instrumenting your application with Tivoli events, you can communicate status information to the Tivoli Enterprise Console (TEC). With Tivoli Module Designer (TMD), you can define important management characteristics of your application to Tivoli Enterprise.

What you get with Tivoli Connection

The Tivoli Connection feature supplies the components for enabling development of VisualAge applications that are Tivoli management-ready:

- Set of parts that allows you to generate standard Tivoli events from your application
- An interface that sends your events to the Tivoli Enterprise Console

Tivoli Event parts

The set of parts provides interfaces to standard Tivoli events that include application start-up, application close, and application error. By adding these parts to your application, you can send events to the Tivoli Enterprise Console. Events can be used to alert the TEC of significant changes in the state of your application, such as error conditions or successful completion of tasks.

Both visual and non-visual interfaces to the event parts are provided. From the Composition Editor, you can find the visual parts in the Tivoli Connection category on the parts palette. A sample application illustrates how to use these parts. The non-visual parts are defined in the *TivoliConnectionRunApp* application.

Tivoli Enterprise Console interface

In order to run your Tivoli-enabled application, you must include the dynamic link library, TECEIF.DLL. This DLL provides the interface between the events generated by your application and the event integration facility of the Tivoli Enterprise Console.

Tivoli Module Designer

The **Tivoli Module Designer** is a tool for creating management-ready applications. It automates the creation of Application Description Files (ADFs), which contain the relevant management information for an application, including application components, dependencies, installation requirements, directory path names, monitors, tasks and component relationships. The ADFs conform to the **Applications Management Specification (AMS),** a formal specification for communicating information about an application between development and

operations. Developed in collaboration with customers and development tool partners, AMS describes both the content and the format of the information which must be transferred. The format is based on the industry-standard MIF format as defined by the Desktop Management Task Force. With AMS, developers have a standard way to communicated everything they know, and an administrator needs to know, about an application.

Both Tivoli Module Designer and the AMS can be downloaded for free from Tivoli's web site.

Chapter 2. Setting up Tivoli Connection

In order to use the Tivoli Connection feature, you must first install the feature and set up your interface to the Tivoli Enterprise Console.

Installing Tivoli Connection

You can install Tivoli Connection by selecting **Load Features...** from the **Options** menu on the VisualAge Organizer window. This will add the *TivoliConnectionRunApp* and *TivoliConnectionEditApp* applications to your VisualAge Organizer window.

Setting up the Tivoli Enterprise Console interface

The Tivoli Enterprise Console is a Tivoli product that collects event information from a wide variety of sources, correlates the information to determine root-cause problems, and performs automated responses. The Tivoli Connection feature assumes that you already have TEC installed on a server in your network.

To run your Tivoli-enabled application, you need two files:

1. **TECEIF.DLL**

   This DLL provides the interface between the events generated by your application and the event integration facility of the Tivoli Enterprise Console.

2. **TECEIF.CFG**

   The configuration file specifies the name of your TEC server and describes your application. This file must be located in the same directory as the TECEIF.DLL file. The TECEIF.DLL file reads this file when it is initialized.

   The following entries are defined in the configuration file:

   Server Machine name that receives the events.

   TestMode
   If YES, events are not sent to the server and the TECEIF.DLL creates a TECEIF.LOG file in its directory to log events.

   If NO, events are sent to the server without logging.

   Application
   Name of the application. If the Developer Kit has been used to define the application, then this is the name found in the Global Description File for the application.

   AppVersion
   Version of the application. If the Developer Kit has been used to define the application, then this is the version specified in the Global Description File for the application.

   AdminRevision
   An administrator revision string supplied when the application is distributed.

   Component
   Name of the component. If the Developer Kit has been used to define the application component, then this is the name found in the Component Description File.

A sample file might contain the following statements:

- `Server=TECServer`
- `TestMode=YES`
- `Application=MyApplication`
- `AppVersion1.0.0`
- `AdminRevision=Rev1`
- `Component=MyComponent`

The default TECEIF.CFG file is defined as:

- `Server=unknown`
- `TestMode=Yes`

Chapter 3. Building an application with Tivoli Connection

The event parts in Tivoli Connection allow you to easily build an application that generates asynchronous events and sends them to the Tivoli/Enterprise Console. You can include event parts in your application in two ways:

- Visually, by selecting event parts from the Tivoli Connection category on the parts palette of the VisualAge Composition Editor.
- Non-visually, by sending messages in your scripts to the event parts.

