WebSphere. Monitor

Version 6.0.2

DB2 Event Emitter

Contents

Figures

1. Introduction

The Sample DB2 Event Emitter is a sample program written in Java™ that demonstrates how an enterprise information system (EIS) resource (an IBM DB2® database in this sample) storing data pertaining to the state of a business can be instrumented to contribute to the overall monitoring of the activities of a business.

The main goal of the Sample DB2 Event Emitter is to introduce the use of the libraries and application programming interfaces (APIs) provided by the Common Event Infrastructure (CEI) to generate and emit business events in the form of Common Base Events (CBEs). Common Base Events are the data packaging and format used by the IBM WebSphere® Business Monitor (Monitor) Server to propagate business events.

The Sample DB2 Event Emitter is made available in two forms of packaging. One is the binary archive, which contains the precompiled Java 2 Platform, Enterprise Edition (J2EE) application, ready to be deployed to the Monitor Server. It also includes the accompanying documentation and necessary scripts for setup and configuration. The second is the source archive, which also includes the documentation and the setup scripts and, additionally, contains the source code projects that can be imported into the IBM WebSphere Integration Developer (WID) integrated development environment (IDE) for browsing through the source code and making custom changes to the emitter and its configuration options.

The Sample DB2 Event Emitter was developed and tested using the following environment:

- *Microsoft® Windows® XP Professional SP2*
- *IBM DB2 Universal Database™ (UDB) Enterprise Server Edition v8.1.13.193*
- *WebSphere Business Monitor (Monitor) Server v6.0.2*
- *WebSphere Integration Developer (WID) v6.0.2*

Note: The configuration and deployment information in this document describes the Sample DB2 Event Emitter being deployed to the Monitor Server itself (that is, to the same application server on which Monitor runs). The emitter can also be deployed to a separate application server and configured to emit its events to the Monitor Server's CEI Server (see section 3, "Configuring and deploying the Emitter application," for more information on this).

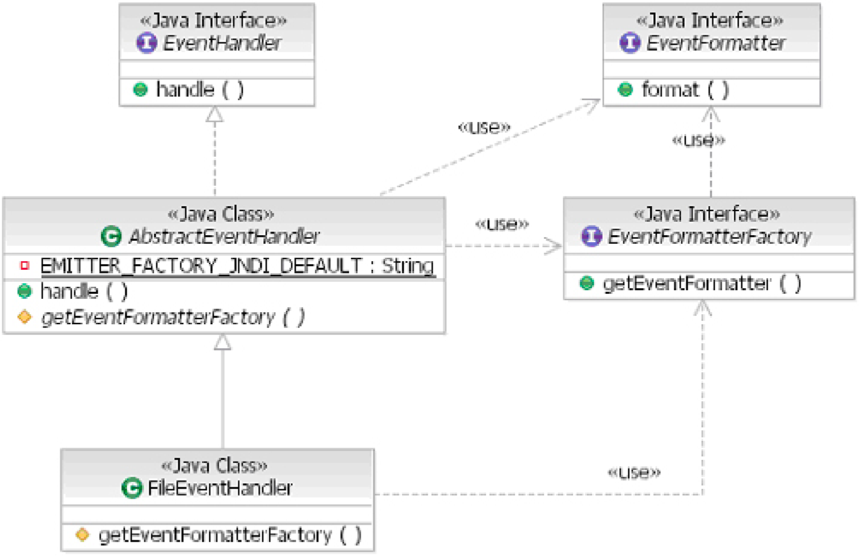
2. Design overview

2.1. Event framework

The Sample DB2 Event Emitter, along with the other sample event emitters made available, is implemented around a common, simple emitter framework. Figure 1 depicts a class diagram of this framework. When an event in the enterprise back-end system is detected, the following flow is regulated by this common framework:

1. *Retrieve, from the EmitterFormatterFactory, an EventFormatter specific for the type of data being processed.*
2. *Invoke the EventFormatter to convert the input data to a CBE object.*
3. *Retrieve an emitter from an EmitterFactory to be used to send the event to the CEI Server.*
4. *Emit the CBE to the CEI Server.*

