HP Business Service Management

for the Windows operating system

Software Version: 9.00

TransactionVision Advanced Customization Guide

Document Release Date: July 2010

Software Release Date: July 2010

Legal Notices

Warranty

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

The information contained herein is subject to change without notice.

Restricted Rights Legend

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

Copyright Notices

Trademark Notices

TransactionVision® is a registered trademark of the Hewlett-Packard Company.

Adobe® and Acrobat® are trademarks of Adobe Systems Incorporated.
AMD and the AMD Arrow symbol are trademarks of Advanced Micro Devices, Inc.
Google™ and Google Maps™ are trademarks of Google Inc.
Intel®, Itanium®, Pentium®, and Intel® Xeon® are trademarks of Intel Corporation in the U.S. and other countries.
Java™ is a US trademark of Sun Microsystems, Inc.
Microsoft®, Windows®, Windows NT®, Windows® XP, and Windows Vista® are U.S. registered trademarks of Microsoft Corporation.
Oracle is a registered trademark of Oracle Corporation and/or its affiliates.
UNIX® is a registered trademark of The Open Group.

Acknowledgements

This product includes software developed by the Apache Software Foundation (**http://www.apache.org**).

This product includes software developed by the JDOM Project (**http://www.jdom.org**).

This product includes software developed by the MX4J project (**http://mx4j.sourceforge.net**).

Contents

1. Welcome to This Guide

This guide describes how the TransactionVision platform can be extended and customized to achieve further control over its various functions. It presents an architecture overview of the TransactionVision system and documents the different methods available to use and extend the Analyzer and the query service.

This chapter contains the following sections:

1.1. Who Should Read This Guide

This guide is for the following users of TransactionVision:

- Application developers or configurators
- System or instance administrators
- Database administrators

Readers of this guide should be moderately knowledgeable about enterprise application development and highly skilled in enterprise system and database administration.

1.2. TransactionVision Documentation

TransactionVision documentation provides information on using the TransactionVision application of the BSM and deploying and administering the TransactionVision-specific components in the BSM deployment environment.

The TransactionVision documentation includes:

- The TransactionVision Deployment Guide describes the installation and configuration of the TransactionVision-specific components in the BSM deployment environment. This guide is available as a PDF in the BSM Documentation Library.
- The Using Transaction Management Guide describes how to set up and configure TransactionVision to track transactions and how to view and customize reports and topologies of business transactions. This guide is available as the TransactionVision Portal or as a PDF in the BSM Online Documentation Library.

- The TransactionVision Planning Guide contains important information for sizing and planning new installations. This guide is available by download from the HP Software Product Manuals site. See “Documentation Updates” on page 3.
- The TransactionVision Advanced Customization Guide contains information for how the TransactionVision platform can be extended and customized to achieve further control over its various functions. It presents an architecture overview of the TransactionVision system and documents the different methods available to use and extend the Analyzer, the query service and the TransactionVision user interface.

Additional TransactionVision documentation can be found in the following areas of the BSM:

Readme. Provides a list of version limitations and last-minute updates. From the HP BSM DVD root directory or download package root directory, double-click readme<version>.html. You can also access the most updated readme file from the HP Software Support Web site.

What’s New. Provides a list of new features and version highlights. In HP BSM, select Help > What’s New.

Online Documentation Library. The Documentation Library is an online help system that describes how to work with HP BSM and the TransactionVision application. You access the Documentation Library using a Web browser. For a list of viewing considerations, see “Viewing the HP BSM Site” in chapter 6 of the the HP BSM Deployment Guide PDF.

To access the Documentation Library, in HP BSM, select Help > Documentation Library. Context-sensitive help is available from specific HP BSM pages by clicking Help > Help on this page and from specific windows by clicking the Help button. For details on using the Documentation Library, see “Working with the HP BSM Documentation Library” in Platform Administration.

1.3. Additional Online Resources

Troubleshooting & Knowledge Base accesses the Troubleshooting page on the HP Software Support Web site where you can search the Self-solve knowledge base. Choose Help > Troubleshooting & Knowledge Base. The URL for this Web site is http://h20230.www2.hp.com/troubleshooting.jsp.

HP Software Support accesses the HP Software Support Web site. This site enables you to browse the Self-solve knowledge base. You can also post to and search user discussion forums, submit support requests, download patches and updated documentation, and more. Choose Help > HP Software Support. The URL for this Web site is www.hp.com/go/hpsoftwaresupport.

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

To find more information about access levels, go to:

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

To register for an HP Passport user ID, go to:

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

2. Architecture Overview

This chapter includes the following sections:

2.1. System Components

TransactionVision consists of the following logical components:

- The Sensor component generates events based on the technology being sensed. The agent gets configuration and filtering messages from the configuration queue and sends events into the event queue. The event and configuration queues are represented by the "messaging middleware" box in the diagram on the following page.
- The Analyzer component is responsible for retrieving and analyzing events from the communication link. It contains a chain of Java bean contexts, each performing a particular function on the event data. Each bean context can hold multiple chained beans to perform custom processing of the event data. The beans in each bean context are controlled by the **Beans.xml** file. The main components of the Analyzer include:

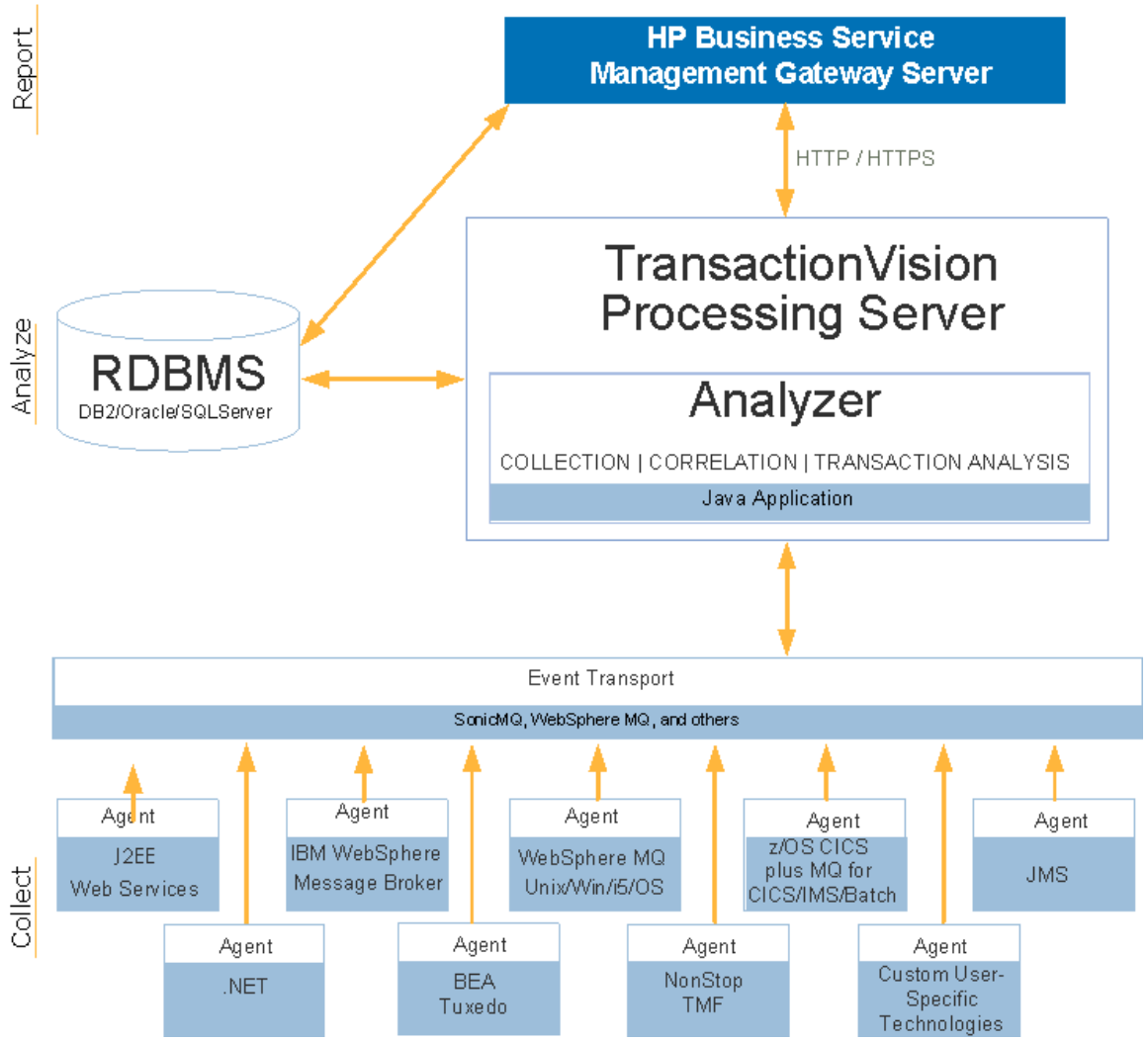
  Unmarshaller bean context. Converts raw event data from its binary format into XML. This bean context provides an environment for user message data unmarshaller beans to be plugged in.

  DBWriteExit bean context. Allows a custom bean to trim or cut down on the data written into the database. This gives a user flexibility to cut down on storage size. Typically this is an XSLT which processes the XML tree generated by the unmarshaller context.

  Database write context. Maps the XML tree generated by the unmarshaller and trim contexts to database tables and writing the tree into the database. This context uses the XML data mapper component to map the XML tree to relational database tables.

  Analysis context. Performs event correlation, local and business transaction analysis, transaction classification, statistics analysis and any other custom data analysis.
- The User Interface and database query services components provide a means of mapping XML data to relational tables, and a XML based query to an SQL statements based on relational tables.

The following diagram shows the TransactionVision architecture layout:



2.2. RDBMS

The general table organization consists of a TVISION schema, where project, communication links, filter, queries and other administration related information is stored, and project-specific schemas, where events collected by a project are stored. Each project schema consists of an event table, where the event identifier and the XML event are stored, and several lookup tables that provide indexes to the event. In addition there are several other tables in a project schema storing event correlation, local transaction, business transaction and other system infrastructure objects related information.

The Configuration and Administration component manages administration of the TransactionVision system. This includes starting/stopping data collection and event processing, changing data collection filters, providing system status, and administering security policies on the Analyzer service. The Analyzer is controlled using embedded RMI and can run on systems different from the application server.

This advanced customization guide provides the details of extending TransactionVision. It is important to note that since this documentation is related to the internals of the product, incorrect changes could break the functioning of the product.

3. Tutorial - Extending the Analyzer

The Analyzer reads in binary event packets from the TransactionVision Event Queue and processes them through a chain of bean contexts. Each bean context performs a specific function to analyze and write data from the event into the database. Many of these operations can be extended and customized to perform transformations based on your systems or application needs. This chain of beans is defined by the Beans.xml file. The sequence of bean contexts includes:

- The event modifier context, which allows users to write custom beans to modify the incoming event, such as convert binary message data into XML.
- The data writer context, which contains beans to write the data into various relational database tables.
- The analysis context, which contains various beans to perform event analysis, transaction analysis and correlation of events to create a business transaction.

Each context holds beans that perform a default function and can be replaced or added on to perform further actions on the data being processed. The following sections document common tasks related to extending the Analyzer:

3.1. How to Handle XML Message Data in Events

When your message data is already composed of XML, a custom bean is not required to have the XML processed by the Analyzer. Instead, TransactionVision provides a default modifier that is enabled by default and attaches the message data XML contents to the TransactionVision event.

This section describes the functionality of the TransactionVision DefaultModifierBean, which detects XML data in the message field and appends it to the XML event. The default event modifier bean, **com.bristol.tvision.services.analysis.eventmodifier.DefaultModifierBean**, scans the user data for any XML data and, if found,

simply adds it to the Event XML document at the position **/Event/UserData/Chunk[@seqNo='n']** where **n** is the number of the data range (defined in the data collection filter).

3.1.1. Verify that XML Data is Extracted Correctly

Collect events using the TransactionVision agents from your application. For each event that generates XML message data, go to the event detail view and verify that the XML data shows up in the user data panel.

Once this task is done, the XML message data can be mapped to custom database tables based on the kind of analysis that is required to be performed on the message data. Section 3.4 describes how to implement this mapping.

3.2. How to Handle Custom Message Data Formats in Events

Typically, event data from applications may contain binary, text or XML data embedded within the message. This data is often in custom and proprietary formats that are not known to the TransactionVision Analyzer. A common task is to convert these custom formats into XML within the Analyzer for later use in reports, for analysis, browsing or querying. The TransactionVision Analyzer allows for embedding a Java bean that implements the IEventModifier interface to perform the format conversion. This bean can modify the event being currently processed to do any kind of format conversion on the event data.

This section describes the steps to write a custom event modifier bean in Java to extract and convert binary message data and insert it into the event data as XML. A custom bean that converts a text message into XML will be used as an example. The sample code used below is from the CICS Accounting sample shipped with TransactionVision. The source code can be found at **<TVISION_HOME>/samples/CICSAccounting**.

3.2.1. Step 1: Document message format layout

The first step in the process of writing a bean to handle custom event data is to know the layout of all message formats in the event data and document them.

Consider the sample message to contain the following text layout, with fields Account Number, Last Name, First Name and Account Type:

AccountNumber (5 characters)	LastName	**S**	FirstName	**S**	AccountType (1 character-S/M/C)	. . .

In the above layout, the first 5 bytes are the AccountNumber field, while the remaining fields of **LastName**, **FirstName** and **AccountType** are separated by a space separator "S". The **LastName** and **FirstName** fields are variable length fields. The **AccountType** field is one character and can be either "S" (Savings), "M" (Money Market) or "C" (Checking). The remaining fields are ignored and not required to be processed by TransactionVision.

3.2.2. Step 2: Document the Target XML Format

First design the target XML document to be created from the above text message. The following is a sample resulting XML structure:

```
<Event sort="false">
...
    <Data>
        <Chunk blobType="2" ccsid="0" from="0" seqNo="0" to="382">
            <Account>
```

```
                <AccountNo>11111</AccountNo>
                <LastName>DOE</LastName>
                <FirstName>JOHN</FirstName>
                <AccountType>Saving</AccountType>
            </Account>
        </Chunk>
    </Data>
...
```

Here, an **Account** node is created under the item **/Event/Data/Chunk**. This is the point where TransactionVision stores references to message data. Hence, this is a good point, though not the only point, where any XML converted message data can be added. Under the **Account** node, nodes for **AccountNo**, **LastName**, **FirstName** and **AccountType** are created and their values filled in.

The XPath values of each of the above fields are as follows:

AccountNo: **/Event/Data/Chunk/Account/AccountNo**

LastName: **/Event/Data/Chunk/Account/LastName**

FirstName: **/Event/Data/Chunk/Account/ FirstName**

AccountType: **/Event/Data/Chunk/Account/AccountType**

3.2.3. Step 3: Implement the Bean to Do the Format Conversion

This section describes the implementation of the Java bean to perform the format conversion described in Steps 1 and 2. The source code for the Modifier Bean is available in **<TVISION_HOME>/samples/stock**.

A Java class, **CICSAccountingModifierBean**, extends from the base class EventModifierBean and implements the modify method of the IEventModifier interface. This modify method is invoked by the Analyzer framework to perform any custom data conversion tasks.

```
public class CICSAccountingModifierBean extends EventModifierBean
{
    ...

    /** Creates new CICSAccountingModifierBean */
    public CICSAccountingModifierBean() {
    ...
    }

public boolean modify(XMLEvent event, ConnectionInfo conInfo)
throws EventModifyException {
```

The above code defines the **CICSAccountingModifierBean** class with a method modify. This method accepts the current XML event object and database connection info as its input and is allowed to modify this object in any way, typically for transforming data from proprietary formats to XML. If the method returns **false**, the current event will be discarded and not processed any further. For modifying purposes, this method should always return **true**.

The following code fragment from the modify method first verifies whether this is the right CICS event whose message data needs to be processed, and then processes chunks of message data.

```
00055             String tech =
event.getDocumentValue(XPathConstants.TECH_NAME);
00056             if (tech == null || !tech.equalsIgnoreCase("CICS"))
```

```
00057                 return;
00058
00059             String eventType =
event.getDocumentValue(XPathConstants.CICS_COMMON_EVENTTYPE);
00060             if (eventType == null)
00061                 return;
00062
00063             int type = Integer.parseInt(eventType);
00064             if (type != CICSConstants.B_CICS_TYPE_FC)
00065                 return;
00066
```

In the above code, the constant **XPathConstants.TECH_NAME** contains the value to the technology XPath expression. The **XPathConstants** class contains various other commonly used XPath expression values. Hence, line 55 extracts the value of the technology name from the event document. Line 56 ignores all events that are not from the CICS Agent (ie. are not of technology CICS). Line 59-64 lookup the event type of the CICS event. Only file control APIs (CICSConstants.B_CICS_TYPE_FC) are considered for further processing. The **getDocumentValue** method returns the value of any XPath location in the DOM tree in the XMLEvent object.

The following code fragment shows how to obtain pieces of user data from the XMLEvent object.

```
00067             XPathSearch lookup = new XPathSearch(event);
00068
00069             /*
00070              * get user data chunks by retrieving all
/Event/Data/Chunk nodes
00071              */
00072             NodeList dataChunks =
lookup.getNodes(XPATH_DATACHUNK);
00073             if (dataChunks == null || dataChunks.getLength() == 0)
00074                 return;
00075
00076             int chunkNum = dataChunks.getLength();
00077
00078             /*
00079              * process each user data chunk
00080              */
00081             for (int i = 0; i < chunkNum; i++) {
00082                 try {
00083                     processUserData(event, lookup,
(Element)(dataChunks.item(i)));
00084                 }
00085                 catch (XMLException xmlEx) {
00086                     throw new EventModifyException(xmlEx);
00087                 }
00088             }
```

Line 67 creates an XPathSearch object, whose function is to perform lookups on the XMLEvent document. The **getNodes** and **getValues** methods on the **XPathSearch** class enable lookups based on given XPath expressions. Section 0 has the documentation on this class and its methods.

The message data in a TransactionVision event is stored as a series of chunks. This is done since the message data from TransactionVision Sensors can be broken up based on data ranges specified in the data collection filter. The XMLEvent document contains the location and byte count of each of these chunks and can be looked up using the XPath expression **/Event/Data/Chunk**. Typically, if no data range is specified in the data collection filters, only one chunk is created.

Line 72 gets a list of data chunk nodes. For each of the data chunks, the method **processUserData** is called to perform the format conversion.

The following code fragment is from the **processUserData** method which converts the text message into XML.

```
00104     private void processUserData(XMLEvent event, XPathSearch
lookup, Element owner) throws XMLException {
00105
00106         int chunkId =
Integer.parseInt(owner.getAttribute(ATTR_CHUNK_ID));
00107
00108         /*
00109          * find the XMLEvent.Blob which has the same chunk ID
00110          */
00111         XMLEvent.Blob chunkBlob = null;
00112
00113         Iterator it = event.blobIterator();
00114         while (it.hasNext() && (chunkBlob == null)) {
00115             XMLEvent.Blob blob = (XMLEvent.Blob)it.next();
00116             if (blob.id == chunkId)
00117                 chunkBlob = blob;
00118         }
00119
00120         if (chunkBlob == null)
00121             return;
00122
00123         int ccsid = chunkBlob.ccsid;
00124
00131         /*
00132          * if ccsid <=0 use ccsid in standard header
00133          */
00134         if (ccsid <= 0) {
00135             ccsid =
Integer.parseInt(lookup.getValue(XPATH_EVENT_CCSID));
00136         }
00137
00138         String strBuf =
Translator.instance(ccsid).translate(chunkBlob.blob);
00139
00142
00143         /*
00144          * parse XML document, and if succeeds, append it to owner
node,
00145          * then change the data type of the chunk.
00146          */
00147         Element acctRoot = event.createElement("Account");
00148         owner.appendChild(acctRoot);
00149
00150         StringElement acctNo = new StringElement("AccountNo");
00151         StringElement lastName = new StringElement("LastName");
00152         StringElement firstName = new StringElement("FirstName");
```

```
00153         StringElement acctType = new StringElement("AccountType");
00154
00155         StringTokenizer tokens = new StringTokenizer(strBuf);
00156         int cnt = tokens.countTokens();
00157         if (cnt < 3)
00158             return;
00159
00160         String[] tokenStrs = new String[cnt];
00161         for (int i = 0; i < cnt; i++) {
00162             tokenStrs[i] = tokens.nextToken();
00164         }
00165
00166         // check for numeric value
00167         char c = tokenStrs[0].charAt(0);
00168         if (!Character.isDigit(c))
00169             return;
00170
00171         acctNo.value = tokenStrs[0].substring(0, 5);
00172         lastName.value = tokenStrs[0].substring(5);
00173         firstName.value = tokenStrs[1];
00174
00175         int idx = strBuf.indexOf("FSR");
00176         acctType.value = strBuf.substring(idx + 3, idx + 4);
00177         if (acctType.value.equalsIgnoreCase("M"))
00178             acctType.value = "Money Market";
00179         else if (acctType.value.equalsIgnoreCase("S"))
00180             acctType.value = "Saving";
00181         else
00182             acctType.value = "Checking";
00183
00184         acctNo.toDOM(event, acctRoot);
00185         lastName.toDOM(event, acctRoot);
00186         firstName.toDOM(event,acctRoot);
00187         acctType.toDOM(event, acctRoot);
00188
00189         chunkBlob.type = TVisionCommon.XMLEVENT_BLOB_XML;
00190
00191         owner.setAttribute(ATTR_BLOBTYPE,
00192             Integer.toString(TVisionCommon.XMLEVENT_BLOB_XML)
00193         );
00194     }
00195 }
```

The method **processUserData** converts one chunk of message data text into XML. Line 98 obtains the id of the current chunk of message data being processed.