Supported events

The event parts provide the interface to the TECEIF.DLL and define the following events:

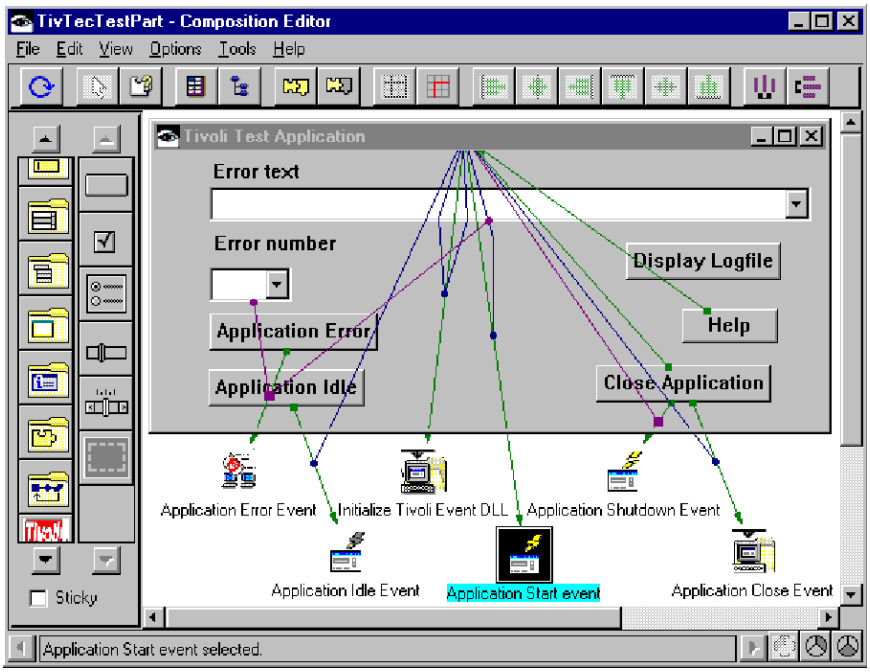
- **Initialize Tivoli Event DLL** initializes the TECEIF.DLL.
- **Application Startup** notifies the console that the application has started.
- **Application Idle** notifies the console that the application has been idle for a given period of time.
- **Application Error** is sent when a runtime error occurs.
- **Application Send** is a generic event that you can define using the Smalltalk message interface.
- **Application Shutdown** is sent when an application is shutting down.
- **Application Close** closes the TECEIF.DLL.
- **SQL Error** notifies the console that an SQL error has occurred.

Adding events visually

The events are located in the Tivoli Connection category on the parts palette of the Composition Editor. First, select a Tivoli event part and drop it on the free-form surface of your application. Then connect an application event to the action of the part. For example, you might connect the *openedWidget* event for your application window to the *Application Start* action of the part, **Application Start Event**.

A sample application, **TivTecTestPart** illustrates how to do this and can be found in *TivoliConnectionRunApp*. The following figure shows the sample application in

the Composition Editor.



This sample shows how to visually connect event parts to add the following events:

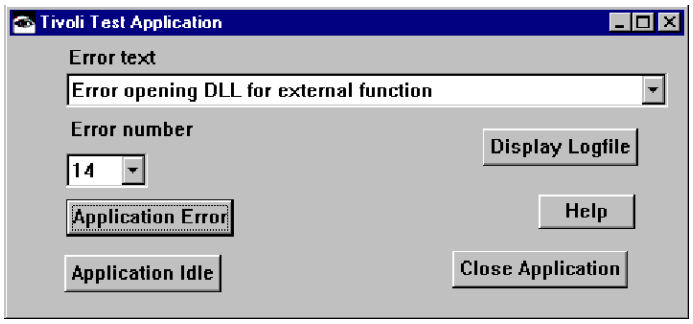
- Application Startup
- Application Idle
- Application Error
- Initialize Tivoli Event DLL
- Application Shutdown
- Application Close

The **Initialize Tivoli Event DLL** and **Application Startup** events are generated when the *openedWidget* event is sent to the application window.

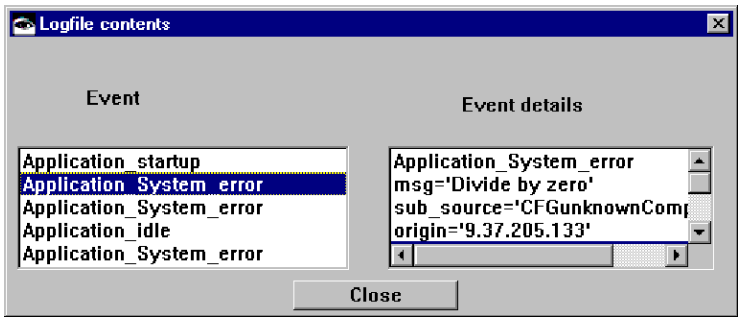
The **Application Idle**, **Application Error**, **Application Shutdown**, and **Application Close** events are generated by the *clicked* event for the associated push buttons. Error messages and error numbers can be selected from the **Error text** or **Error number** drop-down boxes and used as input to the **Application Error** event.