Figure 1
Event Emitter Framework



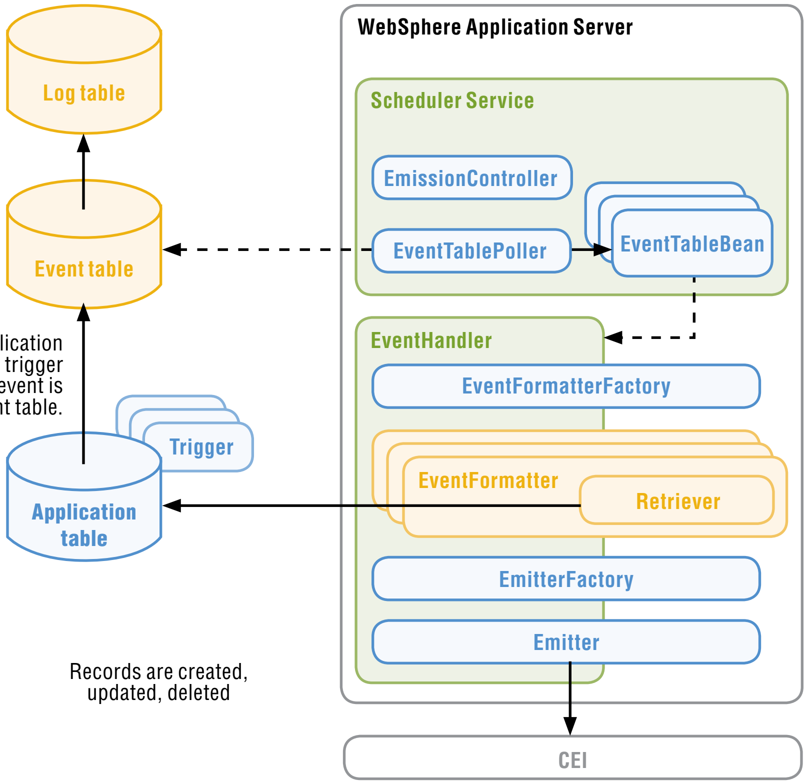
2.2. DB2 Event Emitter

The Sample DB2 Event Emitter is a J2EE enterprise application, which also implements the WebSphere TaskHandler interface, allowing it to handle invocations from a WebSphere Application Server Scheduler. The overall event retrieval and event emission flow is explained below (see Figure 2 for a graphical depiction of this flow):

1. *A record is created, updated or deleted on an application table.*
2. *After the state of the record is changed on the application table, a trigger set to the application table is invoked.*
3. *The trigger inserts a record with the primary key and additional information to the Event table. The additional information includes the trigger type (Create, Update or Delete), the application data type, and the creation timestamp.*
4. *The Scheduler service in the Monitor Server invokes the EmissionController at a specified interval. The EmissionController calls EventTablePoller. If any records have been inserted to the event table, EventTablePoller populates and returns a collection of EventTableBean objects. If the collection of EventTableBean objects is not empty, EmissionController invokes the handle( ) method of EventHandler for each EventTableBean object to handle the emission steps.*
5. *EventHandler calls EventFormatterFactory which returns an EventFormatter object according to the application data type specified by the user. The EventFormatter object transforms the EventTableBean object into a CommonBaseEvent object. Finally, the CEIEmitter sends the CommonBaseEvent object to the CEI Event Server.*

The Sample DB2 Event Emitter is designed so that it can be extended to handle processing of additional application data types. This is done by creating new classes that implement the EventFormatter interface (in the EmitterFW project), extending the AbstractEventRetriever abstract class (in the DBEmitterEJB project), and “registering” the application data type to event formatter association by updating the DB2EventFormatter.properties (in the DBEmitterImpl project). For more-extensive details on this see Section 6: Design details – creating a new event formatter to handle additional application data types.

Figure 2
File Emitter



When the application table is modified a trigger is issued and an event is recorded in the event table.