Lines 105-115 access the array of message data binary objects (BLOB) that are stored in a separate table in the same sequence as the chunks in the XML document. Typically, there is just one object, but there could be more depending on whether data ranges have been set in the data collection filter. Hence, a chunk in the XML document with the ID 1 will have its equivalent BLOB in the user data table at the sequence number 1. The chunk id from the XML document is matched with the message data BLOB ID.

Lines 120-126 find the codepage of the message data and convert it to the code page the Analyzer is using. This is required because the message data is from CICS and needs to be converted from EBCDIC to ASCII.

Lines 134-135 create new nodes in the XMLEvent document to hold the Account related data.

Lines 137-140 create new objects of type StringElement. This class is a TransactionVision utility class that has the ability to generate XML DOM nodes from input values. Refer to Section 4.2.4.4 for details on this class. The toDOM method of this class creates and appends XML DOM nodes to a DOM tree at a specified location.

Lines 142-168 is Java code which parses the message data string buffer and extracts values for Account, LastName, FirstName and AccountType based on the format defined in Step 2.

Lines 170-173 convert the parsed message data from their StringElement values into DOM nodes attached to the XMLEvent DOM tree at location **/Event/Data/Chunk/Account**.

3.2.4. Step 4: Modify the Beans.xml File to use the Custom Bean

The event modifier bean implemented in the previous steps needs to be enabled in the event modifier context of the **Beans.xml** file. Change the **Beans.xml** file to add the following line:

```
<Module type="Context" name="EventModifierCtx">
<Module type="Bean" class=
"com.bristol.tvision.samples.accounting.CICSAccountingModifierBean"/
>
  </Module>
```

The Analyzer needs to be re-started after this change.

3.2.5. Step 5: Test the Custom Bean in the Analyzer Environment

To verify that the above data extraction is working correctly, check the right events user data buffer in the event detail view. In the example above, check the user data for the file control READ API.

3.3. Overview of XDM Files

Certain pieces of information in the message data may be useful to be queried upon by custom reports or analysis modules. In that case, these fields need to be extracted from the message data and mapped to database columns by the Analyzer. Before these fields can be written to a database column by the Analyzer, they need to be extracted from the message and converted to XML (if not already in the XML format). Section 3.2 describes how to extract binary message data and convert it to XML and Section 3.1 describes how to handle XML message data.

The TransactionVision database schema is made extensible through the XML to Database Mapping (XDM) files. As message data specific columns are added to the database, the XDM files can be updated to describe the new schema. Hence XML to Database mapping serves several purposes:

- To describe to the CreateSqlScript program the layout of the project database schema tables.
- To describe to the Analyzer the fields that are to be extracted from the XML event data and stored in event lookup tables for fast searching and retrieval.
- To describe to the Analyzer the fields that are to be extracted from the transaction XML document and stored in the transaction lookup tables.
- To describe the database schema to the query services for use in TransactionVision user interface views and reports.

3.4. How to Map Custom Message Data Fields to Database Tables

The task in this section describes how to map event XML data to database fields using TransactionVision's XDM (XML to Database Mapping) module.

3.4.1. Step 1: Determine which fields in the XML event document need to be mapped to database columns

Consider a WebSphere MQ MQPUT request event which has the following XML segment in its message data:

```
<Event>
   <Data>
         <Order>
           <ID>123456</ID>
           <Branch>Danbury</Branch>
           <Account></Account>
           <Ticker>MSFT</Ticker>
           <Price>88.88</Price>
           <Shares>1000</Shares>
         </Order>
   </Data>
</Event>
```

Consider a WebSphere MQ MQPUT reply event in response to the above request that contains the following XML segment in its message data:

```
<Event>
   <Data>
         <Result>
           <ID>123456</ID>
           <Type>Stock</Type>
           <Status>Success</Status>
         </Result>
   </Data>
</Event>
```

3.4.2. Step 2: Determine the Database Column Names for these Fields

The mapping of message data to database columns enables custom business reports and queries to be written to view and analyze the contents of the message data.

Consider that the following fields need to be mapped to database columns from the message data described in Step (1).

For the MQPUT request message data, a TRADE_ORDER table can be defined as follows:

Field Name	SQL Type	Length
ORDERID	VARCHAR	16
BRANCH	VARCHAR	16
ACCOUNT	VARCHAR	8
TICKER	VARCHAR	8
PRICE	VARCHAR	8
SHARES	VARCHAR	8
PROGINST_ID	INTEGER	4
SEQUENCE_NO	INTEGER	4

For the MQPUT reply message data, a TRADE_RESULT table can be defined as follows:

Field Name	SQL Type	Length
ORDERID	VARCHAR	16
TYPE	VARCHAR	8
STATUS	VARCHAR	12
PROGINST_ID	INTEGER	4
SEQUENCE_NO	INTEGER	4

In both the above tables, PROGINST_ID and SEQUENCE_NO are event identification fields that are required to join with the TransactionVision EVENT table, while the remaining columns contain business content to be extracted from the message data.

3.4.3. Step 3: Construct XDM File Entries

Now that we have determined the format and contents of the message data in Step 1 and which database tables need to be populated in Step 2, a mapping can be created from the XML message data contents to the database columns.

Consider the following XML segment::

```
<Event>
   <Data>
         <Order>
           <ID>123456</ID>
                ...
         </Order>
   </Data>
</Event>
```

The XPath to the Order ID field can be written as: **/Event/Data/Order/ID**.

The value at this XPath needs to be written to the ORDERID column of the TRADE_ORDER table.

This mapping can be done in an XDM file as follows:

```
<Table name="TRADE_ORDER" categoryPath=" /Event/StdHeader/TechName"
categoryValues="MQSERIES,JMS">
<Column name="ORDERID" type="VARCHAR" size="16"
description="OrderID">
         <Path>/Event/Data/Order/ID</Path>
  </Column>
  ...
```

The above XDM file segment defines a table name TRADE_ORDER in the **Table** element. The table contains a column ORDERID, defined by the **Column** element, of type VARCHAR and size 16 bytes. The **Column** of name ORDERID has an XPath mapping, defined by the **Path** element to be **/Event/Data/Order/ID**.

The table definition part of the XDM segment is applied when a new project schema is created either by CreateSqlScript or the project creation web pages. The XPath mapping part of the XDM segment is applied by the Analyzer when processing events. When an event contains data at the XPath value **/Event/Data/Order/ID**, the Analyzer extracts the value and writes a row to the mapped column ORDERID belonging to table TRADE_ODER for that event. The **categoryPath and categoryValues** attributes for the **Table** element, indicates that this mapping is applied only to MQSeries and JMS events.

The complete mapping of the MQPUT request message to the TRADE_ORDER table is as follows:

```
<Mapping documentType="/Event">
  <Key name="proginst_id" type="BIGINT"
description="ProgramInstanceId">
        <Path>/Event/EventID/@programInstID</Path>
  </Key>
  <Key name="sequence_no" type="INTEGER"
description="SequenceNumber">
        <Path>/Event/EventID/@sequenceNum</Path>
  </Key>

  <Table name="TRADE_ORDER" categoryPath="
/Event/StdHeader/TechName" categoryValues="MQSERIES,JMS">
        <Column name="orderid" type="VARCHAR" size="16"
description="OrderID">
              <Path>/Event/Data/Order/ID</Path>
        </Column>
        <Column name="branch" type="VARCHAR" size="16"
description="Branch">
              <Path>/Event/Data/Order/Branch</Path>
        </Column>
        <Column name="account" type="VARCHAR" size="8"
description="AccountNumber">
              <Path>/Event/Data/Order/Account</Path>
        </Column>
              <Column name="ticker" type="VARCHAR" size="8"
description="Ticker">
                    <Path>/Event/Data/Order/Ticker</Path>
        </Column>
              <Column name="price" type="VARCHAR" size="8"
description="Price">
                    <Path>/Event/Data/Order/Price</Path>
        </Column>
              <Column name="shares" type="VARCHAR" size="8"
description="NumberOfShares">
                    <Path>/Event/Data/Order/Shares</Path>
        /Column>
  </Table>

</Mapping>
```

The file **Stock.xdm** is available in **<TVISION_HOME>/samples/stock**.

3.4.4. Step 4: Recreate your Project Database Schema

Copy the new XDM file to **<TVISION_HOME>/config/xdm**.The TransactionVision Analyzer and UI/Job Server components need to be restarted for the modified XDM files to have effect. Once these components are restarted, when new project schemas are created, they will contain the newly defined tables or columns. However, existing database project schemas need to be updated to create the newly added tables or columns. This can be done using options in the **CreateSqlScript** utility.

For example:

```
CreateSqlScript –c –e –n –p TEST –t TRADE_ORDER
```

The above command creates the table TRADE_ORDER as defined in the XDM file in the TEST database schema.

3.4.5. Step 5: Verify that the XDM Mapping works correctly

Start Analyzer collection for the project that has the custom XDM mapping. Generate events containing the message data with the expected XPath entries. Verify that rows are written into the TRADE_ORDER table for every event containing the expected message data.

3.5. Additional XDM File Examples

The XDM mappings can be defined for specific events (technology, platform, etc.) by using the attributes **categoryPath** and **categoryValues**. The common mapping defined in the file **<TVISION_HOME>/config/xdm/Event.xdm** (data in the standard event header) uses categoryPath="COMMON" and will be written for every event. The mappings defined in the other XDM files will only be applied if the value of **categoryPath** in the current event matches one of the values listed in **categoryValues**. The XML schema format of XDM files is defined in **<TVISION_HOME>/config/xmlschema/XDM.xsd**. The following code is an extract from the file **Event.xdm**.

```
<?xml version="1.0"?>
<Mapping documentType="/Event">
        <Key name="proginst_id" type="INTEGER"
description="ProgramInstanceId">
                <Path>/Event/EventID/@programInstID</Path>
        </Key>
        <Key name="sequence_no" type="INTEGER"
description="SequenceNumber">
                <Path>/Event/EventID/@sequenceNum</Path>
        </Key>
        <Table name="EVENT_LOOKUP" categoryPath="COMMON">
            <Column name="host_id" type="INTEGER" description="Host"
isObject="true">
                <Path>/Event/StdHeader/Host/@objectId</Path>
            </Column>
            <Column name="program_id" type="INTEGER"
description="Program" isObject="true">
    <Path>/Event/StdHeader/ProgramName/@objectId</Path>
            </Column>
         ...
        </Table>
</Mapping>
```

The above snippet from **Event.xdm** defines a table EVENT containing the XML document and a table EVENT_LOOKUP, containing various indexed columns of data from the XML document. The key columns proginst_id and sequence_no are integer types and mapped to XPath expressions /Event/EventID/@programInstID and /Event/EventID/@sequenceNum. These key columns are primary keys common to the EVENT and EVENT_LOOKUP tables. Similarly, the columns host_id and program_id are mapped to XPath expressions /Event/StdHeader/Host/@objectId and /Event/StdHeader/ProgramName/@objectId respectively.

The preceding XDM file specifies that when an XML event is written to the database by the DBWrite module in the Analyzer, these fields are extracted and written into the database columns mapped to in the XDM file. Similarly, when the database is queried using the QueryService XML interface, these XDM files are used to construct the corresponding SQL query.

The isObject attribute for a Column tag in the XDM file refers to that column being an identifier for an object in the system model table. The documentType attribute defines the type of the mapping (event, transaction, statistics, etc.). The key is the primary key and is common to the document table and the lookup tables. Each lookup column is indexed.

The conversionType attribute for a Column tag means that field requires a formatting conversion before writing to the database. The TypeConvService is called into before writing that field into the database. This is typically used for writing date/time or enumeration fields.

```
<Column name="primary_time" type="CHAR" size="20"
description="PrimaryTime " conversionType="Date">
<Path>/Event/StdHeader/EntryTime</Path>
</Column>
```

The categoryPath attribute on the Table tag contains either COMMON or an XPath expression into the event document. The string COMMON indicates that this table contains data common to every event and should be written for every event going through the Analyzer. If the categoryPath contains an XPath expression, the mapping only applies to events which have a value at this path that matches any of the values specified in 'categoryValues' If the categoryPath is empty, the mapping will be applied to all events.

Example: a table mapping with categoryPath='/Event/Technology' categoryValues'="MQSERIES,SERVLET" will only insert rows for MQSeries and Servlet events.

```
<Table name="EVENT_LOOKUP" categoryPath="COMMON">
...
</Table>
<Table name="OS390_LOOKUP" categoryPath=" /Event/StdHeader/TechName"
categoryValues="MQSERIES,SERVLET">
...
</Table>
```

A column can map to multiple XPath expressions, as in the following sample code. This assumes that only one of the XPaths will exist in a given event document.

```
<Column name="msgid" type="CHAR" size="72" description="MessageID">

<Path>/Event/Technology/MQSeries/MQPUT/MQPUTExit/MQMD/MsgId</Path>

<Path>/Event/Technology/MQSeries/MQPUT1/MQPUT1Exit/MQMD/MsgId</Path>

<Path>/Event/Technology/MQSeries/MQGET/MQGETExit/MQMD/MsgId</Path>
</Column>
```

Additionally, business transaction attributes (as opposed to event attributes) can also be mapped to transaction based XDM files. Refer to Section 9.2 for details on the XDM file layout.

3.6. How to Classify Business Transactions and Map Attributes to Database Tables

3.6.1. Overview of Transaction Classification

Transaction classification allows users to partition their business transactions into different transaction classes and set transaction attributes based on event data. These classes may be

created based on data in the messages flowing through the business system. A transaction is classified to a transaction class when attributes in one or more events in the transaction match the criteria defined in for the transaction tracing rule assigned to the transaction on the Transaction Tracing Configuration page. This page also allows setting of attributes on transactions. These attribute values can be extracted from one or more events belonging to that transaction. These attributes then can be mapped to database tables using XDM files.

Consider a business system consisting of a JSP/servlet based user interface, a middle-tier based on EJBs and a mainframe based backend. The following sample classification criteria may be applied to such a system:

- Based on the types of business systems these transactions involve. For example, if the 3-tier system described above supports financial transactions such those dealing with stocks and bonds, transaction classes may be created based on this.
- Based on statistics that need to be collected for each class. Such statistics may include service level and response time requirements for different classes of transactions. In the 3-tier system described earlier, aggregate response times could be measured for each tier of the system.

The Transaction Tracking Report lists transaction classes and attributes automatically along with common attributes such as start time, response time etc. For more information about this report, see the *Using Transaction Management*.

3.6.2. Task Description

The task in this section describes the following:

- How to extract event data and map that data to transaction attributes.
- How to map transaction attributes to database tables using transaction XDM files.
- How to use the Transaction Definition Editor to perform transaction classification

The sample message data used in this section is from the TRADE demo system, for which the project and event databases are shipped with TransactionVision. Refer to the *Using Transaction Management* for information on how to set up the TRADE demo database.

The previous sections in this chapter have discussed mapping event attributes to database tables. This section describes how to map business transaction attributes to database tables. This involves extracting attributes from events that apply to the business transaction the event belongs to and writing them to business transaction XDM tables.

3.6.3. Implementation

Step 1: Determine the event attributes that apply to a business transaction

Consider a request event which has the following XML segment in its message data:

```
<Event>
   <Data>
         <Order>
           <Account>123456</Account>
           <Transaction>Danbury</Transaction>
           <Type></Type>
           <Product>MSFT</Product>
           <Quantity>88.88</Quantity>
           <!-- present in FX transactions -->
           <Currency>1000</Currency>
           <RecvAccount>1000</RecvAccount>
           <!-- present in Bond transactions -->
           <Maturity>1000</Maturity>
           <Issue>1000</Issue>
           <!-- present in Equity transactions -->
           <Symbol>1000</Symbol>
         </Order>
   </Data>
</Event>
```

Three kinds of transactions flow through this TRADE system: Bond, Equity and FX (foreign exchange). Besides a common header, each transaction type has data specific to that transaction.

Consider the reply event in response to the above request that contains the following XML segment in its message data:

```
<Event>
   <Data>
         <Order>
           <ID>123456</ID>
           <Region>Stock</UnitPrice>
           <Status>Success</Status>
           <Reason>Success</Reason>
           <!-- present in Bond transactions -->
           <Yield>5.94</Yield>
         </Order>
   </Data>
</Event>
```

Step 2: Determine Database Column Names for Fields

The mapping of message data to transaction database columns enables custom business reports and queries to be written to view and analyze the contents of the business transaction. Consider that the following fields need to be mapped to database columns from the message data described in Step 1.

The TRADE_BUSINESS_TRANSACTION table is defined as below:

Field Name	SQL Type	Length
ORDERID	VARCHAR	20
REGION	VARCHAR	12
ACCOUNT	VARCHAR	12

TRADETYPE	VARCHAR	12
TRADEACTION	VARCHAR	12
AMOUNT	DOUBLE	8
STATUS	VARCHAR	12
REASON	VARCHAR	32
BONDISSUE	VARCHAR	12
BONDMATURITY	INTEGER	4
EQUITYSYMBOL	VARCHAR	8
VALUE	DOUBLE	8
CUSTOMER	VARCHAR	32
BUSINESS_TRANS_ID	INTEGER	4

In the above table, the BUSINESS_TRANS_ID column is a transaction identification field that is required to join with the TransactionVision BUSINESS_TRANSACTION table, while the remaining columns contain business content that are extracted from the message data.

Step 3: Extract Transaction Attributes from Event Data

Now that we have determined the format and contents of the message data in Step 1, these event fields need to be set as transaction attributes. This is done in the Transaction Definition Editor by creating Attribute rules in the transaction class you are defining. These attributes are maintained by the Analyzer as it processes events and are then mapped to database tables defined in the transaction XDM file.

Once a transaction attribute has been defined, with an XPath location of **/Transaction/OrderID**. A Rule with a name of **SetOrderID** sets the value of the transaction attribute at XPath **/Transaction/OrderID** from the attribute value in the event data at XPath **/Event/Technology/Servlet/Response/Headers/Header[@name='orderid']**.

The two important pieces of information in the above attribute rule are the event XPath, which is the source of the data, and the transaction XPath, which is the destination to which the source data is copied into.

Value rules can also set constant values into transaction attributes. In the following example a constant value of **Completed** is set into the transaction attribute at XPath location **/Transaction/State**.

The attribute rules can be used in the context of class rules, which determine that the attribute rules are applied only for certain classes. Consider the example below:

Here, the attribute rule of name **OrderID** is applied only for already classified transactions of class “Bond”.

Attribute rules also can have match criteria such that the rules are applied to every event when a match criteria is successful.

Here, the value rule to set the value of **Amount** at XPath location **/Transaction/Amount** from the event XPath **/Event/Data/Chunk/Order/Amount**, is fired when the logical AND of the Match criteria evaluate to True.

Refer to Section 4.5.9 for details on the syntax of the classification rules.

Step 4: Construct XDM File Entries for Transaction Attributes

Now that we have determined the contents of the transaction attributes and extracted them from the event data as in Step (1) and (3) and determined which database tables need to be populated as in Step (2), a mapping can be created from the XML transaction attributes to the database columns.

Consider the below transaction document created by rules set XML segment:

```
<Transaction>
    <OrderID>123456</OrderID>
    <Account>   </Account>
    <Region>    </Region>
    <TradeType> </ TradeType >
    <TradeAction> </TradeAction>
    <Amount>    </Amount>
        ...
</Transaction>
```

The XPath to the OrderID field can be written as: **/Transaction/OrderID**.

The value at this XPath is to be written to the ORDERID column of the TRADE_BUSINESS_TRANSACTION table for the business transactions for which this value is set.

This mapping can be done in an XDM file as follows:

```
<Mapping documentType="/Transaction">
  <DBSchema>Trade</DBSchema>
  <Key name="business_trans_id" type="INTEGER"
generateSequence="true" description="TransactionId">
        <Path>/Transaction/BusinessTransId</Path>
  </Key>
  <Table name="TRADE_BUSINESS_TRANSACTION">
        <Column name="orderid" type="VARCHAR" size="20"
description="OrderID">
              <Path>/Transaction/OrderID</Path>
        </Column>
        <Column name="account" type="VARCHAR" size="12"
description="Account">
              <Path>/Transaction/Account</Path>
        </Column>
        ...
```

The above XDM file segment, the Table element defines a table name TRADE_BUSINESS_TRANSACTION. The table contains a column ORDERID, defined by the Column element, of type VARCHAR and size 20 bytes. The Column of name ORDERID has an XPath mapping, defined by the Path element to be **/Transaction/OrderID**. The key for the TRADE_BUSINESS_TRANSACTION is defined by the Key element to be business_trans_id column of type INTEGER.

The table definition part of the XDM segment is applied when a new project schema is created either by the CreateSqlScript or the project creation web pages. The XPath mapping part of the XDM segment is applied by the Analyzer when processing events. When a transaction contains data at the XPath value **/Transaction/OrderID** set by the classification rules, the Analyzer extracts the value from the transaction document and writes a row to the mapped column ORDERID belonging to table TRADE_BUSINESS_TRANSACTION for that transaction. The DBSchema attributes indicates that this mapping is applied only to transactions being written to the Trade schema.

Step 5: Determine the Transaction Classes and their Classification Criteria

Transaction classification can be based on a variety of different criteria based on the transactions flowing through your business systems. In the sample TRADE system, transaction classification is performed based on the type of financial transactions flowing through the system, namely Equity, Bonds and Funds Transfer. Hence, the next step would be to identify fields in the message data which identify the event and its transaction to be one of these three types. For this system, this field is an attribute Product in the XPath element **/Event/Technology/Servlet/Request/Parameters/Parameter**. The next section describes how to build a classification rule using this XPath value.