This application should be run in test mode (TestMode=YES in TECEIF.CFG), so that you can view the generated events in the logfile. The **Display Logfile** push button displays the events that were generated and recorded in the logfile, TECEIF.LOG, while running the sample application in test mode.

A test application created from **TivTecTestPart** is shown in the following figure.



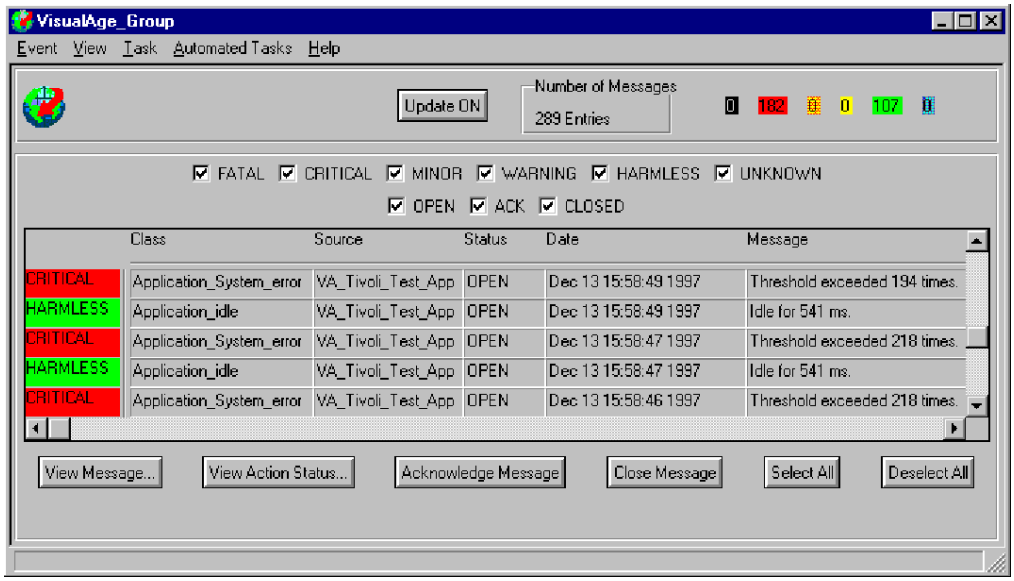
By selecting the **Display Logfile** push button you can view the contents of the logfile as illustrated below.



The left box lists the generated events. When an event is selected, the detailed information about the event is displayed in the **Event details**box on the right.

Select the **Close** push button to return to the Test application.

The following figure illustrates how Tivoli events appear on the Tivoli Enterprise Console.



Adding events as messages

You can also add events to your application by adding event messages to your scripts, or methods. The following sections describe the message interface for each event.

Initialize Tivoli Event DLL

This event is generated by your application to initialize the TECEIF.DLL. It is sent once and must precede other events.

tvInitTamEvents: source **with**: subSource

where *source*is the application name and *subSource*is the application component name that generates this event

Example: TivInitTAMEvent new **tvInitTamEvents**: 'MyApp' **with**: 'InitComponent'

Application Startup Event

This event is generated when an application is started.

tvSendStartupEvent: aString

where aString is a text message

Example: TivApplicationStartupEvent new **tvSendStartupEvent**: 'MyApp', Date dateAndTimeNow

Application Close Event

This event is generated to close the TECEIF.DLL.

tvTecClose

Example: TivApplicationCloseEvent new **tvTecClose**

Application Shutdown Event

This event is generated when an application is shutting down.

tvSendShutdownEvent: aString

where aString is a text message

Example: TivApplicationShutdownEvent new **tvSendShutdownEvent**: 'MyApp ',Date dateAndTimeNow

Application Idle Event

This event is generated when the program enters an extended period of inactivity.

tvSendIdleEvent: aString

where aString is a text message

Example: TivApplicationIdleEvent new **tvSendIdleEvent**: 'MyApp'

Application Error Event

This event is generated when a critical application error occurs.

tvSendApplicationError: errorNum **with**: errorString

where errorNum is an error number and errorString is an appropriate error message string

Example: TivApplicationErrorEvent new **tvSendApplicationError**: 14 **with**: 'myapp.dll not found'

SQL Error

This event is generated when an application encounters an error when interfacing with an SQL database. If information is not available for some of the database fields, an empty string should be specified.

tvSendSQLError: anArray

where *anArray*is an Array object that contains:

- at: 1 - an Integer that contains standard SQL error code,
- at: 2 - a String object that contains name of database server,
- at: 3 - a String object that contains the name of the database,
- at: 4 - a String object that contains the name of the table in the database,
- at: 5 - a String object that contains an appropriate message.