Records are created, updated, deleted

The **Scheduler Service** invokes the EmissionController at regular intervals

DB2EventFormatter.properties

```
CUSTOMER=com.ibm.wbimonitor.samples.db2emitter.formatter.CUSTOMERFormatterImpl
ORDER=com.ibm.wbimonitor.samples.db2emitter.formatter.ORDERFormatterImpl
CLAIM=com.ibm.wbimonitor.samples.db2emitter.formatter.CLAIMFormatterImpl
```

3. Configuring and deploying the Emitter application

This section describes how to configure the Monitor Server to host the Sample DB2 Event Emitter application. The following section, "Importing and working with the source code," describes how to import the source code into the WID IDE and introduce modifications, if needed.

The following default values are specified in the DB2 Emitter's Enterprise JavaBeans (EJB) deployment descriptor. These can be customized by editing the source projects in WID and exporting a new enterprise archive (EAR) file.

```
<env-entry-name>dataSourceJNDI</env-entry-name>
<env-entry-value>jdbc/db2emitter</env-entry-value>

<env-entry-name>emitterFactoryJNDI</env-entry-name>
<env-entry-value>iiop://localhost:2809/com/ibm/events/configuration/
emitter/Default</env-entry-value>

<env-entry-name>schedulerJNDI</env-entry-name>
<env-entry-value>sched/DB2Poller</env-entry-value>
```

Notes:

- The default emitter factory specified in the "emitterFactoryJNDI" environment entry refers to "localhost" because the emitter is being deployed to the same application server as the Monitor Server. If the emitter is deployed to its own application server, ensure that the hostname is that of the Monitor Server (where the CEI Server resides and to where the events will be emitted).
- The default emitter factory specified in the "emitterFactoryJNDI" environment entry uses port 2809. Check the Monitor Server's BOOTSTRAP_ADDRESS setting to ensure that is the correct port to use.

If any of these environment entries need to be modified, import the source projects into WID and make the necessary modifications to the EJB deployment descriptor. (See Section 5: Importing and working with the source code projects, for instructions on how to import the source projects.)

3.1. Create an IBM Cloudscape™ database for the Monitor Server Scheduler:

1. *In the ${WBI_INSTALL_ROOT}/cloudscape/bin/embedded directory, open the cview.bat file.*
2. *Select* ***File→ New→ Database****.*
3. *In the Name field, type* ***${WBI_INSTALL_ROOT}/cloudscape/databases/SKDLR****.*
4. *Click* ***OK*** *to create the new database and then exit from the cview.bat file.*

3.2. Create a data source for the Monitor Server Scheduler:

1. *Open the Administrative Console.*

2. *Open the* ***Resources -> JDBC Providers*** *page, and set the scope to* ***Server****.*

3. *By default, there should be a Java Database Connectivity (JDBC) provider named* ***Cloudscape JDBC Provider (XA)****. Click it then select* ***Data sources*** *under the Additional Properties section.*

4. *Click* ***New*** *to create a new data source with the following properties:*

```
Name: SKDLR Datasource
Java Naming and Directory Interface (JNDI) name: jdbc/skdlr
Disable "Use this Data Source in container managed persistence
(CMP)"
Description: JDBC Datasource for SKDLR Database
Database name: ${WBI _ INSTALL _ ROOT}/cloudscape/databases/SKDLR
```

5. *Click* ***OK*** *and* ***Save****.*

3.3. Create a scheduler for the DB2 Emitter:

1. *Open the Administrative Console.*

2. *Go to* ***Resources→ Schedulers*** *and set the scope to* ***Server****.*

3. *Click* ***New*** *to create a new scheduler with the following properties:*

```
Name: DB2Poller
JNDI name: sched/DB2Poller (This is the default name specified as
an environment entry in the EJB deployment descriptor.)
Description: Scheduler for DB2 Sample Emitter's Event Table Poller
Data source JNDI name: jdbc/skdlr
Table prefix: DB2EMTR _
Poll interval: 30
Work managers: DefaultWorkManager
```

4. *Click* ***OK*** *and* ***Save****.*

5. *On the Schedulers page, select the newly created* ***DB2Poller scheduler*** *and click* ***Create tables****.*

3.4. Open the DB2 Emitter enterprise application using the Administrative Console:

Accept all default values and save the configuration when deployment is completed.