Step 6: Implement Classification Rules

Consider the Transaction definition example for the TRADE sample:

In the above image, a transaction class called Bond is defined, which applies to the database schema TRADE. Within the bond class there are one or more classification rules for the Bond transaction class.

The Match conditions specify the rule criteria. The first Match condition has a rule which evaluates to True when the XPath value of **/Event/StdHeader/ProgramNam**e in an event equals the value of TradeSession. Multiple Match conditions are logically AND'd together. The second Match condition criteria evaluates to True if a JMS event with the XPath element **/Event/Technology/JMS/Method** has a value of **publish**. In other words, any event with the program name **TradeSession**, a JMS method of **publish**, and a Product value of BOND will be classified to a **Bond** transaction class.

Values in "Match" criteria may contain one wildcard character, as in the following example:

```
"*FlowEngine", "DataFlow*", and "Data*Engine"
```

Once a transaction is classified, attributes are attached to the transaction based on the Attribute rules defined in the Transaction Definition editor. The rules for setting and writing attributes are described in Steps 3 and 4.

Step 7: Recreate the Database Schema

Existing database schemas need to be updated to create the newly added tables or columns. This can be done using options in the CreateSqlScript utility.

For example:

```
CreateSqlScript –c –e –n –p TRADE –t TRADE_BUSINESS_TRANSACTION
```

The above command creates the table TRADE_BUSINESS_TRANSACTION as defined in the XDM file in the TRADE database schema.

Step 8: Verify that the transaction classification works correctly and the transaction attributes are written correctly

The results of the above steps can be verified by looking at the Transaction Tracking Report. For each business transaction, this report will show you the class of the transaction and any custom attributes that have been set for that transaction. Other custom reports may be written based on the transaction attributes collected.

3.7. How to Perform Custom Correlation of Related Events

3.7.1. Overview of Custom Event Correlation

By default, the TransactionVision Analyzer correlates WebSphere MQ MQPUT and MQGET events or JMS send and receive events based on certain criteria such as message id, correlation id, put time and other fields in these events. However, there may be times when these criteria are not sufficient to perform event correlation. These criteria may then either need to be expanded to include other data fields, such as those from the message data, or may need to be relaxed to exclude some of the standard fields, or may need to be modified in other ways.

Here are some scenarios where a custom correlation bean may be required:

- TransactionVision agents may not be installed on some systems, such as those belonging to external agents. Hence, the messages going out to the un-sensored systems would need to be correlated with the replies coming back from these systems.
- Unique message ids or correlation ids are not used by the applications. In this scenario, custom fields from the message data may need to be used to correlate message PUTs and GETs.
- An application that replies to a message swaps the message id and correlation id fields and this application is not monitored by TransactionVision agents.

This correlation can be done by writing XML based event correlation rules in the EventCorrelationDefinition.xml file. Alternately, if complex logic is required to be implemented, a Java bean can be written to override the IEventCorrelation interface. Refer to Chapter 4, Section 4.5 on the details of a bean implementation.

3.7.2. Task Description

This task walks through the creation of a XML event correlation rule. The requirement for the bean is to correlate WebSphere MQ events for which the message id and correlation ids have been swapped.

3.7.3. Implementation

Step 1: Determine Correlation Requirements

Consider two applications A and B, where application A is monitored by a TransactionVision Sensor while application B is not. The sequence of events for this system is as follows:

- Application A performed an MQPUT on a queue q1, with message id m1 and correlation id c1.
- Application B read the message using an MQGET from queue q1 and processed the message.
- Application B then placed a reply message using MQPUT on the reply-to queue q2, with message id c1 and correlation id m1. Hence, the message ids and correlation ids were swapped by application B.
- Application A performed an MQGET to read this message.

Now, because application B does not have agents enabled and its MQGET/MQPUT are not received, this transaction path remains un-correlated and no message flow arc is drawn between application A’s MQPUT and application A’s MQGET. The custom event correlation bean seeks to complete this path.

Step 2: Determine which Events need to be Correlated and Common Correlation Data between the Events

For this task, the requirement is to correlate an MQPUT event from application A with an MQGET event from the same application A, which have their message id and correlation id swapped.

Step 3: Implement XML Based Event Correlation Rules

The correlation process in the Analyzer consists of two phases:

- The first phase involves generating lookup keys based on the characteristics of the current event. This lookup key is then inserted into the database and then used to match up with other correlated events as they arrive into the Analyzer. The XML event correlation rule file has a CreateLookupKey stanza that allows creation of custom lookup keys based on fields from the incoming event. If a bean is being implemented, the **createLookupKeys** method is invoked to generate these lookup keys. Hence, for application A for a MQPUT event, a lookup key comprising of the message id needs to be created, while for an MQGET event from application A, a lookup key comprising of the correlation id should be created.
- The second phase involves relation generation. Specifically, a set of events is passed as potential candidate for matching with the current event. This set is composed of the events that have the same lookup key as the current event. The purpose of this phase is to further narrow down set of event matches based on additional criteria which have not been covered by the lookup key data. For example, for application A, the correlation should only be performed between MQPUTs and MQGETs and not between APIs of the same type. This phase is implemented by creating a CreateRelation stanza in the XML event correlation definition file or by implementing the **correlateEvents** method of the event correlation bean.

The event correlation rule file is named **<TVISION_HOME>/config/services/EventCorrelationDefinition.xml**.

The basic template of a correlation rule file is as follows:

```
<EventCorrelationDefinition>
  <RelationLookupType id=1001" name="JMSToUserEvent"
dbschema="BROKER">
    <CreateLookupKey technology="UserEvent" id="1">
       . . .
    </CreateLookupKey>
    . . . .
    <CreateRelation keyRuleId1="1" keyRuleId2="2" id="1">
        . . .
    </CreateRelation>
    . . .
  </RelationLookupType>
</EventCorrelationDefinition>
```

Here, a RelationLookupType stanza is composed of one or more CreateLookupKey and CreateRelation stanzas. The CreateLookupKey stanza allows defining lookup keys from fields of certain events and the CreateRelation stanza allows matching up keys of different events.

The following is the event correlation rule file to correlate on the message id of a successful MQPUT with the correlation id of a successful MQGET. The steps following this listing describe the different stanzas in this file. The file is available in **<TVISION_HOME>/samples/stock**.

```
00001 <?xml version="1.0"?>
00002 <EventCorrelationDefinition>
00003 <!--
00004 Sample correlation rule file to correlate on swapped message id
and correlation
00005 ids for MQPUTs and MQGETs.
00006 -->
00007     <RelationLookupType id="1001" name="SwapMessageCorrelId"
dbschema="*">
00008
00009         <CreateLookupKey technology="MQSERIES" id="1">
00010             <Match xpath="/Event/Technology/MQSeries/@API"
operator="EQUAL" value="MQPUT"/>
00011             <Match xpath="/Event/Technology/
MQSeries/*/*Exit/CompCode" operator="UNEQUAL" value="2"/>
00012             <Attribute name="LookupKey">
00013                 <Path>/RelationLookup/LookupKey</Path>
00014                 <ValueRule name="SetLookupKey">
00015                     <Value type="XPath">/
Event/Technology/MQSeries/*/*Exit/MQMD/MsgId</Value>
00016                 </ValueRule>
00017             </Attribute>
00018         </CreateLookupKey>
00019
00020         <CreateLookupKey technology="MQSERIES" id="2">
00021             <Match xpath="/Event/Technology/MQSeries/@API"
operator="EQUAL" value="MQGET"/>
00022             <Match xpath="/Event/Technology/
MQSeries/*/*Exit/CompCode" operator="UNEQUAL" value="2"/>
00023             <Attribute name="LookupKey">
00024                 <Path>/RelationLookup/LookupKey</Path>
00025                 <ValueRule name="SetLookupKey">
00026                     <Value type="XPath">/Event/Technology/
MQSeries/*/*Exit/MQMD/CorrelId</Value>
00027                 </ValueRule>
00028             </Attribute>
00029         </CreateLookupKey>
00030
00031         <CreateRelation keyRuleId1="1" keyRuleId2="2" id="1">
00032             <Attribute name="RelationType">
00033                 <Path>/EventRelation/RelationType</Path>
00034                 <ValueRule name="SetRelationType">
00035                     <Value type="Constant">17</Value>
00036                 </ValueRule>
00037             </Attribute>
00038              <Attribute name="Direction">
00039                 <Path>/EventRelation/Direction</Path>
00040                 <ValueRule name="SetDirection">
00041                     <Value type="Constant">1</Value>
00042                 </ValueRule>
00043             </Attribute>
00044             <Attribute name="Confidence">
00045                 <Path>/EventRelation/Confidence</Path>
00046                 <ValueRule name="SetConfidence">
00047                     <Value type="Constant">1</Value>
00048                 </ValueRule>
00049             </Attribute>
00050         </CreateRelation>
00051
00052     </RelationLookupType>
```

```
00053
00054 </EventCorrelationDefinition>
```

- Line 7 provides the RelationLookupType stanza that contains the CreateLookupKey and CreateRelation rules. This element provides a constant id and name and defines the list of schemas to which its rules apply. An event correlation definition file may contain multiple RelationLookupType elements. The list of schemas in the dbschema attribute can be comma separated. Note that you should choose an id > 1000 for custom correlation types.
- Lines 9-18 define a lookup key rule for events from the MQSeries technology. Lines 10 and 11 define that this rule should be applied to all events with the API MQPUT and whose CompCode (completion code) is not equal to 2(failed). Lines 12-17 specify that when these criteria are matched for an event, a lookup key from the field MsgId is created for that event.
- Similarly, lines 20-29 create a lookup key from the CorrelId field for all successful MQGET APIs.
- The CreateRelation stanza on lines 31-54 specifies that the lookup keys created by rule id 1 and 2 should be matched up. Hence, two events that have the same lookup key created by rules 1 and 2, will have an event relation created. This event relation has the attributes of RelationType, Direction and Confidence set in the CreateRelation stanza.

Refer to Section “4.6.6 Custom Event Correlation” for details on customizing this rules file.

Step 4: Enable the Analyzer to Invoke the XML Correlation Rules

This involves editing the Beans.xml file to add the XML rule correlation bean, which then loads the **EventCorrelationDefinition.xml** rule file. The following line in bold needs to be added in the **Beans.xml** file:

```
<Module type="Context" name="CorrelationTechHelperCtx">
...
<Attribute name="UserCorrelationBean"
value="com.bristol.tvision.services.analysis.eventanalysis.XMLRuleCo
rrelationBean"

        <Module type="Context" name="CorrelationMQHelperCtx"
class="com.bristol.tvision.services.analysis.eventanalysis.Correlati
onMQHelperCtx">

          <!-- This context contains beans that perform MQ specific
event correlation. -->
            <!-- For each MQ event the bean that matches the
technology of the event to correlate with will be called. -->

            <Module type="Bean"
class="com.bristol.tvision.services.analysis.eventanalysis.MQToMQRel
ationshipBean"/>

    </Module>
```

Step 5: Test the Correlation Bean

The correlation bean can be verified by checking the transaction path in the transaction analysis view. A completely correlated path will have message path flows between local transactions.

4. Reference - Extending the Analyzer

This chapter contains the following sections:

4.1. Using the Beans.xml File

The file **Beans.xml** located in the **<TVISION_HOME>/config/services** directory controls the beans loaded by the Analyzer framework for event processing.

Important: This file is used by the Analyzer internally. Modifying sections that are not documented here could break the correct functioning of the Analyzer.

Each module listed in the Beans.xml file has a type and a name. The type can be a **Context**, which can hold other modules or a **Bean** type, which is loaded by a Context. A module of type **Bean** contains the class that implements an interface which is used by its context. Each context defines a known interface for the beans it contains, loads the bean and calls into the interface implemented by the bean to perform its function. In the example segment below, the EventModifierCtx is a bean context which holds the DefaultModifierBean bean.

```
<Module type="Context" name=" EventModifierCtx">
<Module type="Bean" class="com.bristol.tvision.services.analysis.
eventmodifier.DefaultModifierBean"/>
</Module>
```

Each context uses its own rules to determine how its beans are invoked. The following contexts can be modified or added to:

- EventModifierCtx
- DBWriteExitCtx
- CorrelationTechHelperCtx

The following sections will document how each of the above contexts can be modified.

The following section gives an example of plugging in your own implementation of EJB correlation analysis into the analysis framework:

```
<!-- context bean that allows build relationship between EJBs or EJB
to other technology
   Here is one sample. Note that the SampleEJBToMQRelationshipBean
is not in the shipment.

<Module
class="com.bristol.tvision.services.analysis.eventanalysis.Correlati
onEJBHelperCtx" name="CorrelationEJBHelperCtx" type="Context">
  <Module
class="com.bristol.tvision.extension.staples.services.SampleEJBToMQR
elationshipBean" type="Bean"/>
</Module>
!-->
```

4.1.1. Enabling and Disabling Beans for Specific Events

Many of the default beans of TransactionVision can be configured to only getting called for specific events by using match conditions similar to those used in custom correlation or local transaction analysis. This is accomplished by adding an <Include> or <Exclude> section to the bean definition in the Beans.xml file. The following sample definition will call the Transaction Classification bean only for MQ events:

```
<Module name="ClassifyTransactionCtx" type="Context">
  <Module
class="com.bristol.tvision.services.analysis.eventanalysis.StandardC
lassifyTransactionBean" type="Bean"/>
        <Include>
              <Match xpath="/Event/StdHeader/TechName"
operator="EQUAL" value="MQSERIES"/>
        </Include>
</Module>
```

And this sample definition will disable Transaction Classification for any MQ and JMS events:

```
<Module name="ClassifyTransactionCtx" type="Context">
  <Module
class="com.bristol.tvision.services.analysis.eventanalysis.StandardC
lassifyTransactionBean" type="Bean"/>
<Exclude>
              <Match xpath="/Event/StdHeader/TechName"
operator="EQUAL" value="MQSERIES"/>
              <Match xpath="/Event/StdHeader/TechName"
operator="EQUAL" value="JMS"/>
        </Exclude>
</Module>
```

The match conditions can also be defined on the context instead of the bean level, in which case they are effective for all beans of this context. The next sample definition will disable all event modifier beans for all MQ events:

```
<Module name="EventModifierCtx" type="Context">
  <Exclude>
        <Match xpath="/Event/StdHeader/TechName" operator="EQUAL"
value="MQSERIES"/>
  </Exclude>

  <Module
class="com.bristol.tvision.services.analysis.eventmodifier.DefaultMo
difierBean" type="Bean"/>
  <Module class=...>

</Module>
```

Note that you can only define either one <Include> or <Exclude> definitions for a bean or context, but not both. For a detailed description of the match conditions see the corresponding sections in Custom Correlation or Custom Local Transaction Analysis.

The following default TransactionVision analysis contexts (and their beans) can be configured:

- EventModifierCtx
- SystemModelCtx
- DBWriteExitCtx
- DBWriteEventCtx
- AnalyzeEventCtx
- CorrelationTechHelperCtx
- LocalTransactionTechHelperCtx
- ClassifyTransactionCtx

4.2. Unmarshalling Message Data

Typically, binary message data has a proprietary, user-defined format. The EventModifierCtx context allows a user to add a bean to "unmarshal" this binary data; that is, convert the binary data to XML for later use by TransactionVision in reports, for analysis or querying. To help converting binary data to XML, TransactionVision provides a set of utility classes.

4.2.1. The Default Modifier Bean

The TransactionVision installation comes with a default event modifier bean: **com.bristol.tvision.services.analysis.eventmodifier.DefaultModifierBean**. This bean scans the user data for any XML data and, if found, simply adds it to the Event XML document at the position **/Event/UserData/Chunk[@seqNo='n']** wher **'n'** is the number of the data range (defined in the data collection filter).

4.2.2. Adding a Message Data Unmarshal Bean

Adding a custom message or user data unmarshal bean involves modifying the Beans.xml file to replace the default class with one or more custom written classes.

```
<Module type="Context" name="EventModifierCtx">
<Module type="Bean"
class="com.bristol.tvision.demo.stock.StockTradePayloadProcessingBea
n"/>
  </Module>
```

For example, in the code snippet above, a bean **com.bristol.tvision.demo.stock.StockTradePayloadProcessingBean** processes any stock trade related custom data. If no event modifier bean is plugged in, the binary data will be saved into tables as a BLOB. The bean invoked by the EventModifierCtx context needs to implement the IEventModifier interface.

4.2.3. Disabling CICS Transaction Tracking

Typically in CICS environments, events from the same task are placed in the same business transaction. This assumption does not apply to long running transactions that repeat the same set of operations. Hence, the file **<TVISION_HOME>/config/services/Beans.xml** as follows can be modified to add a list of transaction ids for which events should not be placed in the same business transaction. A comma separated list of transactions may be entered.

```
<Module
class="com.bristol.tvision.services.analysis.unmarshal.mqseries.Unma
rshalMQHeaderBean" type="Bean">
        <Attribute name="UnmarshalMsgDataBean"
value="com.bristol.tvision.services.analysis.unmarshal.UnmarshalAppD
ataDefault"/>
        <Attribute name="MaxDataLength" value="-1"/>
        <Attribute name="DisableCICSTxnTrackingGen" value="BTCD"/>
</Module>
```

In this example, events from the CICS transaction BTCD will not have a tracking id based on their task number automatically generated by the Analyzer. Hence, they will not be automatically placed in the same business transaction, unless they belong to the same unit of work or if they are correlated to other events. The unit of work or event correlation behavior will not be affected by this setting.

4.2.4. IEventModifier Interface

This interface contains one method, modify(), defined as:

```
public boolean modify(XMLEvent event, ConnectionInfo conInfo)
throws EventModifyException
```

Description:

The method **modify()** is called to modify an unmarshaled XML event. For example, to convert the BLOB set stored in the XMLEvent object into the user-data section of the XML tree or modify the event's XML data. The BLOB set contains the event's binary message data.

The framework will check the return boolean value to decide whether to continue the event processing steps or not. A false return value means the current event under processing shall be discarded right away.

Important: data should typically be added in the XML event tree. Removing certain nodes from the tree could break the analysis and database write operations in later contexts.

Parameters:

event - The XML event to which the XML format of the message data is appended to. The XMLEvent class is documented in detail in Section 4.2.5.

conInfo - The connection information data structure.

Throws:

EventModifyException - This exception represents a failure in the bean performing the XMLEvent modification.

4.2.5. XML Related Classes

This section documents the relevant public methods of the classes XMLEvent, XPathSearch and XMLParser. Class XMLEvent contains the incoming event converted to an XML DOM tree. Class XPathSearch is a utility class to search a DOM tree using XPath queries. Class XMLParser is a wrapper class around the Apache DOM parser, with better error handling facilities.

The full TransactionVision event information is saved in XML document format. To retrieve values of different fields, an XPath expression is used to specify the location of the field. TransactionVision provides the file XPathConstants.java, which contains XPath expression constants used to locate different fields in the event. This file is useful for writing plug-in beans and reports and can be found at **<TVISION_HOME>/java/src**.

4.2.6. Class XMLEvent

```
package com.bristol.tvision.services.analysis.xml
public class XMLEvent
extends com.bristol.tvision.shared.xml.XMLDocument
implements java.io.Serializable
```

The class XMLEvent contains event data in XML DOM representation. It also holds a set of cached properties to carry inter-module communication information, and a list of BLOBs to hold application data which cannot be placed in the XML DOM tree. Note, that all the public methods of the class **org.w3c.dom.Document** are available to users of XMLEvent. The following methods are defined in the XMLEvent class.

Methods:

```
getAttribute
public java.lang.Object getAttribute(java.lang.String key)

setAttribute
public void setAttribute(java.lang.String key,
                         java.lang.Object value)
removeAttribute
public java.lang.Object removeAttribute(java.lang.String key)
```

The above three methods allow the user to set a cached value at one stage of event processing, which can be used at another point of event processing without parsing the XML document. For example during the unmarshal message data phase values can be stored which may later be used during analysis. Typically, the key would be an XPath into the XML document and the value would be the XML element value. The user of the above APIs must ensure that TransactionVision internal values are not overwritten or deleted. This can be done by using unique XPaths to message data as the key.

- getDocumentValue

  ```
  getDocumentValue
  public java.lang.String getDocumentValue(java.lang.String xpath)
  ```

  This method retrieves a value from the XML document specified by a given XPath expression . Since it takes advantage of the caching capabilities of XMLEvent it is the preferred method to access values in the XML document. If the value at the given XPath has never been accessed before, the method will perform an XPath search on the DOM tree to retrieve the value, otherwise it will return the value from the cache.

- getBlobCount

```
getBlobcount
public int getBlobCount()
```

Returns the number of BLOBs available, using the blobIterator() method.

- blobIterator

```
blobIterator
public java.util.Iterator blobIterator()
```

Typically, event message data is stored into one BLOB field in the XMLEvent object. However, if data ranges are used in the data collection filter an array of BLOBs is created, one BLOB for each data range. This method returns an Iterator for instances of type XMLEvent.Blob.