Application Send Event

This event is used to send a generic event to the TECEIF interface. If you define a new event, it must also be defined to the Tivoli Enterprise Console. See the next section for an example of how to define a generic event.

Chapter 4. Sending generic events

The events described so far represent standard event classes that are pre-defined in the TECEIF.DLL. The **Application Send** event is a generic event that is pre-defined, but can be customized by supplying your own parameters.

The following sample code shows how to implement a generic application event.

```
| tec nameValueArray nameValuePairs nameValue anArray initArray rc1 |

nameValuePairs := 2.
nameValueArray :=TecEifNameValue new: 2.
nameValueArray at: 0 put: (
  TecEifNameValue new
  szName: 'date';
  bInteger:  0;
  szValue: 'Tue Aug 5 23:59:57 1997';
  nValue: 23).
nameValueArray at: 1 put: (
  TecEifNameValue new
  szName: 'CommonWidgets';
  bInteger: 0;
  szValue: 'CwFileSelectionPrompter';
  nValue: 17).
tec := TivTecEifWrapper new.
anArray := Array new: 5.
anArray at: 1 put: (tec nullTerminate: 'unknown class');
    at: 2 put: (tec nullTerminate: (TecEifConstants at: 'EifHarmless'));
    at: 3 put: 2;
    at: 4 put: nameValueArray  abtAsExternalPassedPointer;
    at: 5 put: (tec nullTerminate: 'string').

initArray := Array new: 3.
initArray  at: 1 put: (tec nullTerminate: 'e:\visualag\teceif.cfg').
initArray  at: 2 put: (tec nullTerminate: 'My test Application' ).
initArray  at: 3 put: (tec nullTerminate: 'ComponentId').

rc1 := tec tecTvInitEvents: initArray.
tec tecTvSendEvent: (anArray ).
tec tecTvTecClose.
```

In this example,

> `nameValueArray` defines two values that are passed as parameters: a date string and a string that defines the widget that generates the event.
>
> `anArray` defines the five parameters to the event: class name, severity, `nameValueArray` size, `nameValueArray` , and a descriptive string.
>
> `initArray` defines three parameters for initializing the TECEIF.DLL: name of the configuration file, application name, and component name.

The last three statements in this example

- Initialize TECEIF.DLL from `initArray`
- Send the generic event with parameters in `anArray`
- Close the TECEIF.DLL

Chapter 5. Generating Tivoli Events from Smalltalk Exceptions

When exceptions occur in your application, you may want to represent them as events that are sent to the Tivoli Enterprise Console. The **TivoliExceptionsApp** defines three classes that assist in generating an **Application Error** event when a Smalltalk exception occurs.

TivExceptionalEvent is a subclass of **ExceptionalEvent** that provides an interface to the TECEIF.DLL and generates an **Application Error** event when an instance is signaled.

TivExceptionalEventRoot is a subclass of **Object** that defines a TivExceptionalEvent instance that is the root, or parent, of all **TivExceptionalEvent** instances.

TivSampleExceptionsCheckingAccount is a sample class that illustrates how to define and signal TivExceptionalEvents.

Defining a TivExceptionalEvent

The **TivSampleExceptionsCheckingAccount** class methods define two exceptions that are represented as class variables. One of the exceptions, TivExCheckingAccountOverdrawn, is defined as

```
anExceptionalEvent := TivExceptionalEventRoot root newChild.
anExceptionalEvent description: 'Checking Account Overdrawn'.
anExceptionalEvent resumable: false.
anExceptionalEvent defaultHandler:
 [: sig | System errorMessage sig description.
 sig exitWith: 'done']
self TivExCheckingAccountOverdrawn: anExceptionalEvent.
```

Signaling a TivExceptionalEvent

The following **TivSampleExceptionsCheckingAccount** instance method illustrates how the TivExCheckingAccountOverdrawn exception is signaled:

```
withdraw: anAmount
  self balance >= anAmount
  ifTrue:  [self balance: self balance - anAmount]
  ifFalse: [self class TivExCheckingAccountOverdrawn
  signalWithArguments:
  (OrderedCollection
  with: self
  with:(Association key:'balance' value: self balance)
  with:(Association key:'amount' value: self amount))].
```

Using the Sample application

The sample generates a TivExCheckingAccountOverdrawn exception when the following is executed:

```
TivSampleExceptionsCheckingAccount new withdraw: 10000.
```

The sample generates a TivExCheckingAccountInsuredAmountExceeded exception when the following is executed:

```
TivSampleExceptionsCheckingAccount new deposit: 500000.
```

The Tivoli Application Error events that are generated by these exceptions are logged to the file, TECEIF.LOG.