3.5. Create an authentication alias for the DB2 database:

1. *Open the Administrative Console.*

2. *Select* ***Security→ Global security→ JAAS Configuration→ J2C Authentication data****.*

3. *Click* **New** *to create a new authentication alias with the following properties:*

```
Alias: DB2EmitterAlias
User ID: (User ID to connect to the database; for example,
db2admin)
Password: (Password to connect to the database)
Description: Authentication Alias for Connecting to DB2 Sample
Emitter's Database
```

3.6. Create the DB2 database and the event and log tables to be used for event management:

1. *Issue the DB2 command create database* `<DATABASE _ NAME>` *from a DB2 command-line processor (CLP).*
   In the DB2 CLP, connect to the newly created database using the same credentials used in the previous step to create the authentication alias.

2. *Invoke the* ***createEventAndLogTable.ddl*** *script located under the "setup" directory. This creates the event table, log table, and necessary procedures used for event management.*

3.7. Create a data source for the DB2 database:

1. *On the Resources→JDBC Providers page, set the scope to Server.*

2. *Click* **New** *to create a new JDBC provider with the following properties:*

```
Database type: DB2
Provider type: "DB2 Universal JDBC Driver Provider"
Implementation type: "XA data source"
```

3. *Click* **Next** *and then* **OK***.*

4. *On the JDBC providers page, click the newly created* ***DB2 Universal JDBC Driver Provider (XA)****, then click* **Data sources** *under the "Additional Properties" section.*

5. *Click* **New** *to create a new data source with the following properties:*

```
Name: DB2 Emitter Datasource
JNDI name: jdbc/db2emitter (This is the default name specified as
an environment entry in the EJB deployment descriptor.)
Disable "Use this Data Source in container managed persistence
(CMP)"
Description: JDBC Datasource for Sample DB2 Event Emitter
Component-managed authentication alias: {NodeName}/
DB2EmitterAlias (the name of the authentication alias previously
created in Step 3.5)
Database name: <DATABASE _ NAME> (the name of the database previ-
ously created in Step 3.6.1)
```

6. *Click* **OK** *and* **Save***.*

7. *On the JDBC providers page, select the newly created* **JDBC provider** *and click* **Test connection***. If the connection test fails, it is likely that the DB2UNIVERSAL_JDBC_DRIVER_PATH environment variable needs to be set.*
   To set it, go to **Environment→ WebSphere Variables***, click* **DB2UNIVERSAL_JDBC_DRIVER_PATH** *and set its value to the location where the DB2 Java libraries are located (for example, C:\Program Files\IBM\SQLLIB\java).*

3.8. From a DB2 CLP, connect to the database previously created in Step 3.6.1, using the same credentials as Step 3.5.

3.9. Run each of the create*Table.ddl scripts found in the “setup” directory to create the sample application tables to be monitored.**
Which application tables to create and experiment with is up to the user. Make sure to also run, from a DB2 CLP, the associated create***Trigger.ddl script for the tables that are created.

3.10. Restart the Monitor Server.

4. Verifying the configuration and deployment of the Emitter application:

1. *Ensure that the Monitor Server is started.*

2. *Open a DB2 CLP and connect to the database created in Step 3.6.1, where the sample application table or tables were created. (Be sure to use the same credentials that were specified when creating the J2EE Connector architecture [J2C] Authentication Alias in Step 3.5).*

   For example: db2 connect to `<DATABASE _ NAME>` *user db2admin using* `<DB _ PWD>`.

3. *Issue an insert statement on the application table (or tables) of interest to create a new record in the table (or tables).*

4. *Open a command prompt and change the directory to* `<MONITOR _ PROFILE _ DIR>/bin` *(for example, C:\IBM\WebSphere\ProcServer\profiles\wbmonitor\bin).*

5. *Run the script* ***eventquery.jacl***.