- deleteBlob

```
deleteBlob
public void deleteBlob(int seqNo,
                       boolean deleteUserDataRef,
                       boolean deleteDataChunk)
                       throws TVisionException
```

This method is used to delete the binary message data from XMLEvent. This method should typically be called if an EventModifier plugin bean converts binary data to XML. In that case, the binary data may no longer be required to be stored in the database and should be deleted using this method. If the message data is unmarshaled into the technology tree under, for example, the /Event/Technology/MQSeries/MQPUTEntry/Buffer subtree, the deleteUserDataRef and deleteDataChunk flags should be set to true. If the message data is unmarshalled into /Event/Data/Chunk, then both flags should be set to false. Also, if you want to replace a chunk with a different BLOB, call this method with both flags set to false and then call addBlob() to add a new BLOL into the XMLEvent.

Parameters:

seqNo - 0-based BLOB index

deleteUserDataRef – true if /Event//UserDataRef[@chunk=n] should be removed

deleteDataChunk – true if /Event/Data/Chunk[@seqNo=n] should be removed

- getPiiID

```
getPiiId
public long getPiiId()
```

- getEventSeqNo

```
getEventSeqNo
public int getEventSeqNo()
```

The PiiId (Program Instance Id) and the SeqNo (Sequence Number) together form a unique identifier to an event. They may be used to access event data from database tables.

- Inner Class XMLEvent.Blob

```
Inner Class XMLEvent.Blob
public static class Blob {
```

```
                public int id;          // id of the blob, starting
with 0
                public int from;        // data range start
                public int to;          // data range end
                public int type;        // type of BLOB data
                                        // (Binary, String, or XML,
                                        // defined in
TVisionCommon.java)
                public int ccsid;       // the character set id
                public byte[] blob;     // the data

                public Blob(int ID, int from, int to, int type,
int ccsid,
byte[] blob);

        }
```

Instances of this class are returned by the method 'blobIterator()' and represent the data ranges for the message data.

4.2.7. Class XPathSearch

```
package com.bristol.tvision.util.xml
public class XPathSearch
extends XPathSearchBase
```

The helper class XPathSearch allows access to elements of an XML document using the XPath syntax.

This class does not support the full standard XPath syntax. The following subset is supported:

- path to a text element: /Test/Value
- path to an attribute: /Test/Value/@attribute
- access a multi-valued element by qualifying attribute value:
  /Test/Value[@attribute='X']/Name
- indexed access to a multi-valued element /Test/List[0], Test/List[0]/Value
- wildcard /Test/*/Name, /Test/*lue, /Test/Val*

Constructor:

- XPathSearch

  ```
  XPathSearch(org.w3c.dom.Document doc)
  ```

  Creates an XPathSearch object from a DOM document or derived class like XMLEvent.

- XPathSearch

  ```
  XPathSearch(java.io.InputStream stream) throws XMLException
  ```

  Creates an XPathSearch object from an InputStream.

  The InputStream is parsed into a DOM document without validation

- XPathSearch

  ```
  XPathSearch(java.io.Reader reader) throws XMLException
  ```

  Creates an XPathSearch object from an InputStream.

  The InputStream is parsed into a DOM document.

Parameters:

stream - The InputStream containing the XML data

validate - Use parser validation

Methods:

- getNodes

```
public org.w3c.dom.NodeList getNodes(java.lang.String xpath)
                              throws XMLException
```

This method returns a list of all nodes in the XML document matching the XPath query. The elements in the array are ordered according to the order of the elements in the DOM tree.

Overrides:

getNodes in class XPathSearchBase

Parameters:

xpath - The XPath expression for the query

Returns:

A list of all nodes matching the query

Throws:

XMLException - Signals error during retrieving the values from the document

- getValues

```
public java.lang.String[] getValues(java.lang.String xpath)
                            throws XMLException
```

This method returns the value of all text elements in the XML document matching the XPath query. The elements in the array are ordered according to the order of the elements in the DOM tree.

Overrides:

getValues in class XPathSearchBase

Parameters:

xpath - The XPath expression for the query

Returns:

The value of all text elements matching the query

Throws:

`XMLException` - Signals error during retrieving the values from the document

- getValue

```
public java.lang.String getValue(java.lang.String xpath)
                 throws XMLException
```

This method returns the value of the first text element in the XML document matching the XPath query.

Overrides:

getValue in class XPathSearchBase

Parameters:

xpath - The XPath expression for the query

Returns:

The value of the first matching text element

Throws:

XMLException - Signals error during retrieving the values from the document

4.2.8. Class XMLParser

```
package com.bristol.tvision.util.xml
public class XMLParser
implements org.xml.sax.ErrorHandler
```

This class is a wrapper around the Apache DOM parser and is a utility useful to parse XML files or convert binary streams containing XML data into a DOM tree.

Constructor:

- XMLParser

```
XMLParser()
```

Creates a parser instance

Parameters:

validation – whether to create a validating parser or not

Methods:

- parse

```
public org.w3c.dom.Document parse(java.lang.String systemId,
                                  java.lang.String schema)
                           throws XMLException
```

Parses a XML file and uses the specified XML schema rather than a schema reference in the document itself for schema validation

Parameters:

systemId - The system id for the XML source

schema - The schema to use for validation

Returns:

The parsed document as a DOM tree

Throws:

XMLException - Signals errors during parsing

- parse

```
public org.w3c.dom.Document parse(org.xml.sax.InputSource src)
throws XMLException
```

Parses a XML document from an Input Source. If schema is not null, the parser property external-noNamespaceSchemaLocation is set for schema validation

Parameters:

src - The input source for the document

Returns:

The parsed document as a DOM tree

Throws:

XMLException - Signals an error during parsing

- parse

```
public org.w3c.dom.Document parse(java.io.InputStream stream,
                                  java.lang.String schema)
                           throws XMLException
```

Parses a XML document from an input stream and uses the specified XML schema rather than a schema reference in the document itself for schema validation

Parameters:

stream - The input stream for the document

schema - The schema to use for validation

Returns:

The parsed document as a DOM tree

Throws:

XMLException - Signals an error during parsing

4.2.9. Other Utility Classes

Often, binary structures embedded in the message data will need to be converted to XML. This can be accomplished with a two step process, first extract the binary data into Java data types and then convert these data types to appropriate XML elements. The Java class java.io.DataInputStream could be used to walk through a binary stream, extract and convert data into Java basic types. Also, the “Translator” class can be used to convert raw binary data into a Java UTF String with code page conversion:

```
package com.bristol.tvision.util.charmapper

public class Translator {

          public static Translator instance(int srcCcsid);
          public String translate(byte[] rawData);
}
```

Once Java basic types have been extracted from the binary stream these values need to be converted to XML data. This can be done using the utility XML builder classes in the package **com.bristol.tvision.services.analysis.xml**. These classes allow a user to set values

of native Java types, a element name and get the XML tag output appended to a DOM tree using the toDOM() method. These classes implement the DOMElement interface.

4.2.10. Interface DOMElement

public interface `DOMElement`

This class defines a common interface for classes which output XML into a DOM tree.

Methods:

- toDom

```
toDOM
public void toDOM(org.w3c.dom.Document doc,
org.w3c.dom.Node root)
```

  This method appends nodes to the DOM tree doc at node location root.

4.2.11. Class EventElement

```
public abstract class EventElement
extends java.lang.Object
implements DOMElement
```

This class is the super class of all XML builder classes that output XML elements into a DOM tree.

Methods:

- Constructor

```
public EventElement(java.lang.String name)
```

  The constructor of the EventElement class takes in the element name as a parameter. The element name is used by the toDOM method to output the node of element name to the XML DOM tree.

- toDOM

```
public abstract org.w3c.dom.Element
       toDOM(org.w3c.dom.Document doc, org.w3c.dom.Node root)
```

  This is the same method as in the interface DOMElement.

4.2.12. Class TextElement

```
public abstract class TextElement
extends EventElement
```

This class is a super class for those XML element classes which have only one text node as a child. This class allows adding attributes to the XML element.

Methods:

- Constructor

```
public TextElement(java.lang.String elementName)
```

  The constructor takes in the element name of the node to be inserted into the XML DOM tree.

- toDOM

```
public void toDOM(org.w3c.dom.Document doc,
                  org.w3c.dom.Node root)
```

Overrides:

toDOM in class EventElement

- addAttribute

```
public void addAttribute(java.lang.String name,
                         java.lang.String value)
```

This method allows adding a name-value pair of attributes to the XML element.

- hasNonNullValue

```
public abstract boolean hasNonNullValue()
```

This method returns true if this element has a non-null value and false otherwise.

4.2.13. Class ByteElement

```
public class ByteElement
extends TextElement
```

Fields:

- value

```
public byte value
```

This field holds the byte value to be converted to an XML DOM tree node by the toDOM method.

Constructors:

- ByteElement

```
public ByteElement(java.lang.String elementName)
```

The constructor takes in the element name of the tag to be output in the XML DOM tree node.

Methods:

- toDOM

```
public void toDOM(org.w3c.dom.Document doc,
org.w3c.dom.Node root)
```

This method appends a node containing the byte value held by the field value to the DOM tree doc at node location root with the element name elementName specified in the constructor of this object.

- toString

```
public java.lang.String toString()
```

Overrides:

toString in class java.lang.Object

This method converts the byte held in the field value to a string representation.

- hasNonNullValue

```
public boolean hasNonNullValue()
```

Overrides:

hasNonNullValue in class TextElement

This method returns true if this element has a non-null value and false otherwise.

4.2.14. Class ByteStringElement

```
public class ByteStringElement
extends TextElement
```

Fields:

- value

```
public byte[] value
```

This field holds the byte array value to be converted to an XML DOM tree node by the toDOM method.

Constructor:

```
public ByteStringElement(java.lang.String elementName)
```

The constructor takes in the element name of the tag to be output into the XML DOM tree node.

4.2.15. ByteStringElement

```
public ByteStringElement(java.lang.String elementName,
                         boolean isZOS)
```

The constructor takes in the element name of the tag to be output into the XML DOM tree node.

Methods:

- toDOM

```
public void toDOM(org.w3c.dom.Document doc,
                  org.w3c.dom.Node root)
```

Specified by:

hasNonNullValue in class TextElement

This method appends a node containing the byte array value held by value to the DOM tree doc at node location root with the element name elementName specified in the constructor of this object.

- toString

```
public java.lang.String toString()
Overrides:
toString in class java.lang.Object
```

This method converts a byte array held in the value field to a string representation.

- hasNonNullValue

```
public boolean hasNonNullValue()
```

Overrides:

hasNonNullValue in class TextElement

This method returns true if this element has a non-null value and false otherwise.

4.2.16. Class IntElement

```
public class IntElement
extends TextElement
```

Fields:

- value

```
public int value
```

This field holds the integer value to be converted to an XML DOM tree node by the toDOM method.

Constructors:

- IntElement

```
public IntElement(java.lang.String elementName)
```

The constructor takes in the element name of the tag to be output into the XML DOM tree node.

Methods:

- toDOM

```
public void toDOM(org.w3c.dom.Document doc,
                  org.w3c.dom.Node root)
```

This method appends a node containing the integer value held by field value to the DOM tree doc at node location root with the element name elementName specified in the constructor of this object.

- toString

```
public java.lang.String toString()
Overrides:
toString in class java.lang.Object
```

This method converts an integer to a string representation.

- hasNonNullValue

```
public boolean hasNonNullValue()
```

Overrides:

hasNonNullValue in class TextElement

This method returns true if this element has a non-null value and false otherwise.

4.2.17. Class IntHexElement

```
public class IntHexElement
extends IntElement
```

This class's toDOM method outputs an integer value to a XML DOM node element as a hexadecimal string.

4.2.18. Class LongElement

```
public class LongElement
extends TextElement
```

Fields:

- value

```
public long value
```

This field holds the integer long value to be converted to an XML DOM tree node by the toDOM method.

Constructors:

- LongElement

```
public LongElement(java.lang.String elementName)
```

The constructor takes in the element name of the tag to be output into the XML DOM tree node.

Methods:

- toDOM

```
public void toDOM(org.w3c.dom.Document doc,
                  org.w3c.dom.Node root)
```

This method appends a node containing the integer long value held by the field `value` to the DOM tree `doc` at node location `root` with the element name `elementName` specified in the constructor of this object.

- toString

```
public java.lang.String toString()
Overrides:
toString in class java.lang.Object
```

This method converts the integer long held in the field `value` to a string representation.

- hasNonNullValue

```
public boolean hasNonNullValue()
```

Overrides:

hasNonNullValue in class TextElement

This method returns true if this element has a non-null value and false otherwise.

4.2.19. Class LongHexElement

```
public class LongHexElement
extends LongElement
```

This class's toDOM method outputs an integer long value to a XML DOM node element as a hexadecimal string.

4.2.20. Class StringElement

```
public class StringElement
extends TextElement
```

Fields:

- value

```
public java.lang.String value
```

This field holds the String value to be converted to an XML DOM tree node by the toDOM method.

Constructor:

- StringElement

```
public StringElement(java.lang.String elementName)
```

The constructor takes in the element name of the tag to be output into the XML DOM tree node.

Methods:

- toDOM

```
public void toDOM(org.w3c.dom.Document doc,
                  org.w3c.dom.Node root)
```

This method appends a node containing the String value held by the field value to the DOM tree doc at node location root with the element name elementName specified in the constructor of this object.

- toString

```
public java.lang.String toString()
Overrides:
toString in class java.lang.Object
```

This method converts the String held in the field value to a string representation.

- hasNonNullValue

```
public boolean hasNonNullValue()
```

Overrides:

hasNonNullValue in class TextElement

This method returns true if this element has a non-null value and false otherwise.

4.2.21. Class RawStringElement

```
public class RawStringElement
extends TextElement
```

This class's toDOM method outputs a String value to a XML DOM node element as a string whose non-ASCII characters are converted to hexadecimal values.

4.3. Trimming Data From an Event

The DBWriteCtx context is invoked by the Analyzer framework before the database write operation. It gives a user defined bean an opportunity to trim out data from the XML event packet. Beans loaded by this context need to implement the IDBWriteExit interface.

4.3.1. Interface IDBWriteExit

```
public interface IDBWrite
```

Methods

- modify

```
public XMLEvent modify(XMLEvent event)
                throws DBWriteExitException
```

This method trims data off the XML event. The bean has to make a copy of the XML event and return the trimmed copy.

Parameters:

event - The XML event to trim.

Returns:

The return value is the trimmed XML event

Throws:

TrimEventDataException - Trimming of the event failed

The sample code under <TVISION_HOME>/samples/dbwritexit shows how to write a bean to plug into the database write exit context.

4.4. XML-Database mapping Using XDM Files

The TransactionVision database schema is made extensible through the XML to Database Mapping (XDM) files. As new technologies or message data specific information is added, new XDM files can be written to describe the lookup tables for the technology and message-specific data in those events. Hence the purpose of the XML to Database mapping is twofold:

- To describe which fields are to be extracted from the XML event and transaction data and stored in lookup tables for fast searching and retrieval.
- To make the database schema partially data-driven.

The definitions contained in the XML Database Mapping (XDM) file are used as input not only to the TransactionVision Data Manager (including the query services), but also to a program that generates the commands necessary to create the lookup tables.

The XDM mappings can be technology or platform specific. The common mapping defined in the file **<TVISION_HOME>/config/xdm/Event.xdm** (data in the standard event header) will be written for every event, but the mappings defined in the other XDM files will only be applied if the current event matches the mapping's "category' (technology or platform) definition. The XML schema format of XDM files is defined in **<TVISION_HOME>/config/xmlschema/XDM.xsd**. The following code is an extract from the file **Event.xdm**.

```
<?xml version="1.0"?>
<Mapping documentType=”/Event”>
        <Key name="proginst_id" type="BIGINT"
description="ProgramInstanceId">
                <Path>/Event/EventID/@programInstID</Path>
        </Key>
        <Key name="sequence_no" type="INTEGER"
description="SequenceNumber">
                <Path>/Event/EventID/@sequenceNum</Path>
        </Key>
        <Table name="EVENT_LOOKUP" categoryPath="COMMON">
            <Column name="host_id" type="BIGINT" description="Host"
isObject="true">
                <Path>/Event/StdHeader/Host/@objectId</Path>
            </Column>
            <Column name="program_id" type="BIGINT"
description="Program" isObject="true">
    <Path>/Event/StdHeader/ProgramName/@objectId</Path>
            </Column>
```

```
        ...
        </Table>
</Mapping>
```

The above snippet from Event.xdm defines a table EVENT_LOOKUP, containing various indexed columns of data from the XML document. The key columns proginst_id and sequence_no are mapped to XPath expressions /Event/EventID/@programInstID and /Event/EventID/@sequenceNum. These key columns are primary keys common to all event based lookup tables. Similarly columns host_id and program_id are mapped to XPath expressions /Event/StdHeader/Host/@objectId and /Event/StdHeader/ProgramName/@objectId respectively.

The above XDM file specifies that when an XML event is written to the database by the DBWrite module in the Analyzer, these fields are extracted and written into the database columns mapped to in the XDM file. Similarly, when the database is queried using the QueryService XML interface, these XDM files are used to construct the corresponding SQL query.

For more details on the XDM functionality, see chapter 10.2.

4.5. Performing Event Analysis

There are five categories of event analysis activities defined in TransactionVision:

- **Event Correlation:** Establishing relation(s) between any two events. Examples include message path relation representing a message flow from one event to another, and transaction path relation representing a control flow between the two events.
- **Local Transaction Analysis**: Grouping events of the same technology that participate in the same unit of work in the same thread of execution into one local transaction object.
- **Business Transaction Analysis**: Grouping local transaction objects participating in the processing of the same business activity instance into one business transaction object. This is achieved by establishing relation between any two local transaction objects through the corresponding message path or transaction path relation of respective events in the local transaction objects.
- **Statistics Analysis:** Calculating event statistics for the Static Topology View
- **User Analysis**: This can be any customized infrastructure or business level analysis.

Each event analysis task is implemented in an event analysis bean. The class AnalyzeEventBean defines the base class for these beans.

The individual beans are managed under a multi-level analyze event context framework. The class AnalyzeEventCtx defines the top level context. The set of beans to be managed under this context are specified in the Beans.xml file. Each registered bean is executed following the order defined in the file. The following is an example of the event analysis context setup for the stock trade simulation example:

```
<Module type="Context" name="AnalyzeEventCtx">

<!-- This context contains beans that perform transaction analysis. -
->
<!-- Each registered bean in the chain is called. -->

   <!—- TransactionVision Event Correlation bean -->
```

```
<Module type="Bean"

class="com.bristol.tvision.services.analysis.eventanalysis.EventCo

rrelationBean"/>

<!-- TransactionVision Local Transaction Analysis bean -->

<Module type="Bean"
 class="com.bristol.tvision.services.analysis.eventanalysis.Local
 TransactionAnalysisBean"/>
<!-- TransactionVision Default Business Transaction Analysis bean

-->

<Module type="Bean"

class="com.bristol.tvision.services.analysis.eventanalysis.Busines

sTransactionAnalysisBean">

<!-- TransactionVision Statistics beans -->

<Module type="Context" name="StatisticsCtx"
class="com.bristol.tvision.services.analysis.statistics.Statistics
Ctx">
   <Module type="Bean"
class="com.bristol.tvision.services.analysis.statistics.MQStatisti
csBean"/>
   <Module type="Bean"
class="com.bristol.tvision.services.analysis.statistics.JMSStatist
icsBean"/>
   <Module type="Bean"
class="com.bristol.tvision.services.analysis.statistics.ServletSta
tisticsBean"/>
   <Module type="Bean"
class="com.bristol.tvision.services.analysis.statistics.EJBStatist
icsBean"/>
</Module>

<!—User Analysis bean for the stock trade simulation -->
<Module type="Bean"
 class="com.bristol.tvision.demo.stock.StockTradeAnalysisBean"/>
</Module>
```

4.5.1. Event Analysis Utility Classes and Interface

The following utility classes are extensively used in implementing various types of event analysis beans.

4.5.2. Interface Cache

```
package com.bristol.tvision.util.cache
public interface Cache
```

TransactionVision maintains various in-memory caches for miscellaneous objects. These caches are implemented as LRU caches, meaning that always the most recent processed data is available. For example, a local transaction cache is maintained to store a mapping from

event ID to local transaction data. This interface defines the methods for manipulating the cache.

Methods:

- insert

```
public void insert(java.lang.Object key, java.lang.Object value)
```

  Insert a new key-value pair into the cache.

  Parameters:

  key - new cache object key field

  value - new cache object value field

- get

```
public Object get(java.lang.Object key)
```

  This method returns the value field of the cache entry with the matching key.

  Parameters:

  key - key field of the cache entry to be matched

  Returns:

  The value field of the cache entry if a matching object is found.

- remove

```
public void remove(java.lang.Object key)
```

  Remove the cache entry with the matching key.

  Parameters:

  key - key field of the cache entry to be matched

- removeAll

```
public void removeAll()
```

  Remove all cache entries.

- getSize

```
public int getSize()
```

  Return the defined cache size specified in the CacheProperty file.

  Returns:

  The defined cache size

- resize

```
public void resize(int size)
```

  Resizes (and clears) the cache.

  Parameters:

  size - new cache size

- getElementCount

```
public int getElementCount()
```

  Return the current number of cache entries.

  Returns:

  The current number of cache entries.

- getCacheName

```
public java.lang.String getCacheName()
```

  Return the name of this cache.

  Returns:

  The name of this cache

4.5.3. Class ConnectionInfo

```
package com.bristol.tvision.datamgr
public class ConnectionInfo
```

This class is a simple structure for holding the TransactionVision database connection and schema name within an object which can be passed through the event analysis service framework.

Fields:

- con

```
public java.sql.Connection con
```

  A TransactionVision Connection object to the database. This connection object implements the Java SQL Connection object interface.

- schema

```
public java.lang.String schema
```

  String for the current project database schema.