Chapter 6. Defining event classes

The Tivoli Enterprise Console (TEC) provides rule-based management of distributed events. TEC collects, processes, and automatically responds to common management events such as server outages, lost network connections, or successful completion of application tasks. Tivoli Connection assumes that TEC is installed on a server in your network.

TEC events are defined by using the Basic Recorder of Objects in C (BAROC) language. Once a BAROC definition for an event type has been imported into the TEC rule base, events consistent with that definition will be recognized. See the TEC documentation for a detailed description of how to create event definitions and add them to the rule base.

The BAROC definitions for the application events supported by Tivoli Connection are provided below. You must import these definitions, and any others you choose to define, into the TEC rule base. The rule base must then be compiled and loaded before TEC will begin recognizing the event instances as valid events. Refer to the TEC documentation for a detailed description of event definition.

Application event definitions

```
TEC_CLASS:
   Application_Event ISA EVENT
   DEFINES {
   source: default = "Application Name";
   sub_source: default = "Component Name";
   sub_origin: default = "TME Managed Node Name";
   deployment_map: STRING ;
   };
END
TEC_CLASS:
   Application_notice ISA Application_Event
   DEFINES {
   severity: default = HARMLESS;
   };
END
TEC_CLASS:
   Application_warning ISA Application_Event
   DEFINES {
   severity: default = WARNING;
   };
END
TEC_CLASS:
   Application_error ISA Application_Event
   DEFINES {
   severity: default = CRITICAL;
   };
END
TEC_CLASS:
   Application_startup ISA Application_notice;
END
TEC_CLASS:
   Application_idle ISA Application_notice;
END
TEC_CLASS:
   Application_shutdown ISA Application_notice;
END
TEC_CLASS:
```

```
   Application_System_error ISA Application_error
   DEFINES {
   error_number: INTEGER;
   };
END
TEC_CLASS:
   Application_SQL_error ISA Application_error
   DEFINES {
   sqlcode: INTEGER;
   database: STRING;
   table: STRING;
   sqlca: STRING;
   };
END
```

The following event slots are defined for application events:

source Application Name, as it appears in the ″Application Name″ attribute of the Global Description File for the application. This field can be overridden in the configuration file.

sub_source
Component Name, as it appears in the ″Component Name″ attribute of the Component Description File for the component. This field can be overridden in the configuration file.

origin IP address

sub_origin
operating system and version

hostname
local hostname, as obtained by the gethostname() function or equivalent

The library populates event slots as follows:

- **class** from caller

  source application name as passed in 2nd argument to **tvInitTamEvents()**

  sub_source
  component name as passed in 3rd argument to **tvInitTamEvents()**

  origin IP address of local system discovered by the TECEIF DLL

  sub_origin
  Managed Node name from configuration file if available

  hostname
  from local gethostname() function or equivalent

  adapter_host
  same as hostname

  severity
  from caller, if non-NULL, otherwise default severity

  message
  from caller

Chapter 7. Tivoli Help

VisualAge ManageWare for Tivoli Event Components

The following VisualAge for Smalltalk interfaces to the TECEIF DLL have been defined to allow the following events can be sent from your program:

- Initialize Tivoli Event DLL
- Application Startup Event
- Application Idle Event
- Application Error Event
- Application Send Event
- Application Close Event
- SQL Error

Note: The path to the configuration file is assumed to be the same as the event DLL, TECEIF.DLL.

The installation program will put the configuration file (TECEIF.CFG) and the event DLL (TECEIF.DLL) in your root VisualAge directory.

Initialize Tivoli Event DLL

ClassName: *TivInitTamEvents*

Message interface: tvInitTamEvents: source with: subSource

This event is issued once by your application to initialize the event DLL.

The source is the application name.

The subSource is the application component name that generates this event.

Application Startup Event

This event should be sent during program startup.

```
tvSendStartupEvent: aString
```

Where *aString* is a String Object with an appropriate message.