   For example, wsadmin -f ..\event\bin\eventquery.jacl -group "All events"
   All the CBEs categorized under that group are displayed. Among them should be one or more representing the creation of a record in the application table (or tables).

6. *Back in the DB2 CLP, connect to the database and issue a select query against the LOGTABLE.*

   A record should have been recorded for each event detected in the application tables.

   An application table event that was successfully emitted as a CBE will have RESULT='S'.

   An application table event that failed to be emitted as a CBE will have RESULT='F'.

5. Importing and working with the source code projects

The source code for the Sample DB2 Event Emitter is available in the source archive package in the form of WID projects. The following projects need to be imported into the workspace: EmitterFW, CEIEmitter, DBEmitter, DBEmitterEJB, and DBEmitterImpl.

To import a project perform the following steps:

1. *Open* ***WebSphere Integration Developer****.*
2. *From the menu bar, select* ***File→ Import****.*
3. *In the Import dialog box, select* ***Existing Project into Workspace…*** *then click* ***Next****.*
4. *Select the project to be imported into the workspace then click* ***Finish****.*

Perform these four steps for each of the five projects listed previously.

You can expect build errors to be raised until the classpath reference to the events-client.jar library in the CEIEmitter project is corrected. The events-client.jar library includes classes used for creating CBEs and emitting them.

Perform these additional steps:

5. *Right-click the* ***CEIEmitter*** *project and select* ***Properties****.*
6. *Go to* ***Java Build Path→ Libraries****.*
7. *Select the* ***events-client.jar*** *entry then click* ***Edit…***
8. *Browse to the folder in your file system where the events-client.jar file is located (for example, <WPS_INSTALL_DIR>/CEI/client).*

5.1. Projects overview

The DB2 Sample Event Emitter source code is split out across five logical projects as follows:

- *EmitterFW*
  The common emitter framework that is used by all the sample event emitters made available.
- *CEIEmitter*
  The CEI emitter code. This is also common code share by all the sample event emitters
- *DBEmitter*
  The enterprise application package that is deployed to the server.
- *DBEmitterEJB: DB2 Emitter specific EJB code.*
- *DBEmitterImpl*
  DB2 Emitter specific code. This project contains implementation code for retrieving records from the database and formatting events into CBEs. Additional formatters can be implemented and added here to extend the number of application data types that the emitter can process. (See the next section, "Design details – creating a new event formatter to handle additional application data types," for more information on formatters.)

6. Design details—creating a new event formatter to handle additional application data types

The DB2 Sample Event Emitter is designed so that it can be extended to handle processing of additional application data types by: creating new classes that implement the EventFormatter interface (in the EmitterFW project); extending the AbstractEventRetriever abstract class (in the DBEmitterEJB project); and “registering” the event formatter associated with an application data type by updating the DB2EventFormatter.properties (in the DBEmitterImpl project).

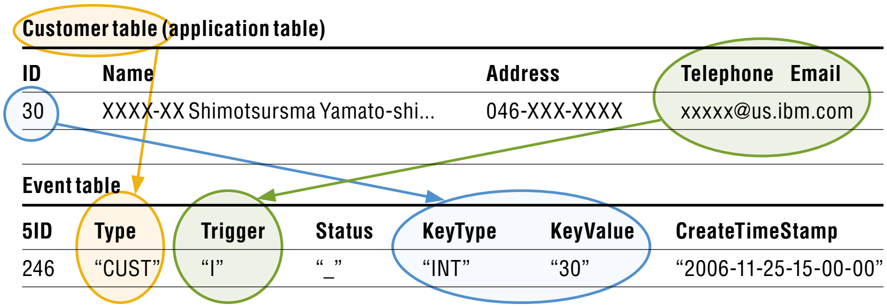
Before describing the new classes that need to be created, it should be mentioned that the application table first needs to be configured with the appropriate triggers that will create a record in the events table when an event of interest has occurred in the application table (or tables).