4.5.4. Class EventID

```
package com.bristol.tvision.datamgr.dbtypes
public class EventID
```

Each event is uniquely identified by a pair of integer ID: a program instance (PII) ID and a sequence number. The program instance ID points to the program instance (threads, tasks, etc.) the event occurs within. This class defines a wrapper around these two identifiers for an event.

Constructor:

- EventID

```
EventID(int piiId, int seqNo)
```

  Creates an event ID object for an event with the program instance ID piiId and sequence number seqNo.

Fields:

- piiId

```
public int piiId
```

The program instance id for this event

- seqNo

```
public int seqNo
```

The sequence number of this event

Methods:

- equals

```
public boolean equals(EventID eventId)
```

Parameters:

eventId - eventId to be matched

Returns:

true if the event ID matches, false otherwise.

- hashCode

```
public int hashCode()
```

Return a unique integer has code for this event ID object.

Returns:

The integer hash code for this event ID object

Overrides:

equals in class java.lang.Object

Parameters:

eventID - EventID object to compare to.

Returns:

true if the two EventIDs are equivalent.

Determine if the input event is the same as this event.

- toString

```
public java.lang.String toString()
```

Returns:

A string describing this event ID object.

4.5.5. Class TechEventID

```
package com.bristol.tvision.datamgr.dbtypes
public class TechEventID
```

This class extends class EventID and additionally holds the technology ID of the event.

Constructor:

- TechEventID

```
TechEventID(int piiId, int seqNo, int techId)
```

Creates an event ID object for an event with the program instance ID piiId, sequence number seqNo., and technology ID techId

Fields:

public int techId

The technology ID for this event.

4.5.6. Interface IAnalyze

```
package com.bristol.tvision.services.analysis.eventanalysis
public interface IAnalyze
```

This defines the interface for general-purpose event analysis beans.

Methods:

- analyze

```
public void analyze(XMLEvent event, ConnectionInfo)
                        throws AnalyzeEventException
```

This method implements a specific event analysis task on the given event.

Parameters:

conInfo - database connection info object for the current project

event - completed XML document for the current event

Throws:

AnalyzeEventException - Signals errors during the event correlation analysis

4.5.7. Class AnalyzeEventCtx

```
package com.bristol.tvision.services.analysis.eventanalysis
public class AnalyzeEventCtx extends ChainManagerCtx implements
IAnalyze
```

This is the top level event analysis context class and holds all analysis beans for the event analysis. During analysis, the analyze() interface will be called for all beans contained in this context (in sequential order).

4.5.8. Class AnalyzeEventBean

```
package com.bristol.tvision.services.analysis.eventanalysis
public abstract class AnalyzeEventBean extends ChainManagedBean
implements IAnalyze
```

This is the abstract base class for all event analysis beans. Any custom event analysis bean should derive directly or indirectly from this class, and implement the IAnalyze interface methods.

Fields:

- Analysis Type

  public static final int EVENT_CORRELATION = 1;

  public static final int LOCAL_TRANSACTION_ANALYSIS = 2;

  public static final int BUSINESS_TRANSACTION_ANALYSIS = 3;

  public static final int BUSINESS_PROCESS_ANALYSIS = 4;

  public static final int USER_ANALYSIS = 5;

The type of analysis implemented by the event analysis bean instance.

Methods:

- getAnalysisType

```
public int getAnalysisType()
```

  Return the analysis type of the event analysis bean.

4.5.9. Custom Business Transaction Attributes and Classification

Business transaction attributes are stored in the table BUSINESS_TRANSACTION which is defined by an XDM file, and thus are easily extensible. Additional custom business transaction attributes can be simply added by modifying the corresponding **Transaction.xdm** file. The table schema which is defined by the standard definition in **Transaction.xdm** contains the following columns (among others that are used internally):

- business_trans_id: a unique ID for the transaction generated by the database
- class_id: the ID of the transaction class (FK into table transaction_class)
- starttime: the start time of the transaction
- endtime: the end time of the transaction
- responsetime: the time difference between start and end timestate: the current state of the transaction (-1=Unknown, 0=Processing, 1=Completed)
- result: the result of the transaction (-1=Unknown, 0=Failed, 1=Success)
- exception: the transaction has been flagged erroneous (0=false, 1=true)
- label: a label for the transaction to display in the GUI
- sla_state: the SLA state of the transaction (0=None, 1=Violated, 2=AgedOut)
- value: the transaction value (based on the currency defined in the transaction class)
- update_id: a unique ID which gets incremented every time the transaction has been updated
- events_stored: whether event data has been stored (only applicable in failure mode, 0=No, 1=Yes)

When modifying the XDM definition to add custom business transaction attributes it is important not to alter or delete any of those predefined, standard, attributes.

If no standard or custom transaction classification bean is plugged in into the Analyzer framework, the attributes are populated with the following values during event transaction analysis:

business_trans_id	generated by the database
class_id	0 (Unclassified)
starttime	time of the earliest event in this business transaction
endtime	time of the latest event in this business transaction
state	-1 (Unknown)
result	-1 (Unknown)
exception	0 (false)

label	null
sla_state	0 (None)
value	null
events_stored	1 (in normal mode), 0 (in failure mode)
responsetime	difference between starttime and endtime
update_id	generated by the database

There are two different ways to populate the values of custom transaction attributes or to modify the default values of the standard attributes:

- Use the StandardClassifyTransactionBean and define rules how to classify transactions and update attribute values. This approach does not require any additional coding, only the classification rules have to be defined in the Transaction Definition Editor .
- Write a custom classification bean that implements the IClassifyTransaction interface. This approach is useful if more complex transaction classification is needed than the standard classification bean can provide

4.6. Transaction Classification

By default, TransactionVision does not classify the business transactions it processes; the class ID of each transaction will be 0 (Unclassified), indicating that this transaction does not belong to any transaction class. To enable transaction classification, you have to define your classes and classification rules in the Transaction Definition Editor.

4.6.1. Transaction Classification with the Standard Classification Bean

The StandardClassifyTransactionBean is a default implementation of a classification bean and allows user customized transaction classification without the need to write a single line of code. Although the rule engine of this standard bean is simple and fairly limited, it may well be sufficient for a great amount of classification cases. It is well suited for transactions that can be classified based on the attributes of one event of the transaction.

The classification logic is driven by classification rules defined in the Transaction Definition Editor which specify how and when transactions are classified and transaction attributes are set or updated. These rules will get evaluated for each event being processed in the transaction analysis in the Analyzer. Each class can be assigned to one or more database schemas, so that for an event of a particular project only class rules that are valid for the project schema will get evaluated. For more information about the Transaction Definition Editor, see the *Using Transaction Management*.

Prior to TransactionVision version 7.50 classification rules have been defined in XML form in the **TransactionDefinition.xml** file, but now the class definitions can be easily created and edited from within the TransactionVision application of HP BSM. In the following sections we will continue to describe the structure of the classification rules in XML form, since it relates tightly to the structure of presenting and editing the various components in the Transaction Definition Editor. Internally, the class definitions are stored in XML format in the TVISION system table CLASSIFICATION.

The main structure of a class definition is:

```
<Class name="StockTrade">
  <Classify>
```

```
        Conditions for setting the class
        Rules for updating the transaction attributes (once)
        Action rules
  </Classify>

  <Classify>
        Different conditions for setting the class
        Rules for updating the transaction attributes (once)
        Action rules
  </Classify>

Rules for updating the transaction attributes

</Class>
```

Each <Class> definition consists of one or more <Classify> sections that contain rules for identifying the transaction class, a list of rules to set transaction attribute values at the time of classification, and a list of action rules (described later), followed by a list of rules outside of the <Classify> section for setting attribute values of all events of the transaction.

The evaluation flow is as follows:

- If the current transaction has not been classified yet (class_id == -1, Unclassified), then all <Classify> sections of all class definitions matching with the current event schema are evaluated. If a classification is successful, the transaction class ID of the transaction will get set and all attribute rules contained in the class definition will get evaluated as well. No further <Classify> section will be evaluated any more. If none of the classifications are successful, the union of all attribute rules (outside of <Classify> sections) of all class definitions for the current event schema are evaluated.

  Note: This is necessary because the processing order of events in the analyzer can be different to the order the events really happened, and the classification algorithm needs to make sure that all rules for a certain class will get evaluated even if the event which will classify the transaction will be processed at a later time. As a consequence, rules outside of <Classify> sections should always be specific enough (by defining appropriate matching rules) to match only on events of the class they are meant for, because they will also get executed on events that might belong to another class for which the classifying event has not been processed yet.

- If the current transaction already has its class attribute set, only the attribute rules in the corresponding class definition outside of the <Classify> sections are evaluated. The conditions inside of the corresponding <Classify> section are not evaluated again.

  Each <Classify> section contains one ore more <Match> conditions, e.g.:

```
<Class name="StockTrade">
<Classify>
<Match xpath="/Event/Technology/JMS/Caller" operator="EQUAL"
value="StockTrade"/>
   <Match xpath="/Event/Technology/JMS/MQObject/Queue"
operator="EQUAL" value="TRADE_REQUEST"/>
      {…}
```

If the logical AND of these conditions results in true, the current transaction is considered to be ‘classified’, and the class_id attribute of the current transaction is set to the corresponding class ID of the definition class. In general, a match condition consist of a @xpath,

@operator, and @value attribute. The @xpath attribute specifies a certain value from either the current XML event or the transaction document. @operator can be one of the following:

- EQUAL, UNEQUAL: compares the value in the document (specified by xpath) against the string in 'value'. For EQUAL, a single wildcard "*" is allowed at any position, e.g. "amqsput*", "*QUEUE", "TV*QUEUE". For strings without wildcard both operators are case-insensitive.
- GREATER, LESS, GREATEREQUAL. LESSEQUAL: compares the numeric value in the document against the numeric value of the string in 'value'
- EXISTS, NOTEXISTS: checks for existence of any value at the specified xpath. The 'value' attribute is ignored and should be set to ""
- SUBSTRING: matches if the value in the document contains the string in 'value' as a substring. This operator is case-sensitive.
- REGEXPR: matches if the regular expression given in 'value' matches the value in the document. Examples for regular expressions are "MQPUT|MQGET", "QUEUE[1-9]", etc. See the Java documentation for regular expressions at http://java.sun.com/j2se/1.5.0/docs/api/java/util/regex/Pattern.html for details.

@value can either contain a literal string value - an enumeration constant (if there is an enumeration defined for this XPath), or another XPath expression into the current event or transaction document. The condition gets evaluated by string comparison of the document value with the specified value.

As mentioned before, the match conditions in one <Classify> section are logically AND-ed together. To specify an alternative set of conditions (logical OR), one or more additional <Classify> sections for the same class can be added.

Attribute rules are used to set and update values of transaction attributes. They can either be defined inside of a <Classify> section, in which case they are only evaluated once at the time of classification, or they can appear outside of the <Classify> section if they have to be evaluated for all events the transaction.

Here is an example of such an attribute rule:

```
<Attribute>
<Path>/Transaction/Declined</Path>
<ValueRule>
<Match xpath="/Event/StdHeader/ProgramName" operator="EQUAL"
value="DeclineTrade01"/>
   <Match xpath="/Event/Technology/MQSeries/MQObject/@objectName"
operator="EQUAL" value="TRADE_REPLY"/>
  <Value type="Constant">true</Value>
</ValueRule>

<ValueRule>
<Match xpath="/Event/StdHeader/ProgramName" operator="EQUAL"
value="DeclineTrade02"/>
  <Match xpath="/Event/Technology/MQSeries/MQObject/@objectName"
operator="EQUAL" value="TRADE_REPLY"/>
   <Value type="Constant">true</Value>
</ValueRule>
</Attribute>
```

Each <Attribute> element defines rules for setting the value of a certain transaction attribute. The <Path> element specifies the Xpath for the transaction attribute. The possible values for this transaction attribute are specified in one or more <ValueRule> sections. Each

<ValueRule> specifies a set of match conditions (logical AND) and the new value for the attribute if the match conditions fire. The <ValueRule> definitions for an <Attribute> are evaluated in sequential order, and once a certain rule has fired, the transaction attribute will get updated with the value defined within this rule, and all following <ValueRule> sections will get skipped.

The new values for a transaction attribute are specified within the <Value> element and can have one of two possible types (specified with the `@type` attribute):

- "Constant" specifies a literal String value or an enumeration constant (if there is an enumeration defined for this XPath)
- "XPath" specifies that the new value should be retrieved dynamically at runtime from either the XML event or transaction document

It is possible to specify multiple <Value> element for one attribute, in which case the attribute value will be the concatenation of all evaluated <Value> definitions, like .e.g.:

```
<Attribute>
<Path>/Transaction/Label</Path>
<ValueRule>
<Value type="XPath">/Event/Data/Order/Ticker</Value>
   <Value type="Constant">_</Value>
   <Value type="XPath">/Transaction/Account</Value>
   <Value type="Constant">_</Value>
   <Value type="XPath">/Transaction/OrderID</Value>
</ValueRule>
</Attribute>
```

Every time the transaction analysis calls into the standard classification bean for an event all <Attribute> definitions for the corresponding transaction class are getting evaluated in sequential order. But by default the <Attribute> rules are only evaluated if the corresponding transaction attribute has no value yet, the definition is considered to be final. Once a final rule has set the value of the transaction attribute, it (and other final rules that refer to the same attribute) will not be evaluated again.

To allow transaction attributes to get set and updated more than once, the attribute rule can be declared with an attribute @final set to "false":

```
<Attribute final=”false”>
        <Path>/Transaction/EndTime</Path>
        {…}
```

This forces an attribute rule to get evaluated every time, even when the transaction attribute is already set. An attribute rule without the @final attribute is equivalent to @final="true".

Another rule attribute, @precedence, can be used to control the setting of new values for transaction attributes:

```
<Attribute precedence="true">
<Path>/Transaction/State</Path>
{…}
```

This attribute can only be set for rules referencing integer valued transaction attributes. If set to true then an existing attribute value only gets overwritten if the new value is greater than the old value. This mainly makes sense for 'state' and 'result' like attributes where all values can be ordered according to a priority (e.g. UNKNOWN->PROCESSING->COMPLETE),

though in general it can be applied to any integer valued attribute. All @precedence rules are automatically considered to be non-final too. By default (if the @precedence attribute is not specified) the value is false.

In addition to the <Class> definitions you can also define one or more <Common> sections in the UI Transaction Definition Editor. The structure of a <Common> definition is similar to the <Class> definition:

```
<Common name="Common1">
  <Evaluate>
Conditions for triggering the following rules
Rules for updating the transaction attributes
Action rules
  </Evaluate>

  <Evaluate>
Conditions for triggering the following rules
Rules for updating the transaction attributes
Action rules
  </Evaluate>

</Common>
```

Unlike the rules defined in the <Class> sections, the rules defined in the <Common> section are valid for all classes (including UNCLASSIFIED) and will get evaluated on every event, irrespectively of the classification status. Like <Class> definitions, <Common> sections can be assigned to one or more project schemas to restrict the evaluation of the sections to events of those projects.

Any transactions that have been successfully classified will show up with their respective class name in the reports that categorize by class, such as the Transaction Tracking Report. Also, any errors that are encountered during the classification process will get logged in the Analyzer.log file.

4.6.2. Classification Action Rules

In addition to setting and updating transaction attribute values, the classification rules can also trigger custom actions. Action rules specify a java class implementing **com.bristol.tvision.services.analysis.eventanalysis.IAnalyzerAction** and can appear in two locations of the classification rules:

- As part of an Attribute rule. The action is executed after the value of the attribute has been updated

```
<Attribute precedence="true">
   <Path>/Transaction/State</Path>
      <ValueRule >
         <Match xpath="/Event/StdHeader/ProgramName"
operator="EQUAL" value="TradeServlet"/>
         <Value type="Constant">Completed</Value>
         <Action type="JAVACLASS" code="1"
reason="TestInvocation">com.bristol.tvision.services.analysis.acti
ons.SampleAction</Action>
      </ValueRule>
</Attribute>
```

  In this example, if the event is generated from the program **TradeServlet**, the transaction attribute named **State** will be set to "Completed". After the attribute has been

updated, the bean specified in the action tag is invoked. The sample bean logs information about the event and the transaction to the analyzer trace log.

The <Action> rule can have the following attributes: type, code, and reason. Currently, the only Action type available is “JAVACLASS”. Code and reason provide a means of passing an integer and/or string for use in the action method. They are not required. If the action is invoked as part of an Attribute definition, a reference to the transaction attribute is also passed into the action method call.

- As part of a <Classify> or <Evaluate> section:

```
<Classify>
      <Match….>
      <Attribute….>
<Action type="JAVACLASS" code="1"
reason="TestInvocation">com.bristol.tvision.services.analysis.acti
ons.SampleAction</Action>
</Classify>

<Evaluate>
      <Match….>
      <Attribute….>
<Action type="JAVACLASS" code="1"
reason="TestInvocation">com.bristol.tvision.services.analysis.acti
ons.SampleAction</Action>
</Evaluate>
```

The actions defined inside of <Classify> and <Common> sections will be executed in textual order, after all attribute rules have been evaluated.

If you write a custom action class, it must implement com.bristol.tvision.services.analysis.eventanalysis.IAnalyzerAction interface and must provide an action method to be invoked by the standard classification bean:

```
public interface IAnalyzerAction {

  boolean action(int code, String reason, XMLDocument inputDoc,
XMLDocument outputDoc, Attribute attr, ConnectionInfo conInfo);
}
```

The code and reason string will get passed in from the rule definition (null if not specified). InputDoc is the event XML document, and outputDoc is the transaction XML document. Attr is a reference to the updated transaction attribute if the action has been invoked as part of an attribute rule, or null otherwise. To identify which attribute triggered the action, you can reference **attr.attrPath** which contains the XPath for the attribute. The method has to return **true** if it has modified the transaction document, or **false** otherwise.

The custom class has to be added to the Analyzer’s CLASSPATH.

4.6.3. The ClassifyTransactionCtx and the IClassifyTransaction Interface

Transaction classification beans are plugged in into the Analyzer framework by placing them into the ClassifyTransactionCtx in the **Beans.xml** file; for example:

```
<Module type="Context" name="ClassifyTransactionCtx">
        <Module type="Bean"
class="com.bristol.tvision.services.analysis.eventanalysis.StandardC
lassifyTransactionBean"/>
    </Module>
```

The context can contain multiple beans, in which case the beans are processed in sequential order. Each classification bean has to extend **com.bristol.tvision.services.analysis.framework** and implement the **IClassifyTransaction interface** which contains the following two methods:

- classify

```
public boolean
classify(com.bristol.tvision.services.analysis.xml.XMLEvent event,

com.bristol.tvision.services.analysis.eventanalysis.XMLTransaction txn,
com.bristol.tvision.datamgr.dbtypes.EventID[] correlatedEvents,
com.bristol.tvision.datamgr.ConnectionInfo conInfo)
       throws
com.bristol.tvision.services.analysis.eventanalysis.AnalyzeEventExceptio
n
```

  Parameters:

  event - The current event.

  txn - The transaction document for the current event.

  correlatedEvents - The list of correlated events.

  conInfo - The current database connection.

  Returns:

  true if the transaction doc has been updated, false otherwise

  Throws:

  com.bristol.tvision.services.analysis.eventanalysis.AnalyzeEventException - The analysis process failed.

- hasTimeRules

```
public boolean
hasTimeRules(com.bristol.tvision.datamgr.ConnectionInfo conInfo.
String className)
```

  Returns whether the classification rules for the given class contain rules for start/endtime of the transaction (/Transaction/Starttime and /Transaction/Endtime).

  Parameters:

  conInfo - The current database connection

  className - The name of the class

Returns:

true if the given class has rules for start/endtime of the transaction, false otherwise

For each event that gets processed during the event transaction analysis phase the classify method of each registered classification bean will be called, and the logical OR of all bean invocations will be returned back to the transaction analysis phase in the Analyzer. If the returned value is true (meaning one or more beans have modified the transaction document) the corresponding row values in the business_transaction table will get updated by the Analyzer framework.

4.6.4. Writing a Custom Classification Bean

A classification bean has to implement the classify interface described above and can trigger the update of business transaction attributes by modifying the XMLTransaction object (the business transaction for the current event), which gets passed into the call. The bean has access to all XMLDocument values in the current event and the corresponding business transaction object by using the method `getDocumentValue(String xpath)`; for example:

```
String progName =
event.getDocumentValue(XpathConstants.PROGRAM_NAME);
String oldLabel = txn.getDocumentValue(XMLTransaction.LABEL_XPATH);
```

The bean can set and modify all of the additional custom transaction attributes, and most of the standard ones. The exceptions are business_trans_id, update_id, timerule_state, and events_stored. Updating those values is not allowed and may lead to unexpected results in the Analyzer. The update of transaction attributes is done by using the method setDocumentValue(String xpath, String value); for example

```
tnx.setDocumentValue(XMLTransaction.LABEL_XPATH, newLabel);
```

If the bean has modified any of the transaction attributes, it has to return a boolean true value from the classify call; otherwise, the new values will not be written to the database in the Analyzer framework.

If the transaction document remains unchanged, the bean should return false to avoid unnecessary database write overhead.

To classify a certain transaction, the bean has to update the class_id attribute of the transaction document (XMLTransaction.CLASS_ID_XPATH). This integer value is a foreign key into the CLASSIFICATION table and thus should only contain values that correspond to valid transaction class entries. The transaction class Ids can easily be accessed by using the utility class TransactionClassCache:

```
int classId =
TransactionClassCache.instance(schema).getClassId(conInfo,
className);
```

The utility class reads the transaction class data only once at initialization time from the database and returns all Ids without any further database access.