Application Idle Event

This event is sent when the program enters an extended period of inactivity and could be a security problem.

```
tvSendIdleEvent: aString
```

Where *aString* is a String Object with an appropriate message

Application Error Event

This event is sent when a critical application error occurs.

```
tvSendApplicationError: errorNum with: errorString
```

Where *errorNum* is an error number and *errorString* is an appropriate error message string.

Application Send Event

The following would be used to send any generic event through the TECEIF interface.

```
tecTvSendEvent
```

See the sample code for detail

Application Close Event

This event should be sent during program shutdown

```
tvSendShutdownEvent: aString
```

Where *aString* is a String Object with an appropriate message.

This message will also generate a tvTecClose event to shutdown TECEIF.

SQL Error

This event is sent when an application has an error interfacing with an SQL database. You may not have information for some of the database fields. You should supply an empty string in these situations.

```
tvSendSQLError: anArray
```

Where *anArray* is an Array object that contains:

- at: 1 - an Integer that contains standard SQL error code,
- at: 2 - a String object that contains name of database server,
- at: 3 - a String object that contains the name of the database,
- at: 4 - a String object that contains the name of the table in the database,
- at: 5 - a String object that contains an appropriate message.

Generating a Tivoli Event from a Smalltalk Exception

The *TivExceptionsApp* includes three classes that assist in generating a Tivoli Application System Error event when a Smalltalk exception occurs.

1. *TivExceptionalEvent* is a subclass of *ExceptionalEvent* that provides an interface to the TECEIF.DLL and generates an Application System Error event when an instance is signalled.
2. *TivExceptionalEventRoot* is a subclass of *Object* that defines a *TivExceptionalEvent* instance that is the root, or parent, of all *TivExceptionalEvent* instances.
3. *TivSampleExceptionsCheckingAccount* is a sample class that illustrates how to define and signal *TivExceptionalEvents*.

Defining a TivExceptionalEvent

The *TivSampleExceptionsCheckingAccount* class methods define two exceptions that are represented as class variables. One of the exceptions, *TivExCheckingAccountOverdrawn*, is defined by

```
anExceptionalEvent := TivExceptionalEventRoot root newChild.
anExceptionalEvent description: 'Checking Account Overdrawn'.
anExceptionalEvent resumable: false.
anExceptionalEvent defaultHandler:
```

```
   [:sig | System errorMessage sig description.
           sig exitWith: 'done'].
self TivExCheckingAccountOverdrawn: anExceptionalEvent.
```

Signalling a TivExceptionalEvent

The following *TivSampleExceptionsCheckingAccount* instance method illustrates how the *TivExCheckingAccountOverdrawn* exception is signalled:

```
withdraw: anAmount
   self balance >= anAmount
      ifTrue:  [self balance: self balance - anAmount]
      ifFalse: [self class TivExCheckingAccountOverdrawn
                  signalWithArguments:
                     (OrderedCollection
                         with:self
                         with:(Association key:'balance'
                                        value:self balance)
                         with:(Association key:'amount'
                                        value:self amount))].
```

Using the Sample application

The sample will generate a *TivExCheckingAccountOverdrawn* exception when the following is executed:

```
TivSampleExceptionsCheckingAccount new withdraw: 10000.
```

The sample will generate a *TivExCheckingAccountInsuredAmountExceeded* exception when the following is executed:

```
TivSampleExceptionsCheckingAccount new deposit: 500000.
```

Note: The Tivoli Application System Error events that are generated by these exceptions are logged to the file, TECEIF.LOG.

Sample Code

The following code sample implements the *tvSendEvent* API in the TECEIF.DLL header:

```
| tec nameValueArray nameValuePairs nameValue anArray initArray rc1 |

nameValuePairs := 2.

nameValueArray :=TecEifNameValue new: 2.

nameValueArray at: 0 put: (
TecEifNameValue new
szName: 'date';
bInteger:  0;
szValue: 'Tue Nov 5 23:59:57 1996';
nValue: 23).

nameValueArray at: 1 put: (
TecEifNameValue new
szName: 'CommonWidgets';
bInteger: 0;
szValue: 'CwFileSelectionPrompter';
nValue: 17).

tec := TivTecEifWrapper new.

anArray := Array new: 5.

anArray at: 1 put: (tec nullTerminate: 'unknown class');
```

```
        at: 2 put: (tec nullTerminate: (TecEifConstants at: 'EifHarmless'));
        at: 3 put: 2;
        at: 4 put: nameValueArray  abtAsExternalPassedPointer;
        at: 5 put: (tec nullTerminate: 'string').

initArray := Array new: 3.

initArray  at: 1 put: (tec nullTerminate: 'e:\visualag\teceif.cfg').
initArray at: 2 put: (tec nullTerminate: 'My test Application' ).
initArray at: 3 put: (tec nullTerminate: 'ComponentId').

rc1 := tec tecTvInitEvents: initArray.

tec tecTvSendEvent: (anArray ).

tec tecTvTecClose.
```

Sample Application

This sample application demonstrates the visual use of the following events:

- Initialize Tivoli Event DLL
- Application Startup Event
- Application Idle Event
- Application Error Event
- Application Close Event

These events are located on the VisualAge parts palette in the IBM ManageWare for Tivoli Parts category. In this example, the Initialize Tivoli Event DLL and Application Startup Event events are generated when the application is started (when the openWidget event occurs).