The record in the events table will contain the summary information regarding the event that occurred in the application table. It also stores the primary key to the application table record, which can be used to retrieve the full information on the event. Table 1 lists the schema for the event table used by this sample. Figure 3 depicts the source of the information stored in the events table records.

Table 1
Event table schema

SID	BIGINT	The primary key
TYPE	VARCHAR	The application data type that occurred in the application table
TRIGGER	CHAR	The kind of trigger (Create, Update or Delete)
KEYTYPE	VARCHAR	The type of key, which the EventFormatter knows
KEYVALUE	VARCHAR	The value of key, which the EventFormatter knows
CREATETIMESTAMP	TIMESTAMP	Time when an event posted

Figure 3
Events-table information source



Customer table (application table)

ID	Name	Address	Telephone	Email
30	XXXX-XX Shimotsursma Yamato-shi...	046-XXX-XXXX		xxxxx@us.ibm.com

Event table

5ID	Type	Trigger	Status	KeyType	KeyValue	CreateTimeStamp
246	“CUST”	“I”	“_”	“INT”	“30”	“2006-11-25-15-00-00”

Listing 1 shows the triggers created and used for the Customer data type.

Listing 1
Customer table triggers

```
CREATE TRIGGER CUSTOMERCREATE AFTER INSERT ON CUSTOMER REFERENCING
NEW AS NEWROW FOR EACH ROW MODE DB2SQL BEGIN ATOMIC INSERT INTO
EVENTTABLE(TYPE, TRIGGER, KEYTYPE, KEYVALUE, CREATETIMESTAMP)
VALUES(‘CUSTOMER’, ‘C’, ‘INT’, RTRIM(CHAR(NEWROW.ID)), CURRENT
TIMESTAMP); END;

CREATE TRIGGER CUSTOMERUPDATE AFTER UPDATE ON CUSTOMER REFERENCING
OLD AS OLDROW FOR EACH ROW MODE DB2SQL BEGIN ATOMIC INSERT INTO
EVENTTABLE(TYPE, TRIGGER, KEYTYPE, KEYVALUE, CREATETIMESTAMP)
VALUES(‘CUSTOMER’, ‘U’, ‘INT’, RTRIM(CHAR(OLDROW.ID)), CURRENT
TIMESTAMP); END;

CREATE TRIGGER CUSTOMERDELETE AFTER DELETE ON CUSTOMER REFERENCING
OLD AS OLDROW FOR EACH ROW MODE DB2SQL BEGIN ATOMIC INSERT INTO
EVENTTABLE(TYPE, TRIGGER, KEYTYPE, KEYVALUE, CREATETIMESTAMP)
VALUES(‘CUSTOMER’, ‘D’, ‘INT’, RTRIM(CHAR(OLDROW.ID)), CURRENT
TIMESTAMP); END;
```

The events table is periodically polled by the EmissionController, using the EventTablePoller, for the presence of new event records. When new records are present in the events table the DB2 emitter will obtain the appropriate formatter and retriever based on the data type recorded and based on the mapping specified in the DB2EventFormatter.properties file.

6.1. DB2EmitterFormatter.properties file

The DB2EmitterFormatter.properties file is a typical properties resource file containing entries in the form of “name equals value.” Listing 2 shows the one used by this sample.

Listing 2
DB2EventFormatter.properties mapping file

```
#
# This properties file defines the formatter to use for a given data
type.
# The accepted format is:
#    EventType=EventFormatter
# Where the EventFormatter is provided as the fully qualified Java
class.
#
CUSTOMER=com.ibm.wbimonitor.samples.db2emitter.formatter.
CUSTOMERFormatterImpl
ORDER=com.ibm.wbimonitor.samples.db2emitter.formatter.
ORDERFormatterImpl
CLAIM=com.ibm.wbimonitor.samples.db2emitter.formatter.
CLAIMFormatterImpl
```

To introduce support for a new application data type, a new entry mapping the event name to the fully qualified event formatter implementation class needs to be added to this file.