4.6.5. Logging SLA Violations

When a transaction gets classified, the analyzer can monitor its response time against the SLA value defined for the corresponding transaction class, and fire an alert in case the SLA

is violated. The SLA violation logging can be enabled by removing the comment around the LogSLAViolationCtx section in **Beans.xml** and by placing the appropriate logging bean (standard or custom logging bean) into it.

TransactionVision ships with a standard logging bean, **com.bristol.tvision. .services.analysis.eventanalysis.LogSLAViolationBean**, which logs the transaction together with its SLA and response time to the SLAViolationLog defined in **Analyzer.Logging.xml**.

If you write a custom logging class, it must implement the **com.bristol.tvision.services.analysis.eventanalysis.ILogSLAViolation interface**:

```
public boolean slaViolation(XMLTransaction txn, ConnectionInfo conInfo);
```

For the normal analyzer processing mode, the return value of this method is ignored. In failure mode, the return value indicates to the analyzer whether to write the whole business transaction to the database (return **true**) or to discard it (return **false**). The custom class needs to be added to the Analyzer's CLASSPATH by running the **TVisionSetupInfo** utility again.

4.6.6. Custom Event Correlation

There are two ways to establish relationships between either two user events or a user event and a standard Sensor event:

1. Implement the correlation logic through a Java bean that implements the interface com.bristol.tvision.services.analysis.eventanalysis.IEventCorrelation. Install this bean as the UserCorelationBean for the CorrelationTechHelperCtx in the analyzer configuration file **<TVISION_HOME>/config/services/Beans.xml**:

   ```
   (extracted from <TVISION_HOME>/config/services/Beans.xml)
   <Module name="CorrelationTechHelperCtx" type="Context">
   <Attribute name="UserCorrelationBean"
          value="com.bristol.tvision.extension.MyCorrelationBean/>
   ```

2. TransactionVision supports an XML rule engine for event correlation purposes **(com.bristol.tvision.services.analysis.eventanalysis.XMLRuleCorrelationBean**). This is similar to the rule engine for transaction classification. The custom correlation logic is implemented through XML syntax rules that are stored in the configuration file **<TVISION_HOME>/config/services/EventCorrelationDefinition.xml**. For each event (Sensor or user), it will evaluate the correlation rules against the event, create correlation lookup key(s) and event relation(s) according to the matched rules. The bean will also take care of updating the memory cache and database tables for the entities created.

   The rule engine bean can be enabled by modifying the `Beans.xml` file as follows:

   ```
   (extracted from <TVISION_HOME>/config/services/Beans.xml)
   <Module name="CorrelationTechHelperCtx" type="Context">
     <Attribute name="UserCorrelationBean" value="com.bristol.
   tvision.services.analysis.eventanalysis.XMLRuleCorrelationBean"
   />
   ```

Event Correlation Using the XML Rule File

The event correlation rules follow the same syntax as the transaction classification rules. Refer to the transaction classification section in Chapter 3 for a detailed description on the

rule basics. This section covers the details specific to the event correlation rule engine. For an example of the rules, see **<TVISION_HOME>/config/services/EventCorrelationDefinition.xml**.

The high level framework for the correlation rules is as follows:

```
<EventCorrelationDefinition>
<RelationLookupType id=1001" name="JMSToUserEvent"
dbschema="BROKER">
  <CreateLookupKey technology="UserEvent" id="1">
     . . .
  </CreateLookupKey>
  . . .
  <CreateRelation keyRuleId1="1" keyRuleId2="2" id="1">
    . . .
  </CreateRelation>
</RelationLookupType>
</EventCorrelationDefinition>
```

RelationLookupType

This element defines a relation type. It takes three attributes that characterizes the lookup type:

Attributes:

Name	Type	Use	Description
id	xsd:int	required	The relation lookup type ID. This ID should be unique in the type definition scope. The type ID should have a value greater than 1000.
name	xsd:string	required	Relation lookup type name.
dbschema	xsd:string	optional	A string representing the database schema. The presence of this attribute limits the relation lookup type scope to the particular database schema.

This element can have two types of child elements: CreateLookupKey and CreateRelation. The former implements a single rule set for creating lookup keys from individual event specific for this relation lookup type. The latter implements a single rule set for creating relation entity between two events that obey the matching conditions specified.

- CreateLookupKey

  This element defines a set of rules for creating a lookup key for the relation type this element belongs to. The following illustrates the structure of this element and its children:

```
<CreateLookupKey technology="UserEvent" id="1">
<Match xpath="/Event/StdHeader/ProgramName" operator="EQUAL"
value="Validate"/>
<Match xpath="/Event/Technology/UserEvent/Class" operator="EQUAL"
value="JDBC"/>
<Attribute name="LookupKey">
  <Path>/RelationLookup/LookupKey</Path>
  <ValueRule name="SetLookupKey">
    <Value type="XPath">/Event/Data/Chunk/Order/OrderID</Value>
  </ValueRule>
```

```
</Attribute>
</CreateLookupKey>
```

Attributes:

Name	Type	Use	Description
technology	xsd:string	required	String representing a technology name. This must be one of the technologies supported by TransactionVision. Only events belonging to the specified technology will be evaluated against this rule.
Id	xsd:int	required	An integer uniquely identifying this CreateLookupKey rule among all belonging to the same RelationLookupType object. This ID can be used in the relation creation stage to identify events that have lookup keys created based on this rule.

The following is a list of supported technology names to be used for reference in TransactionVision configuration or definition files (for example, in XML event correlation definition):

1. BTTRACE for application tracing library for WebSphere MQ
2. MQSERIES for WebSphere MQ
3. MQIMSBRIDGE for WebSphere MQ IMS bridge
4. Servlet for J2EE Servlet
5. JSP for J2EE JSP
6. JMS for J2EE Java Message Service
7. EJB for J2EE Enterprise Java Beans
8. CICS for IBM CICS
9. UserEvent for TransactionVision User Event

Match

There can be one or more match conditions. All the conditions must be met (AND) for a proper event match.

Attribute LookupKey

There should be exactly one Attribute element with the name **LookupKey** and path **/RelationLookup/LookupKey**, as shown in the above example. There can be one or more ValueRule elements with optional match conditions for assigning the lookup key value based on the event contents.

In the above example, the lookup key value is extracted from the event document under the path **/Event/Data/Chunk/Order/OrderID**.

- CreateRelation

  This element implements a rule for creating a relation between two events having the same lookup key. This element has two attributes “keyRuleId1” and “keyRuleId2”. These attributes refer to the CreateLookupKey id attribute:

Attributes:

Name	Type	Use	Description
keyRuleId1	xsd:int	required	The source event of this relation object should have its lookup key generated by the CreateLookupKey element with id equals to the value of this attribute.
keyRuleId2	xsd:int	required	The destination event of this relation object should have its lookup key generated by the CreateLookupKey element with id equals to the value of this attribute.
id	xsd:int	required	An integer ID for this CreateRelation element.

The following illustrates the structure of this element and its children:

```
<CreateRelation keyRuleId1="3" keyRuleId2="5" id="1">
  <Attribute name="RelationType">
    <Path>/EventRelation/RelationType</Path>
      <ValueRule name="SetRelationType">
        <Value type="Constant">18</Value>
      </ValueRule>
  </Attribute>
  <Attribute name="Direction">
    <Path>/EventRelation/Direction</Path>
    <ValueRule name="SetDirection">
      <Value type="Constant">2</Value>
    </ValueRule>
  </Attribute>
  <Attribute name="Confidence">
    <Path>/EventRelation/Confidence</Path>
    <ValueRule name="SetConfidence">
      <Value type="Constant">1</Value>
    </ValueRule>
  </Attribute>
</CreateLookupKey>
```

This example says that a relation is to be created between event 1 (source) and 2 (destination) if the following conditions are met:

1. Event 1 and 2 has the same lookup key value for this relation type.
2. Event 1's lookup key for this relation type is created under the CreateLookupKey rule with id equals to 3.
3. Event 2's lookup key for this relation type is created under the CreateLookupKey rule with id equals to 5

The CreateRelation element should always have the three child Attribute elements as shown above:

- The RelationType element should always have the value 17 or 18. 17 indicates a message path (suitable for representing message oriented middleware activities) while 18 indicates general purpose transaction control flow.
- The Direction element defines the relation direction, and should have value equals to 0 (unknown), 1 (inbound, flow from destination to source event), or 2 (outbound, flow from source to destination event).
- The Confidence element indicates whether the relation is strong (value = 1) or weak (value = 0). In general, the relation confidence should be set to strong (1).

Time-Based Correlation

In the area of event correlation, there are certain scenarios where perfect correlation data is not available. While there may be enough correlation information--based on either standard technology context or customer specific business data--to triage the events and limit the matching event candidates to a minimum set, additional factors need to be considered to complete the correlation process and result in one-to-one event relationship.

One such factor is event time stamps. In certain situations, TransactionVision can correlate specific event candidates by considering the respective event execution time. One example: two events A and B representing EJB X and Y method calls respectively are reported to TransactionVision, with EJB X method invoking the EJB Y method. In this case TransactionVision can reasonably assume that the two events' exit timestamps are very close based on the call latency nature. Thus TransactionVision can deduce that by truncating the respective event time stamps to a precision consistent with the expected latency, the two events would have the same truncated timestamp, and this can in turn be considered as a matching key in a lookup key based correlation algorithm.

TransactionVision provides a correlation component that supports this type of time based correlation enhancement. One can consider this as a correlation algorithm based on a mix of time and payload/technology specific data keys.

```
<RelationLookupType id="1001" name="Time_Correlation" >
<CreateLookupKey technology="MQSERIES" id="101" timeInterval="10">
<Match xpath="/Event/StdHeader/ProgramName" operator="EQUAL"
value="amqsput"/>
<Match xpath="/Event/Technology/MQSeries/@API" operator="EQUAL"
value="MQPUT"/
<Attribute name="LookupKey">
<Path>/RelationLookup/LookupKey</Path>
<ValueRule name="SetLookupKey">
<Value type="Constant">0</Value>
     </ValueRule>
               </Attribute>
</CreateLookupKey>
[…]
<CreateRelation keyRuleId1="101" keyRuleId2="101" id="1">
 […]
 </CreateRelation>
</RelationLookupType>
```

This will correlate all events which happen within a 10 minute time interval into one business transaction, provided they match the other conditions (if present, this is optional).

Note that in order for this to work, different transactions have to be separated in time by at least the same interval length.

Event Correlation Using a Custom Bean

For event correlation, the class CorrelationTechHelperCtx defines the top-level context for managing all event correlation beans. These beans are managed into different groups according to the technology categories the beans are associated with. Each category is managed by a technology specific event correlation context. Each context is designated to handle a particular type of technology (e.g.: WebSphere MQ). That is, all the events being passed to the context belong to the same technology. The technology specific context itself

holds a set of correlation beans which implements the Interface IEventCorrelation, each is responsible for correlating the current technology to one particular other technology.

In Addition to these technology specific contexts it is possible to plug in a custom 'UserCorrelationBean', which will be invoked for every event processed by the event analysis service, irrespectively of the technology.

The following is an example of event correlation context definition in the Beans.xml file:

```
<Module type="Context" name="CorrelationTechHelperCtx">

<!-- This context contains beans that perform event correlation. -->
<!-- For each event the correlation context that matches the event's
technology will be called. -->


<!-- This context contains beans that perform MQSeries event
correlation -->
<Module type="Context" name="CorrelationMQHelperCtx"
class="com.bristol.tvision.services.analysis.eventanalysis.Correlati
onMQHelperCtx">

<!—- This bean is provided by TransactionVision for establishing
default intra MQSeries event correlation such as MQPUT – MQGET
message path relations -->

<Module type="Bean"
class="com.bristol.tvision.services.analysis.eventanalysis.MQToMQRel
ationshipBean"/>


<!—- This bean is provided by TransactionVision for establishing
MQSeries – IMSBridge message path relations -->

<Module type="Bean"
class="com.bristol.tvision.services.analysis.eventanalysis.MQToBridg
eRelationshipBean"/>

<!—- This bean is developed specifically the stock trade simulation
for establishing a custom transaction path relation between a failed
MQGET call and the MQPUT call issued by the stock trade initiating
program -->

<Module type="Bean" class="com.bristol.tvision.demo.
stock.StockTradeRelationshipBean"/>

<!—The CorrelationTechHelperCtx provides a hook for the user to plug
in a technology independent custom correlation bean:
<!-- UserCorrelationBean :
1.) the 'createLookupKeys()' method of the user bean is called after
the default lookup key generation for events of all technologies and
can add additional lookup keys
2.) the 'correlateEvents()' method of the user bean is called after
the default correlation for events of all technologies and can
generate additional event relations -->

<Attribute name="UserCorrelationBean"
value="com.bristol.tvision.services.analysis.eventanalysis.UserCorre
lationBean"/ -->

</Module>
```

```
</Module>
```

For WebSphere MQ, TransactionVision provides a bean MQToMQRelationshipBean that handles all WebSphere MQ correlation tasks. This includes matching MQPUT or MQPUT1 calls to MQGET calls that handle the same message. The resultant relation is known as the message path relation, indicating a data flow between the two corresponding applications.

It is possible to add additional correlation logic in several ways:

- A new correlation bean can be developed and added to the correlation processing chain. In the above example, the StockTradeRelationshipBean bean is invoked in the MQSeries event context along with the MQToMQRelationshipBean.
- The default correlation bean can be replaced by a user bean through subclassing or aggregation. This allows modifications to the default correlation behavior. For example, a bean can be developed that invokes the MQToMQRelationshipBean correlation interfaces, examines the correlation results, and makes modifications to the results if necessary.
- Provide an implementation for the UserCorrelationBean.

An event correlation bean should implement the interface IEventCorrelation. The IEventCorrelation interface defines two methods createLookupKeys and correlate for the two phases discussed before. The class CorrelationTechHelperBean serves as the base class for all event correlation beans

In TransactionVision, event correlation is performed on a per event, per technology basis. The correlation task is divided into two phases.

The first phase involves generating lookup keys based on the characteristics of the current event. The purpose of setting up these keys is to identify the set of events bearing the same lookup key as the potential candidates for correlation in the second phase. For example, in the case of MQPUT(1) – MQGET message path relation generation, for each MQPUT(1) and MQGET event, a key composed of the message ID (MQMD.MsgId), correlation ID (MQMD.CorrelId), message put data and time is generated.

For any event, the createLookupKeys() method of each bean contained in the technology specific context will be called. In the above example, for a MQ event the MQToMQRelationshipBean as well as the MSToBridge RelationshipBean will both generate a lookup key for the current event.

The second phase involves relation generation. Specifically, a set of events is passed as potential candidate for matching with the current event. This set is composed of the events that have the same lookup key as the current event. For example, for a MQGET event, all the MQPUT(1) /MQGET events having the same key (message Id + correlation ID + message put data + message put time) are passed as potential match candidates. Further tests can now be conducted on individual candidate event to see if it is truly related to the current event. For example, events with the same method/API name (MQPUT-MQPUT, MQGET-MQGET) should not result in a message path relation.

For a certain set of candidates with matching lookup keys, the type of the correlation (e.g., MQ-MQ or MQ-IMS) determines which beans correlateEvents() method is called. In the above example, a set of events with matching lookup key of type MQ-MQ will be passed on to the MQToMQRelationshipBean, a set of events with type MQ-IMS will be passed on to the MQToBridgeRelationshipBean. Currently the following correlation types are defined for TransactionVision as constants in class EventCorrelationBean:

```
public class EventCorrelationBean extends AnalyzeEventBean {

  public static final int MQ_PUT_GET_TYPE           = 1;
  public static final int MQ_IMSBRIDGE_TYPE         = 2;
  public static final int IMSBRIDGE_ENTRY_EXIT_TYPE = 3;
  public static final int JMS_SEND_RCV_TYPE         = 4;
  public static final int PROXY_TYPE                = 5;
  public static final int PUBSUB_TYPE               = 6;
  public static final int CICS_TRANS_TYPE           = 7;
  public static final int MQ_CICS_TYPE              = 8;
. . .
}
```

The correlation type for a correlation bean has to provided in the constructor call. For user defined correlation beans, new correlation types should be >= 100.

4.6.7. Interface IEventCorrelation

package `com.bristol.tvision.util.services.analysis.eventanalysis`
public interface IEventCorrelation

The IEventCorrelation interface defines the methods to be implemented by any event correlation bean.

Methods:

- createLookupKeys

```
public void createLookupKeys(ConnectionInfo conInfo, XMLEvent event,
java.awt.List lookupKeys) throws AnalyzeEventException
```

  Generate one or more lookup keys for correlation purpose for the given event.

  Parameters:

  conInfo - database connection info object for the current project

  event - completed XML document for the current event

  lookupKeys - list of lookup keys to be added

  Throws:

  `AnalyzeEventException` - Signals errors during the event correlation analysis

- correlateEvents

```
public void correlateEvents (ConnectionInfo conInfo,
TechEventID id, TechEventID idToMatch, int correlationType, List
eventRelations)
throws AnalyzeEventException
```

  Decide whether a relation should be established between the two events passed. If the conclusion is affirmative, generate new relation objects and add them to the given list.

  Parameters:

conInfo - database connection info object for the current project

id - event ID object for the current event to be matched

idToMatch - event ID object for the potential matching event candidate

correlationType - the correlation type

eventRelations - list of event relations generated

Throws:

AnalyzeEventException - Signals errors during the event correlation analysis

4.6.8. Class CorrelationTechHelperBean

```
package com.bristol.tvision.util.services.analysis.eventanalysis
public abstract class CorrelationTechHelperBean
extends ChainManagedBean
implements IEventCorrelation
```

This is the abstract base class for all event correlation beans.

Constructor:

- CorrelationTechHelperBean

```
CorrelationTechHelperBean(java.lang.String technology, int
correlationType) throws AnalyzeEventException
```

  Creates an instance of this event correlation bean for the given technology and correlation type. The correlation type is a unique integer and should be >= 100 for new user-defined correlation types.

Methods:

- createLookupKeys

  Refer to the definition of IEventCorrelation.

- correlateEvents

  Refer to the definition of IEventCorrelation.

- getCorrelationType

```
public int getCorrelationType()
```

  Return the correlation type string.

- Class MQCorrelationData

```
package com.bristol.tvision.datamgr.dbtypes
public class MQCorrelationData
```

This class defines a collection of event attributes relevant to the event correlation process. For example, in the IEventCorrelation::correlateEvents method, event attributes for the two events to be matched can be retrieved through a correlation data cache. The attributes are returned in an object instance of this class.

Constructor:

MQCorrelationData

```
MQCorrelationData(int apiCode, java.lang.String putApplName,
java.lang.String putApplType,String userId, long qmgrId, long
mqObjId, java.lang.String eventTime, long programId)
```

```
MQCorrelationData(int apiCode, java.lang.String putApplName,
java.lang.String putApplType,String userId, long qmgrId, long
mqObjId, java.lang.String eventTime, long programId,
java.lang.String jobNameId, java.lang.String jobStepId,
java.lang.String sysId, java.lang.String transId,
java.lang.String imsId, java.lang.String imsRegionType,
java.lang.String imsRegionId, long imsTxnId,
java.lang.String imsPsbId)
```

Creates an instance of a WebSphere MQ correlation event attribute data collection object based on the given event attributes.

Fields:

- int apiCode
- String putApplName
- String putApplType
- String userId
- long qmgrId
- long mqObjId
- String eventTime
- long programId

4.6.9. Class JMSCorrelationData

```
package com.bristol.tvision.datamgr.dbtypes
public class JMSCorrelationData
```

Similar to the class MQCorrelationData, this class defines a collection of event attributes relevant to the event correlation process of JMS events.

Constructor:

JMSCorrelationData

JMSCorrelationData(int methodCode, String appId, String userId, String destination, String eventTime, long programId, String putApplType, long qmgrId, long mqObjId)

Creates an instance of a JMS correlation event attribute data collection object based on the given event attributes.

Fields:

- int methodCode
- String appId
- String userId
- String destination
- String eventTime
- long programId
- String putApplType
- long qmgrId
- long mqObjid

4.6.10. Class LookupKey

```
package com.bristol.tvision.datamgr.dbtypes
public class LookupKey
```

This class defines the lookup key object to be used in identifying potential events for correlation purpose.

Constructor:

LookupKey

```
LookupKey(java.lang.String keyValue, int typeId)
```

Creates a new lookup key instance with the given key and the correlation type id.

Fields:

- String keyValue
- int typeId

Methods:

- equals

```
public boolean equals (LookupKey lookupKey)
```

Determine whether two LookupKey objects are equivalent.

Overrides:

equals in class java.lang.Object

Decide whether the given lookupKey is equal to this key object. The two objects are equal if the corresponding key, correlation type string, and type ID are the same.

Parameters:

lookupKey - lookup key object to be compared

Returns:

true if the two keys are equal, false otherwise

4.6.11. Class EventRelation

```
package com.bristol.tvision.datamgr.dbtypes
public class EventRelation
```

This class defines an event relation object between any two events.

Fields:

Relation Type

public static final int UNKNOWN_PATH = 0**;**

public static final int MESSAGE_PATH = 1**;**

public static final int TRANSACTION_PATH = 2**;**

public static final int BIDIRECTION = 16

Type of the event relation:

- MESSAGE_PATH indicates a direct message flow between the two events. That means the two events are associated with the same message data. For example, a MQPUT and MQGET call dealing with the same message bears a message path relation.