The Application Idle Event , Application Error Event , and Application Close Event events are generated by the appropriate push button.

Error messages and error numbers can be selected from the can be selected from the Error text or Error number drop-down boxes and used as input to the Application Error event.

This application is intended to be run with in test mode, (TestMode=YES in the Configuration file definition).

The Display logfile push button will display the events that were generated and recorded in the logfile, TECEIF.log, during test mode.

TECEIF Description

This section provides a brief description of the approach for enabling an application to send events to the Tivoli/Application Manager console using the Tivoli Event Integration Facility (EIF).

Note: The Tivoli Manager™ of Applications 1.0 product does not support application monitoring using EIF events. Monitoring is available only through synchronous monitors. Monitoring of EIF events will be supported in the Tivoli Manager of Applications 2.0.

- Event Groups
- Configuration file definition
- Header File for TECEIF DLL

- Standard BAROC Definitions for Managed Application
- Standard BAROC Classes for Managed Applications

Event Groups

As T/MA imports new applications, it will generate and install an EIF event group for each application. The event group will be set up with one filter set to the Application Name. The Application Name can be specified by the *tvInitTamEvents* call or can be overridden by an application= entry in the teceif configuration file located on each target machine that is sending events.

These Event Groups will be used to filter events that are delivered to the T/MA console and the T/MA event viewer. From the detailed event view, further filtering by component, severity, origin, deployment group, etc., can be performed. Sends an application event to the TEC. All standard event slots except for class, message and severity are populated by the event DLL (or by the TEC server).

The library will populate event slots as follows:

class: from caller

source:
application name as passed in 2nd argument to `tvInitTamEvents()`

sub_source:
component name as passed in 3rd argument to `tvInitTamEvents()`

origin: IP address of local system discovered by the TECEIF DLL

sub_origin:
will use Managed Node name from configuration file if available

hostname:
from local gethostname() function or equivalent

adapter_host:
same as hostname

severity:
from caller, if non-NULL, otherwise default severity

message:
from caller

Configuration File Description (TECEIF.CFG)

The TECEIF.DLL will look for a TECEIF.CFG configuration file in the same directory as the TECEIF.DLL.

The following entries will be read when the TECEIF.DLL is started:

- Server=
- TestMode=
- Application=
- AppVersion=
- AdminRevision=
- Component=

Server The TME or TMA machine name that will receive the events.

TestMode
YES indicates that the TECEIF.DLL will only create a teceif.log file in the same directory as the TECEIF.DLL and log the events to this file.

Application
The Application name from inside the .gdf file.

AppVersion
The application version from inside the .gdf file. Ex: 1.2.a

AdminRevision
An administrator revision string supplied when the application is distributed.

Component
The component name from inside the .cdf file contains the .exe sending the events.

The default teceif.cfg file will contain the following lines:

```
Server=unknown
TestMode=YES
```