6.2. EventFormatter implementation

After an EventFormatter is obtained (based on the mapping in the DB2EmitterFormatter.properties file), it will be invoked to handle the transformation of the event from its "native format" to the CBE format that is expected by the Monitor Server.

A new EventFormatter will need to implement the EventFormatter interface defined in the com.ibm.wbimonitor.samples.emitterframework package in the EmitterFW project.

```
public CommonBaseEvent format( Object o ) throws EventFormatFailed
Exception;
```

Inspect the EventFormatter classes implemented by this sample emitter for an idea on how this format( ) interface can be implemented. They can be found in the com.ibm.wbimonitor.samples.db2emitter.formatter package in the DB2EmitterImpl project.

6.3. EventRetriever implementation

The EventFormatter uses an EventRetriever specific to the application data type being handled to retrieve the full data record from the application table (using the summarized information in the event table record). A new application data type retriever should extend the AbstractEventRetriever class and implement the retrieve( ) abstract method:

```
public abstract Object retrieve( Object o ) throws EventRetrieveFa
iledException;
```

Inspect the EventRetriever classes implemented by this sample emitter for an idea on how this retrieve( ) interface can be implemented. They can be found in the com.ibm.wbimonitor.samples.db2emitter.retriever package in the DB2EmitterImpl project.

7. Working with CBEs and CEI – key concepts

This section outlines the key concepts related to CEI and CBEs used in the Sample File Event Emitter.

7.1 Creating a new or obtaining an existing EventFactory:

1. *Creating a new one (with and without a ContentHandler):*

```
EventFactory eventFactory =
(EventFactory) EventFactoryFactory.createEventFactory();

EventFactory eventFactory =
(EventFactory) EventFactoryFactory.createEventFactory
(ContentHandler);
```

2. *Obtaining an existing one through JNDI (inherits ContentHandler, if one exists):*

```
Context context = new InitialContext();
EventFactory eventFactory = (EventFactory) context.lookup("com/ibm/
events/EventFactory");
```

7.2 Creating the new CommonBaseEvent:

```
CommonBaseEvent event = eventFactory.createCommonBaseEvent("Activi
tyEvent");
```

7.3 Setting mandatory fields:

```
event.setVersion( "1.0.1" );
event.setCreationTimeAsLong( System.currentTimeMillis( ) );
event.setGlobalInstanceId( eventFactory.createGlobalInstanceId( ) );

ComponentIdentification componentId = eventFactory.createComponen-
tIdentification( );
componentId.setApplication( "DB2" ); // Additional setters available
event.setSourceComponentId( componentId );

Situation situation = eventFactory.createSituation( );
situation.setStopSituation( "EXTERNAL", "STOP _ COMPLETED",
"SUCCESSFUL" );
event.setSituation( situation );
```

7.4 Creating an ExtendedDataElement and its children:

```
ExtendedDataElement activityEventData =
event.addExtendedDataElementWithNoValue( "ActivityEventData" );
activityEventData.addChild( "activityName", activityName );
activityEventData.addChild( "eventType", "completed" );
activityEventData.addChild( "activityDisplayState", "Completed" );
activityEventData.addChildWithDateAsLongValue( "startTime", efb.
getLastModifiedDate( ) );
activityEventData.addChildWithDateAsLongValue( "endTime", efb.get-
LastModifiedDate( ) );
```

7.5 Obtaining the EmitterFactory:

```
import javax.naming.*
import com.ibm.events.*
Context context = new InitialContext();
EmitterFactory emitterFactory =
   (EmitterFactory) context.lookup("com/ibm/events/configuration/
emitter/Default");
```

7.6 Obtaining the Emitter from the EmitterFactory:

```
Emitter emitter = emitterFactory.getEmitter();
```

7.7 Sending an Event:

```
emitter.sendEvent((CommonsBaseEvent)event);
```

IBM Corporation
Software Group
Route 100
Somers, NY 10589
U.S.A.

Produced in the United States of America
12-06