- TRANSACTION_PATH indicates a control flow between two events.
- BIDIRECTION is a type mask that indicates the bi-direction nature of the relation between the two events.

Relation Direction

public static final int RELATION_PATH_IN = 1**;**

public static final int RELATION_PATH_OUT = 2**;**

public static final int RELATION_UNKNOWN = 0**;**

Direction of the event relation. Note that the event object is created in conjunction with an event pair (event1, event2). This indicates the direction from event1 to event2.

Confidence Factor

public static final int WEAK_RELATION = 0**;**

public static final int STRONG_RELATION = 1**;**

This factor is assigned by the event correlation module. There are cases where the correlation module may not have perfect data for a deterministic decision on the event relation generated. In such case, the relation created can carry a WEAK_RELATION confidence factor indicating the uncertainty in the decision.

int relation

Bitfield indicating the relation type, e.g. MESSAGE_PATH | BIDIRECTION

int direction

Bitfield indicating the relation direction, e.g. RELATION_PATH_IN | RELATION_PATH_OUT

int confidence

Confidence factor, either WAEK_RELATION or STRONG_RELATION

int latency

The latency between the two events in milliseconds

Constructor:

EventRelation

```
EventRelation(int relation, int direction, int confidence, int
latency)
```

Creates a relation object with the given relation type, direction, confidence factor, and latency.

4.6.12. Class MQRelationDBService

```
package com.bristol.tvision.datamgr.dbservices
public class MQRelationDBService
```

This class defines an internal database service for accessing MQSeries correlation related information. For example, this service works in conjunction with the caching mechanism and stores MQSeries event correlation attributes. The following describes the public interfaces of interest to the custom event analysis beans developers.

Methods:

- instance

```
public static MQRelationDBService
instance(java.lang.String schema)
```

Return the singleton instance of the MQRelationDBServices.

Parameters:

schema - Database schema for the current project

Returns:

Singleton instance of the MQRelationDBService.

- getCorrelationData

```
public MQCorrelationData getCorrelationData(java.lang.Connection
con, EventID eventID) throws DataManagerException
```

Return the MQSeries correlation event data for the given event.

Parameters:

con - Java SQL database connection handle, probably from the ConnectionInfo object.

eventID - EventID object for the interested event

Returns:

A MQCorrelationData object for the given event.

Throws:

DataManagerException - Signals errors during internal database operations.

4.6.13. Class JMSRelationDBService

```
package com.bristol.tvision.datamgr.dbservices
public class JMSRelationDBService
```

This class defines an internal database service for accessing JMS correlation related information.

Methods:

- instance

```
public static JMSRelationDBService
instance(java.lang.String schema)
```

Return the singleton instance of the JMSRelationDBServices.

Parameters:

schema - Database schema for the current project

Returns:

Singleton instance of the JMSRelationDBService.

- getCorrelationData

```
public JMSCorrelationData
getCorrelationData(java.sql.Connection con,EventID eventId)
throws DataManagerException
```

Return the MQSeries correlation event data for the given event.

Parameters:

con – Java SQL database connection handle, probably from the ConnectionInfo object.

eventID – EventID object for the interested event

Returns:

A JMSCorrelationData object for the given event.

Throws:

DataManagerException - Signals errors during internal database operations.

Sample Custom Event Correlation Bean

Refer to the code in the directory **<TVISION_HOME>/samples/stock** to see a sample implementation of a custom event correlation bean (**StockTradeRelationshipBean.java**).

StockTradeRelationshipBean implements the IEventCorrelation interface and is derived from the class CorrelationTechHelperBean. It builds a custom message path relation between a failed MQGET event (CompCode equals to MQCC_FAILED) and the MQPUT event that participates in the same trade request processing. The stock trade example follows a request-reply messaging model. The StockTrade program records the message ID field of the initial request message, and uses this value as the correlation ID value to be matched when it reads the reply message through the MQGET call. In other words, for a particular transaction, the message ID field in the MQMD object of the StockTrade – MQPUT(1) event should be the equal to the correlation ID field in the MQGET event.

The following is the code fragment for the StockTradeRelationshipBean constructor. It specifies that the bean handles MQSeries events and generates custom event relation of type “REQUEST_REPLY_TYPE” correlation as described above:

```
public static final String REQUEST_REPLY_TYPE = 100;
public StockTradeRelationshipBean() throws AnalyzeEventException {
super(TVisionCommon.TECH_NAME_MQSERIES, REQUEST_REPLY_TYPE);
}
```

The next code fragement contains the implementation of the createLookupKeys method. As discussed before, the message ID or correlation ID value in the message descriptor record is used as the lookup key for MQPUT(1) and MQGET respectively.

```
public void createLookupKeys(ConnectionInfo conInfo, XMLEvent event,
                             List lookupKeys) throws
AnalyzeEventException {
try {
XPathSearch lookup = new XPathSearch(event);
String correlId;
/* for StockTrade->MQPUT call (request event), use MQMD.MsgID as */
/* lookup key, for StockTrade->MQGET call (reply event), use */
/* MQMD.CorrelId as the lookup key */
switch (StockTradeHelper.getEventType(lookup)) {
case StockTradeHelper.MQSERIES_REQUEST_EVENT:
correlId = lookup.getValue(XPathConstants.MSGID);
if (correlId == null)
  return;
break;
case StockTradeHelper.MQSERIES_REPLY_EVENT:
if (Integer.parseInt(lookup.getValue(XPathConstants.COMPCODE)) !=
```

```
    MQDefs.MQCC_FAILED)
return;
  correlId = lookup.getValue(XPathConstants.CORRELID);
     if (correlId == null)
return;
  break;
default:
return;
 }

/* create a new lookup key and add it to the list */
LookupKey key = new LookupKey(correlId, REQUEST_REPLY_TYPE);
lookupKeys.add(key);
  }
  catch (XMLException ex) {
throw new AnalyzeEventException(ex);
  }
}
```

The next code fragment contains the implementation of the correlateEvents method:

```
public void correlateEvents(ConnectionInfo conInfo, TechEventID id,
TechEventID idToMatch, List eventRelations) throws
AnalyzeEventException {

try {
/* Retrieve data relevant for event correlation from cache. */
Cache cache = AnalysisCacheManager.instance().getCorrelationCache
(conInfo.schema);
MQCorrelationData data = (MQCorrelationData) cache.get(id);
if (data == null) {
  data =
MQRelationDBService.instance(conInfo.schema).getCorrelationData(
           conInfo.con, id);
  if (data != null)
    cache.insert(id, data);
  else
    return;
}
MQCorrelationData dataToMatch = (MQCorrelationData)
cache.get(idToMatch);
if (dataToMatch == null) {
  dataToMatch =
    MQRelationDBService.instance(conInfo.schema).getCorrelationData(
      conInfo.con, idToMatch);
  if (dataToMatch != null)
    cache.insert(idToMatch, dataToMatch);
  else
    return;
}
int apiId = data. apiCode;
int apiIdToMatch = dataToMatch.apiCode;
if (apiId != apiIdToMatch) {
  EventRelation eventRelation = new EventRelation();
  eventRelation.setRelation(EventRelation.MESSAGE_PATH |
                            EventRelation.BIDIRECTION);
  eventRelation.setDirection(EventRelation.RELATION_UNKNOWN);
  eventRelation.setConfidence(EventRelation.STRONG_RELATION);
  eventRelations.add(eventRelation);
}
}
```

```
catch (DataManagerException ex) {
  throw new AnalyzeEventException(ex);
}
}
```

The AnalysisCacheManager object provides an internal memory cache for storing selected attributes of the events to be matched. Refer to the MQCorrelationData class definition for a list of attributes supported. This cache allows quick access to certain event attributes without executing an event data query, thus improving the correlation process performance.

To decide whether the two events are indeed related, the API code of the two events are compared to ensure that one event is MQPUT(1) and the other one is MQGET. Since only MQPUT(1) and MQGET events can be potential candidates, it is enough to check whether the two event API codes are different or not.

Once it is decided that the two events are related, a new event relation object is created and inserted to the relation list. The relation is of type MESSAGE_PATH, has no direction attribute, and has a STRONG_RELATION confidence factor.

The following code fragment is the change to the Beans.xml file for including this custom event correlation bean. It tells the Analyzer framework to load and run the StockTradeCorrelationBean bean as a part of the CorrelationMQHelperCtx context.

This bean will be invoked after the default MQToMQRelationshipBean for every MQSeries event.

```
<Module type="Context" name="CorrelationTechHelperCtx">

<Module type="Context" name="CorrelationMQHelperCtx"
class="com.bristol.tvision.services.analysis.eventanalysis.Correlati
onMQHelperCtx">

<Module type="Bean"
class="com.bristol.tvision.services.analysis.eventanalysis.MQToMQRel
ationshipBean"/>


<Module type="Bean" class="com.bristol.tvision.demo.
stock.StockTradeRelationshipBean"/>

</Module>

</Module>
```

4.6.14. Custom Local Transaction Definition

Customization of the local transaction analysis algorithm in the Analyzer allows modification of the unit of work or local transaction definition for a set of events. By default, TransactionVision uses the sync-point APIs such as MQCMIT, MQBACK, etc., to group events into local transactions. However some applications may not be transactional in nature. For these applications, it may be useful to group sets of events into logical local transactions.

The local transaction rule definition file follows the same syntax as the transaction classification rules. See the “Transaction Classification” section earlier in this chapter for a detailed description on the rule basics. This section covers the details specific to the local transaction rule engine.

The basic goal of the rules defined in the **LocalTransactionDefinition.xml** file is to set local transaction attributes, if the event currently being processed matches certain criteria. These attributes, such as the LookupKey attribute, are then used by the framework to either, create a new local transaction id and assign that id to the event or find an existing local transaction that has the same attributes, and assign its local transaction id to the current event.

An example application of the LocalTransactionDefinition.xml rule file is to correlate an MQPUT of a request with an MQGET for the reply in the same process based on message id, where the MQPUT and MQGET do not exist in the same unit of work. This happens when an application puts a request, and waits for a reply with an MQGET for the same id until it times out. The request and reply will by placed in the same unit of work by the Analyzer only if the sync-point options have been used by the application. If not, the LocalTransactionDefinition.xml file may be used to generate a custom LookupKey attribute based on the message id field in the MQPUT and MQGET events.

4.6.15. LocalTransactionDefinition.xml File

This file is located in the **<TVISION_HOME>/config/services** directory. The layout of this rule file is as follows:

```
<LocalTransactionDefinition>
      <LocalTransactionType dbschema="*"
                            hasMultiTracking="false" >

            <Match xpath=". . ." operator="EQUAL" value=". . ."/>
            . . .
            <LocalTransactionAttributes>
                  <Attribute name="LookupKey">
                              <Path> . . . </Path>
                        <ValueRule name="SetLookupKey">
                              <Value type="XPath"> . . . </Value>
                              <Value type="XPath"> . . . </Value>
                        </ValueRule>
                  </Attribute>

            </LocalTransactionAttributes>

      </LocalTransactionType>

  </LocalTransactionDefinition>
```

The LocalTransactionDefinition element is the root element and only one instance of this element can exist in a definition file. Each root element can contain several LocalTransactionType elements. Each LocalTransactionType element has a dbschema attribute containing one or more schemas (comma separated) to which this rule type applies. Hence, the attributes and match criteria contained in this LocalTransactionType element only apply to events being written to the given schemas. The schema attribute can be set to “*”, or left out completely, to indicate that the rule is applicable to all schemas. A set of Match child elements determine whether the attributes specified in the LocalTransactionAttributes element should be applied to the current event. The LocalTransactionAttributes element contains a set of Attribute elements. Each attribute is set at the XPath specified in the Path element. The value for this attribute comes from the Value elements. These may be constants or XPaths into the current event document. The Attribute element may contain additional Match criteria to determine which attributes need to be set.

4.6.16. LocalTransactionType

This element defines a local transaction rule type. It takes three attributes that characterizes the lookup type:

Attributes:

Name	Type	Use	Description
dbschema	xsd:string	optional	A string representing the database schema. The presence of this attribute limits the relation lookup type scope to the particular database schema.
hasMultiTracking	xsd:boolean	optional	A boolean value, which when true indicates that the local transaction can have multiple tracking ids and the processMultiTracking() method of the ILocalTransaction interface needs to be executed.

This element can contain two kinds of child elements, multiple Match elements and one LocalTransactionAttributes element. The Match elements contain the criteria based on which attributes will be set for an event. For example:

```
  <Match xpath="/Event/StdHeader/TechName" operator="EQUAL"
value="MQSERIES"/>
  <Match xpath="/Event/StdHeader/HostArch/OS" operator="UNEQUAL"
value="OS390_CICS"/>
```

The above two Match criteria evaluate to true if the event is an MQSeries event, but not from z/OS CICS. When an event which matches these criteria is evaluated, the attribute setting rules contained in the LocalTransactionAttributes element are executed.

4.6.17. LocalTranasctionAttributes

One element of this type is required. This element holds multiple attribute elements, each defining an Attribute to be set. The LookupKey attribute containing a Path /LocalTransaction/LookupKey is required. Attribute names need to be unique for a given LocalTransactionAttributes element. There can be multiple Attribute rules with the same XPath but a different name, Match and Value rules.

For example:

```
<LocalTransactionAttributes>
  <Attribute name="LookupKey">
   <Path>/LocalTransaction/LookupKey</Path>
     <ValueRule name="SetLookupKey">
        <Value type="XPath">/Event/EventID/@programInstID</Value>
<Value type="Constant">-</Value>
        <Value type="XPath">/Event/StdHeader/@uow</Value>
      </ValueRule>
   </Attribute>
</LocalTransactionAttributes>
```

The above LocalTransactionAttributes element contains one Attribute called LookupKey. This attribute maps to the XPath /LocalTransaction/LookupKey and is set to a concatenation of three values in the Value elements. The attribute ‘LookupKey’ determines the local transaction for the current event – events with the same LookupKey will be part of the same local transaction.

Typically, for WebSphere MQ events, only the LookupKey attribute needs to be set to group events into a unit of work. However, for other events such as JMS, Servlet or EJB events, additional attributes such as TrackingId (/LocalTransaction/TrackingId), ParentTxnKey (/LocalTransaction/ParentTxnKey) and TrackingSeq (/LocalTransaction/TrackingSeq) may be set. The TrackingId attribute is used to group multiple local transactions into business transactions for the J2EE Sensors. In custom local transaction definitions, generating the same tracking id for certain events can be used to group their local transactions into the same business transaction. The ParentTxnKey and TrackingSeq attributes are primarily used by the TransactionVision Transaction Analysis view to draw links between local transactions. These attributes are reported by the Sensors and typically would not need to be customized.

4.6.18. Sample LocalTransactionDefinition.xml Rule File

The following sample rule file sets the `LookupKey` local transaction attribute to the event message id field for all events from queue TVISION.TEST.Q for all events being written to the TEST.SCHEMA. For events to any other schema besides TEST.SCHEMA, the `LookupKey` attribute is set using the default MQSeries strict algorithm to use the program instance id and unit of work ids.

```
<LocalTransactionDefinition>
   <LocalTransactionType dbschema="TEST.SCHEMA"
                         hasMultiTracking="false" >
<Match xpath="/Event/StdHeader/TechName" operator="EQUAL"
value="MQSERIES"/>
<Match xpath="/Event/Technology/MQSeries/MQObject/@objectName"
operator="EQUAL" value="TVISION.TEST.Q"/>
<LocalTransactionAttributes>
<Attribute name="LookupKey">
<Path>/LocalTransaction/LookupKey</Path>
<ValueRule name="SetLookupKey">
                       <Value
type="XPath">/Event/Technology/MQSeries/*/*Exit/MQMD/MsgId</Value>
</ValueRule>
</Attribute>
</LocalTransactionAttributes>
</LocalTransactionType>

   <LocalTransactionType dbschema="*"
                         hasMultiTracking="false" >
<Match xpath="/Event/StdHeader/TechName" operator="EQUAL"
value="MQSERIES"/>
<LocalTransactionAttributes>
<Attribute name="LookupKey">
<ValueRule name="SetLookupKey">
<Value type="XPath">/Event/EventID/@programInstID</Value>
<Value type="Constant">-</Value>
                    <Value type="XPath">/Event/StdHeader/@uow</Value>
</ValueRule>
</Attribute>
</LocalTransactionAttributes>
</LocalTransactionType>
</LocalTransactionDefinition>
```

4.6.19. Changes to the Beans.xml File

To enable usage of the LocalTransactionDefinition.xml rules file, the **<TVISION_HOME>/config/services/Beans.xml** file must be modified to enable use of the rules bean. The following changes are required to the Beans.xml file:

```
<Module type="Context" name="LocalTransactionTechHelperCtx">
        . . .
        <Module type="Bean"
class="com.bristol.tvision.services.analysis.eventanalysis.MQLocalTr
ansactionBean">
             <Attribute name="AlgorithmBean"
value="com.bristol.tvision.services.analysis.eventanalysis.XMLRuleLo
calTransactionBean"/>
             <!--Attribute name="AlgorithmBean"
value="com.bristol.tvision.services.analysis.eventanalysis.MQStrictL
ocalTransaction"/ -->
             <!-- Attribute name="AlgorithmBean"
value="com.bristol.tvision.services.analysis.eventanalysis.MQDefault
LocalTransaction"/ -->
        </Module>
   . . .
```

The same needs to be repeated for the corresponding technology where the rule bean needs to be applied.

Local transaction analysis algorithm beans can be chained by placing multiple bean names in the **Beans.xml** file as below:

```
<Module type="Bean"
class="com.bristol.tvision.services.analysis.eventanalysis.
MQLocalTransactionBean">
        <Attribute name="AlgorithmBean"
value="com.bristol.tvision.services.
analysis.eventanalysis.XMLRuleLocalTransactionBean"/>
        <Attribute name="AlgorithmBean"
value="com.bristol.tvision.services.
analysis.eventanalysis.MQStrictLocalTransaction"/>
</Module>
```

The local transaction beans are initialized and invoked in the sequence they are placed in the Beans.xml file. For example, in the above snippet the XMLRuleLocalTransactionBean rules will be executed before the MQStrictLocalTransaction getAttributes() method is invoked. By default, the chain of invocation is broken and subsequent beans are NOT called when a bean's getAttribute() method returns a non-null lookup key. Hence, in the above example, the MQStrictLocalTransaction bean is invoked only when there is no matching rule set in the **LocalTransactionDefinition.xml** file which create a non-null lookup key. Note: it is important to place the XMLRuleLocalTransactionBean before any standard beans if it is intended to replace the generated default lookup key.In some scenarios, it may be desired that certain events do not have a local transaction id. To do this, create a rule that sets the return key value as a constant NULL.

The following example rule does not create a local transaction id for all events from queue TVISION.TEST.Q, by setting the LookupKey attribute to a constant NULL value.

```
<LocalTransactionType dbschema="*"  hasMultiTracking="false" >
    <Match xpath="/Event/StdHeader/TechName" operator="EQUAL"
value="MQSERIES"/>
    <Match xpath="/Event/Technology/MQSeries/MQObject/@objectName"
operator="EQUAL" value="TVISION.TEST.Q"/>
    <LocalTransactionAttributes>
            <Attribute name="LookupKey">
```

```
<Path>/LocalTransaction/LookupKey</Path>
<ValueRule name="SetLookupKey">
                                <Value type="Constant">NULL</Value>
                        </ValueRule>
                </Attribute>
    </LocalTransactionAttributes>
</LocalTransactionType>
```

4.7. Extending the System Model

Use the **<TVISION_HOME>/config/services/RemoteDefinition.xml** file to define objects in your system that the agent might otherwise not be able to fully resolve.

For example, suppose you have a remote queue on queue manager QM1 that points to some queue on queue manager QM2. A sensored application putting to the queue on QM1 does not connect to QM2 to fully discover what type of object the final destination queue is. The destination queue might be an alias queue or even another remote queue. If no sensored application on QM2 ever connects directly to the destination of the QM1 remote queue, then the object will never be fully resolved, possibly resulting in a missing link in the correlation of events.

By manually defining objects in RemoteDefinition.xml, you can specify the details of objects that the agent could not completely resolve otherwise.

Each <RemoteObject> tag defines an object. When the analyzer attempts to resolve the target of a remote queue, it checks whether an entry exists with the same object and queue manager name. If such a match is found, the MQObject definitions within the RemoteObject tag will replace the generic queue definition provided by the agent. Embedding an additional MQObject tag within the first MQObject tag creates a "resolveto" relationship.

Therefore, the first RemoteObject tag in the following example can be interpreted as: If the destination of a remote queue has the name RALIAS2.QUEUE on queue manager perplex7.tv2.manager, create for this object an alias queue RALIAS2.QUEUE that resolves to a local queue RRR.QUEUE.

Possible values for the objectType attribute include:

- Q_LOCAL
- Q_MODEL
- Q_ALIAS
- Q_REMOTE
- Q_CLUSTER
- Q_LOCAL_CLUSTER
- Q_ALIAS_CLUSTER
- Q_REMOTE_CLUSTER

Take care in creating and modifying these definitions as inserting objects that don't actually match the topology of your system could break the correlation of events.