Header File for TECEIF DLL

```
// TECEIF.H
// interface to the TECEIF .DLL to send events to a TEC server

#ifndef _TECEIF_H
#define _TECEIF_H

// only define _TECEIF_DLL in the TECEIF DLL
#ifdef _TECEIF_DLL
#define MYLIBAPI __declspec(dllexport)
#define MYREFAPI
#else
#define MYLIBAPI __declspec(dllimport)
#define MYREFAPI "C"
#endif

// severity defines can ether define or text string
#define EIF_FATAL "Fatal"
#define EIF_CRITICAL "Critical"
#define EIF_SEVERE "Severe"
#define EIF_WARNING "Warning"
#define EIF_HARMLESS "Harmless"
#define EIF_NORAML "Normal"

// a structure for passing name value pairs into send event calls

#define MAX_EIF_NAME   32
#define MAX_EIF_VALUE  256

// structure for name value pairs to be inserted into event
typedef struct tagNameValue
{
  char szName[MAX_EIF_NAME]; // name for slot value
  BOOL bInteger; // use integer or string field
  int  nValue; // integer field 32 bits
  char szValue[MAX_EIF_VALUE]; // string field
} NAMEVALUE, *PNAMEVALUE;

// generic EIF initialization
long WINAPI tvInitEvents(
```

```
    char *szPathName,     // full path to config file
    char *pszSource,     // application name from gdf file
    char *pszSubSource);    // component name from cdf file

// send event call
long WINAPI tvSendEvent(
   char *pszClass,     // class type from BAROC file
    char *pszSeverity,     // severity from list above
    int  nNumValues,     // num of name/value pairs
    PNAMEVALUE pNameValues, // name value pairs to include in event
    char *pszMessage,     // message which can include printf
    ...);     // formatting NOTE: arg list must match
    // formatting in szMessage as the compiler
    // will not check this!

// shut down EIF
long WINAPI tvTecClose();

// TAM specific calls
long WINAPI tvInitTamEvents(
    char *pszPathName,   // full path to config file
    char *pszApplicationName,   // application name from gdf file
    char *pszComponentName);   // component name from CDF file

// event for program startup
long WINAPI tvSendStartupEvent(
    char *pszMessage);     // appropriate message

// event for program in idle state
long WINAPI tvSendIdleEvent(
    char *pszMessage);     // appropriate message

// event for program shutdown
long WINAPI tvSendShutdownEvent(
     char *pszMessage);     // appropriate message

// event for internal program error
long WINAPI tvSendApplicationError(
    int nErrorNum,     // internal error number
    char *pszMessage);     // appropriate message

// event for SQL login or transaction error
long WINAPI tvSendSQLError(
    int nSQLCode,     // standard SQL error code
    char *pszServer,     // name of database server
    char *pszDataBase,     // name of databse
    char *pszTable,     // table in database
    char *pszMessage);     // message

#endif
```

Standard BAROC Definitions for Managed Applications

The following event slots are defined for application events:

source = Application Name, as it appears in the ″Application Name″ attribute of the GDF for the application. This field can be overridden by an application = entry in the EIF configuration file.

sub_source = Component Name, as it appears in the ″Component Name″ attribute of the CDF for the component, This field can be overridden by a component= entry in the EIF configuration file.

origin = IP address

sub_origin = The operating system and version.

hostname = local hostname, as obtained by the gethostname()

function or equivalent

Application developers can either use the standard event classes or derive their own classes from the Application base event class.

Standard BAROC Classes for Managed Applications

```
TEC_CLASS:
    Application_synchronous_monitor ISA Sentry2_0_Base
END
TEC_CLASS:
    Application_process_failure ISA Sentry2_0_Base
END
TEC_CLASS:
    Application_process_restart ISA Sentry2_0_Base
END
TEC_CLASS:
    Application_performance_alert ISA Sentry2_0_Base
END
TEC_CLASS:
    Application_out_of_resources ISA Sentry2_0_Base
END
TEC_CLASS:
    Application_low_on_resources ISA Sentry2_0_Base
END
TEC_CLASS:
    Application_message_logged ISA Sentry2_0_Base
END
TEC_CLASS:
    Application_transaction_failure ISA Sentry2_0_Base
END
TEC_CLASS:
    Application_communication_failure ISA Sentry2_0_Base
END
TEC_CLASS:
    Application_security_alert ISA Sentry2_0_Base
END
TEC_CLASS:
    Application_audit_alert ISA Sentry2_0_Base
END
TEC_CLASS:
    Application_Event ISA EVENT
        DEFINES {
                source: default = "Application Name";
                sub_source: default = "Component Name";
                sub_origin: default = "TME Managed Node Name";
        deployment_map: STRING ;
                };
END
TEC_CLASS:
    Application_notice ISA Application_Event
        DEFINES {
                severity: default = HARMLESS;
                };
END
```

```
TEC_CLASS:
    Application_warning ISA Application_Event
        DEFINES {
                severity: default = WARNING;
                };
END
TEC_CLASS:
    Application_error ISA Application_Event
        DEFINES {
                severity: default = CRITICAL;
                };END
TEC_CLASS:
    Application_startup ISA Application_notice;
END
TEC_CLASS:
    Application_idle ISA Application_notice;
END
TEC_CLASS:
    Application_shutdown ISA Application_notice;
END
TEC_CLASS:
    Application_System_error ISA Application_error
        DEFINES {
                error_number: INTEGER;
                };
END
TEC_CLASS:
    Application_SQL_error ISA Application_error
        DEFINES {
            sqlcode: INTEGER;
            database: STRING;
            table: STRING;
            sqlca: STRING;
            };
END
```