Example RemoteDefinition.xml file:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<RemoteDefinition>
   <RemoteObject objectName="RALIAS2.QUEUE"
queueManager="perplex7.tv2.manager">
         <MQObject objectName="RALIAS2.QUEUE" objectType="Q_ALIAS"
queueManager="perplex7.tv2.manager">
               <MQObject objectName="RRR.QUEUE" objectType="Q_LOCAL"
queueManager="perplex7.tv2.manager"/>
         </MQObject>
   </RemoteObject->

   <RemoteObject objectName="TEST.CLUSTER.QUEUE"
queueManager="SECOND_CLUSTER">
   <MQObject objectName="TEST.CLUSTER.QUEUE" objectType="Q_REMOTE"
queueManager="SECOND_CLUSTER">
         <MQObject clusterName="SECOND_CLUSTER"
objectName="TEST.CLUSTER.QUEUE" objectType="Q_LOCAL_CLUSTER"
queueManager="deepakelap.tv3.manager"/>
   </MQObject>


   </RemoteObject>
</RemoteDefinition>
```

4.7.1. User Events

Each user event can optionally carry data about system resource objects involved in the event. The user defined types have type ID greater than the value com.bristol.tvision.userevents.Constants.USEROBJECT_TYPE_BASE.

On the Analyzer side, all user object types should be included in a central configuration file **<TVISION_HOME>/config/sysmodel/SystemModelDefinition.xml**. Both the Analyzer and Web components read this configuration file, and use the information for runtime object type validation.

The following is an example of this file:

```
<?xml version="1.0" encoding="UTF-8"?>
<SystemModelDefinition>
   <ObjectClass name="JDBC" base="100000">
         <ObjectType name="DatabaseServer" id="1"/>
   </ObjectClass>
   <ObjectClass name="FTP" base="101000">
         <ObjectType name="FTPServer" id="1"/>
   </ObjectClass>
</SystemModelDefinition>
```

- User object types should be grouped under various object type classes. Each class is defined under the element /SystemModelDefinition/ObjectClass. In the example, two classes are defined for database and FTP technology objects respectively.
- Each object type class should have a string attribute “name” and integer attribute “base”, which defines the base for the type ID for all objects in the class.
- The element /SystemModelDefinition/ObjectClass/ObjectType defines a single object type. It has a string attribute “name” for the object type name, and an integer attribute “id”. The id attribute, combining with the object type class ID base, forms the final type ID for the object type. In this example, the object type “DatabaseServer” has type ID

100001 (100000 + 1), and the object type “FTPServer” has type ID 101001 (101000 + 1).

It is important to ensure that the object type ID values used by the user events are consistent with the ones from the central configuration file.

4.8. Generating Application Events to Tivoli Enterprise Console (TEC)

TransactionVision allows plugging in custom code to generate TEC events when certain application events occur. These can either be plugin beans into the Analyzer or as scheduled jobs running in the application server hosting the UI. This custom code can use the log4j classes to generate log messages. The log4j appender, TECAppender, routes these log messages to Tivoli, when enabled. The MonitoringEvent class is provided to allow setting of parameters into the log4j message, which are then mapped to Tivoli slots by the TECAppender. The TECAppender uses the file **<TVISION_HOME>/config/logging/tivoli/SlotMap.properties** to map MonitoringEvent parameters to Tivoli slots.

4.8.1. Monitoring Events

The class com.bristol.tvision.util.log.MonitoringEvent implements the monitoring event structure. TransactionVision defines various monitoring events reporting the analyzer and web component runtime states. Custom monitoring events based on business data can also be constructed with this class.

Class com.bristol.tvision.util.log.MonitoringEvent

```
package com.bristol.tvision.util.log;

public class MonitoringEvent implements Cloneable,
java.io.Serializable {

   . . .
}
```

Attributes and Access Methods

Event Source Components

This refers to a string identifying the main event source component. TransactionVision defines several standard values in the class com.bristol.tvision.util.TVisionCommon:

- TVisionCommon.COMP_ANALYZER: Analyzer components
- TVisionCommon.COMP_JOB: Job bean
- TVisionCommon.COMP_UI: TransactionVision UI/Job Server components
- TVisionCommon.COMP_OTHERS: Other event sources.

Access methods:

```
public void setSourceComponent(String srcComp);
public String getSourceComponent();
```

Event Source Sub-components

This refers to a string identifying the event source subcomponent. Usually this refers to the Java class name (e.g.: com.bristol.tvision.services.analysis.action.LogSLAViolation).

Access methods:

```
public void setSourceSubComponent(String subSrcComp);
public String getSourceSubComponent();
```

Event Class

This refers to a string identifying the event class. TransactionVision defines the following standard values:

- MonitoringEvent.CLASS_INTERNAL: Analyzer and web components internal monitoring events.
- MonitoringEvent.CLASS_APPLICATION: Application specific monitoring events.
- MonitoringEvent.CLASS_CEP_SITUATION: Complex event processor situation events.

Access methods:

```
public void setClassName(String className);
public String getClassName();
```

Event Type

This refers to a string describing the monitoring event type. All TransactionVision standard values for internal monitoring events have the prefix “TVision”. Any custom defined monitoring event should have a type name consistent with the complex event processor event definitions.

Access methods:

```
public void setTypeName(String typeName);
public String getTypeName();
```

Priority

This refers to an integer value reflecting the priority of the monitoring events. The following is a list of standard values:

- MonitoringEvent.PRIORITY_HIGH = 70
- MonitoringEvent.PRIORITY_MEDIUM = 50
- MonitoringEvent.PRIORITY_LOW = 10
- MonitoringEvent.PRIORITY_UNKNOWN = 0

Access methods:

```
public void setPriority(int priority);
public int getPriority();
```

Severity

This refers to an integer value reflecting the severity of the monitoring events. The following is a list of standard values:

- MonitoringEvent.SEVERITY_FATAL= 60
- MonitoringEvent.SEVERITY_ERROR = 50
- MonitoringEvent.SEVERITY_MINOR = 40
- MonitoringEvent.SEVERITY_WARNING = 30
- MonitoringEvent.SEVERITY_HARMLESS = 20
- MonitoringEvent.SEVERITY_INFORMATION = 10
- MonitoringEvent.SEVERITY_UNKNOWN = 0

Access methods:

```
public void setSeverity(int severity);
public int getSeverity();
```

Message

This refers to a string containing the description for the monitoring event. By default this is set to an empty string.

Access methods:

```
public void setMessage(String msg);
public String getMessage();
```

Time

This refers to a long value representing the event time in milliseconds (specifically, the difference, measured in milliseconds, between the current time and midnight, January 1, 1970 UTC). The constructors for this class automatically set this to the current time.

Access methods:

```
public void setTime(long eventTime);
public long getTime();
```

Parameters

Additional parameters for the monitoring event are stored as name-value pair in an internal hash map in the MonitoringEvent class. The name part has to be a Java string, while the value part can be any Java serializable objects. Use the following access methods to get and set additional parameters.

Access methods:

```
public void setParameter(String name, Object value);
public Object getParameter(String name);
public HashMap getParameters(); (return parameter hash map)
public Set getEntries(); (return parameter hash map entry set)
```

Constructors

```
public MonitoringEvent()
```

The default constructor only initializes the monitoring event time. All other attributes are set to the default values. The attribute access methods should be used to set the required attributes.

```
public MonitoringEvent(String srcComp, String srcSubComp, String
eventClass, String eventType, int severity)
```

This constructor initializes the monitoring event with the given source component, subcomponent, event class, event type, and severity level. Event time is by default set to the current time.

Clone

```
public Object clone();
```

This method returns a shallow clone copy of the given MonitoringEvent object. The parameter hash map contents are indeed copied over.

Serialization

The toString() method can serialize the MonitoringEvent contents in two forms:

1. By default, the message attribute will be returned by the toString() method (Message Serialization).

2. The complete event content can be serialized in XML format (XML Serialization).

The serialization behavior can be toggled by the following two methods:

```
public void enableMessageSerialization
public void enableXMLSerialization
```

Helper functions

```
public static String getHostName()
```

This method returns the local host name.

4.8.2. Event Delivery

This section describes the steps for implementing monitoring event delivery.

The class com.bristol.tvision.util.log.Logging supports various methods for delivering the monitoring events through Log4J. Any log4j initialization is taken care of by the TransactionVision components. TransactionVision implements the following Logger (Category) objects in various components:

Analyzer:

1. AppLog: for reporting Analyzer errors, warning, and information type messages. This is the default logger object in the Logging class. One can invoke any of the following four Log4J category logging methods directly through this class:

```
Logging.fatal(MonitoringEvent eventObj)
Logging.error(MonitoringEvent eventObj)
Logging.warn(MonitoringEvent eventObj)
Logging.info(MonitoringEvent eventObj)
```

2. AnalyzerActivityLog: for internal Analyzer activity logging such as start and stop operation. This can also be used for logging transaction related report such as service level agreement violation. This logger object can be accessed through the variable Logging.analyzerActivityLog:

```
Logging.analyzerActivityLog.fatal(MonitoringEvent eventObj)
Logging.analyzerActivityLog.error(MonitoringEvent eventObj)
Logging.analyzerActivityLog.warn(MonitoringEvent eventObj)
Logging.analyzerActivityLog.info(MonitoringEvent eventObj)
```

UI/Job Server components:

1. AppLog: for reporting UI component errors, warning, and information type messages. This is the default logger object in the Logging class. One can invoke any of the following four Log4J category logging methods directly through this class:

```
Logging.fatal(MonitoringEvent eventObj)
Logging.error(MonitoringEvent eventObj)
Logging.warn(MonitoringEvent eventObj)
Logging.info(MonitoringEvent eventObj)
```

2. UIActivityLog: for internal UI components activity logging such as start and stop operation. This logger object can be accessed through the variable Logging.uiActivityLog:

```
Logging.uiActivityLog.fatal(MonitoringEvent eventObj)
Logging.uiActivityLog.error(MonitoringEvent eventObj)
```

```
Logging.uiActivityLog.warn(MonitoringEvent eventObj)
Logging.uiActivityLog.info(MonitoringEvent eventObj)
```

The following code segment provides an example of logging a service level agreement violation monitoring event:

```
import com.bristol.tvision.util.TVisionCommon;
import com.bristol.tvision.util.log.Logging;
import com.bristol.tvision.util.log.MonitoringEvent;
. . . . .
public static void logViolation(String txnClass, . . .) {
  String msg = "Service level agreement violation detected";

  MonitoringEvent me = new MonitoringEvent(
    TVisionCommon.COMP_ANALYZER,

"com.bristol.tvision.services.analysis.actions.LogSLAViolation"
,
    MonitoringEvent.CLASS_APPLICATION,
    "TVisionSLAViolation",
    MonitoringEvent.SEVERITY_WARNING);

  me.setMessage(msg);
  me.setParameter("txnClass", txnClass);
  . . . . .
  Logging.analyzerActivityLog.warn(me);
}
```

4.8.3. SlotMap.properties

This file is used by the log4j TECAppender to allow mapping of parameters set into MonitoringEvent to Tivoli slots. The file format is:

<MonitoringEvent parameter> = <Tivoli slot>

Any parameter specified here is explicitly mapped to a Tivoli slot. Parameter names unspecified in this file are mapped to Tivoli slots tv_attrib[1|2|3] and their values are mapped to slots tv_value[1|2|3].

4.8.4. Example Usage:

The following sample code writes an ERROR log message of class BTV_app_red, with parameters "application", "transaction_class" set.

```
MonitoringEvent ev = new MonitoringEvent
(MonitoringEvent.TVISION_EVENT_APPLICATION, MonitoringEvent.ERROR);

          ev.setParameter("application", "Trade");
          ev.setParameter("transaction_class", "TRADE_CLASS");
          ev.setParameter("message_id", "TransactionError");
        ev.setMessage("Error fulfilling transaction xyz");
          Logging.analyzerActivityLog.error(ev);
```

4.8.5. BTV Class Definitions and Rulebase

Class definitions supplied address the following events:

- Internal events - events generated regarding the TransactionVision application itself
- Applications events - events generated by entities that TransactionVision is monitoring

- Unknown events - events that have not fit the criteria to be defined beyond coming from TransactionVision.
- Escalation events - events of either internal or application that have exceeded count thresholds

The rules file creates the following classes related to TransactionVision:

BTV_app_black
BTV_app_red
BTV_app_yellow
BTV_app_green
BTV_int_black
BTV_int_red
BTV_int_yellow
BTV_int_green
BTV_unk

The classes BTV_int_[black|red|yellow|green] are used by TransactionVision internally while the classes BTV_app_[black|red|yellow|green] may be used by application plugin code. The color black, red, yellow, green indicates the severity level to be FATAL, ERROR, WARN and INFO respectively.

The following slots will be created:

message_id
tv_component
tv_attrib1
tv_attrib2
tv_attrib3
tv_value1
tv_value2
tv_value3
err_code
application
event_time
transact_class
transact_id

All slots may not be filled by TransactionVision internal messages.

Rulesets have supplied rules for the following:

- First instance rule which takes action upon an event the first time it arrives, or if there are no other like events in either OPEN or ACK status
- Duplicate rule which identifies an event as a duplicate to a previous event in either OPEN or ACK status, increments the repeat count on the original event, and drops the new event
- Escalation rule which takes action when an event has been received in succession for a defined count and status is of OPEN or ACK
- internal events, which are focused on the TransactionVision application itself.

5. Using the Query Services

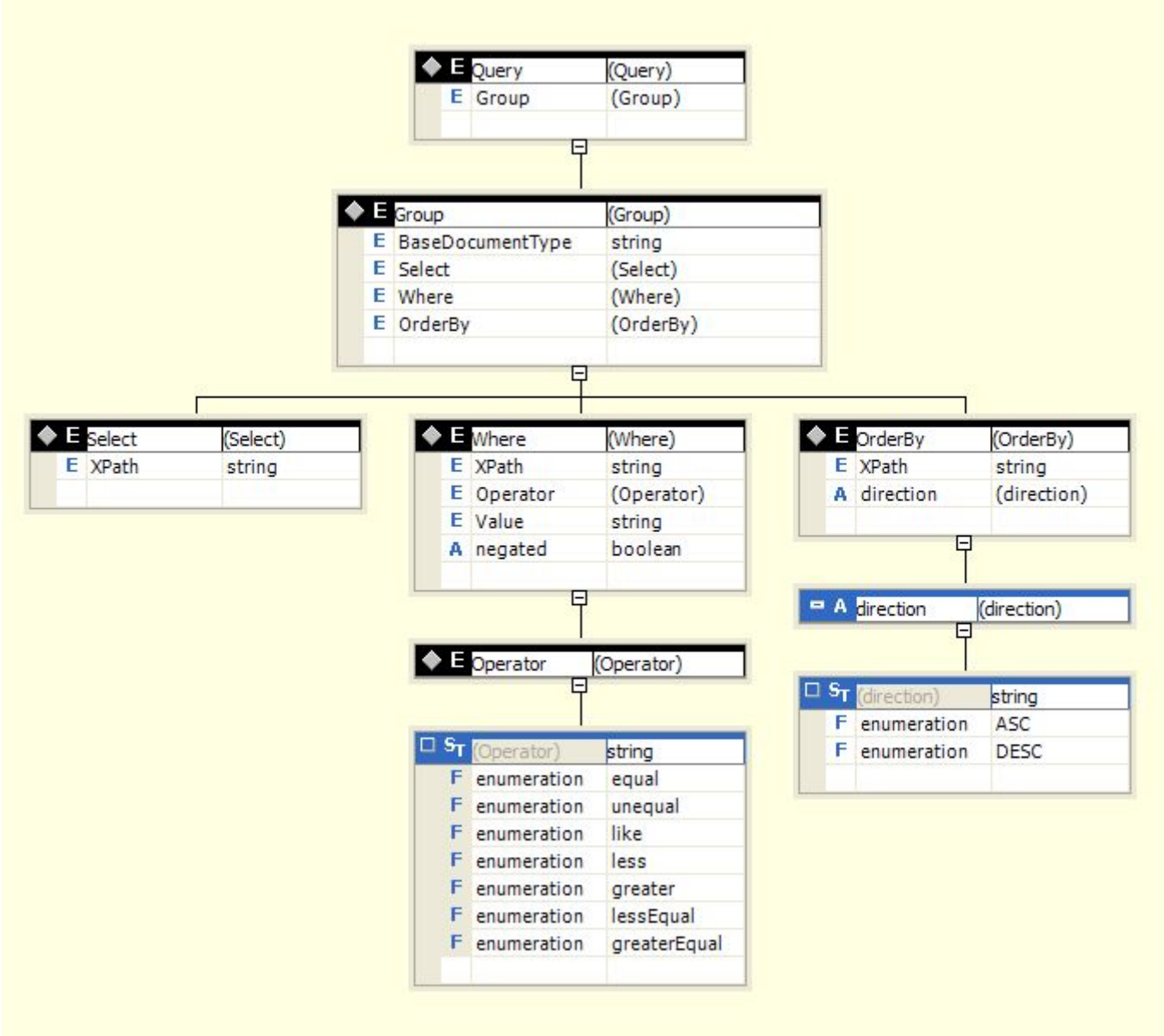
This chapter contains the following sections:

The Query Services interfaces provide a means to retrieve XDM mapped data from the database using an XML based query document. The QueryService interface is the top-level interface to create and run queries. The methods in this class return an object that implements the Query interface, which can be used to execute the query. Many of the methods in this class take a “query doc” argument – an XML document describing the query to execute. The query object can either be constructed manually with DOM tree operations, or by using the helper class QueryDoc which offers convenient methods to assemble the query and which is described in more detail later in this chapter. A Cursor object is returned from several of the QueryService methods, which allows a user to iterate over the results. The QueryService implementation converts the input XML query into an SQL statement and executes it. The Cursor class is a wrapper around the JDBC cursor classes.

The following sections will describe each of these objects and interfaces and show sample code to document their usage.

5.1. The Query Document

The query document is used to describe the query to be executed. The schema of the XML document is defined in the file **<TVISION_HOME>/config/xmlschema/Query.xsd**:



A sample query document:

```
<?xml version="1.0" encoding="UTF-8"?>
<Query>
   <BaseDocumentType>/Event</BaseDocumentType>

   <Select>/Event/PrimaryTime</Select>
   <Select>/Event/SecondaryTime</Select>

   <Group>
      <Where negated="false" translateValue=”false”>
         <XPath>/Event/Technology/MQSeries/@apiCode</XPath>
         <Operator>equal</Operator>
```

```
         <Value>8</Value>
         <Value>11</Value>
         <Value>12</Value>
      </Where>
   </Group>
   <OrderBy direction=”DESC”>/Event/PrimaryTime</OrderBy>
</Query>
```

The above query searches for events on the XPath “/Event/Technology/MQSeries/@apiCode”, that is the lookup column corresponding to the MQSeries API code for values 8 (MQGET), 11(MQPUT) and 12 (MQPUT1), and retrieves their primary and secondary time. The result is sorted by primary time in descending order. Note that you can specify multiple <Group> sections, and the conditions of all <Group> sections are ORed together in the final query.

```
<?xml version="1.0" encoding="UTF-8"?>
<Query>
   <Group>
      <Where negated="false" translateValue=”false”>
         <XPath>/Event/Technology/MQSeries/@apiCode</XPath>
         <Operator>equal</Operator>
         <Value>MQGET</Value>
      </Where>
      <Where negated="false" translateValue=”true”>
       <XPath>/Event/StdHeader/ProgramName/@objectId</XPath>
       <Operator>equal</Operator>
       <Value>amqsput</Value>
       <Value>amqsput1</Value>
      </Where>
   </Group>
</Query>
```

The above query searches for WebSphere MQ API “MQGET” events from programs with name “amqsput”and “amqsput1” and returns all column values.

An AND operation is performed on the two Where clauses in the above query, while an OR operation is performed on values within the same Where clause. There are two approaches to reference system model objects in a query document: you can specify the object_id that is stored in the lookup table, or you can set the attribute “translateValue” to true and compose a query doc based on object name instead of object id. This attribute causes the data in the <Value> subelement to be treated as an object name. The corresponding object ID is looked up and used before submitting the query to the query engine.

For example, the following code looks up an event where the program name is test and the internal system model table says the object ID of test is 12:

```
<Where negated="false">
  <XPath>/Event/StdHeader/ProgramName/@objectId</XPath>
  <Operator>equal</Operator>
  <Value>12</Value>
</Where>
```

To use the object name instead of the object ID, the code would be as follows:

```
<Where negated="false" translateValue=”true”>
   <XPath>/Event/StdHeader/ProgramName/@objectId</XPath>
   <Operator>equal</Operator>
```

```
   <Value>test</Value>
</Where>
```

Furthermore, you can use SQL wildcard support for a more powerful query:

```
<Where negated="false" translateValue=”true”>
     <XPath>/Event/StdHeader/ProgramName/@objectId</XPath>
     <Operator>equal</Operator>
     <Value>tes%</Value><!— query all program names beginning with
‘tes’ !-->
  </Where>
```

The “negated” and “translateValue” attributes are optional and default to ‘false’.

By default, if the query document does not contain a Select clause, the query will retrieve all available columns of the base document type. It is recommended to specify an explicit Select clause since the retrieval of all column values requires a database join of all tables.

The “BaseDocumentType” specifies the document type that should be used for the base query and determines the join order in the resulting SQL query. The element is optional, by default every query is considered to be an event based query with document type “/Event”. Besides “/Event”, the query engine also supports transaction based queries with the document type “/Transaction”:

```
<?xml version="1.0" encoding="UTF-8"?>
<Query>
   <BaseDocumentType>/Transaction</BaseDocumentType>

   <Select>/Transaction/ResponseTime</Select>

   <Group>
      <Where>
         <XPath>/Transaction/StartTime</XPath>
         <Operator>greater</Operator>
         <Value>20070327140015000000</Value>
      </Where>
   </Group>
</Query>
```

As a third option, the query engine allows to use both document types together. This allows to define queries that can use both event and transaction data. In this case the BaseDocumentType has to be set to the document type that should be used as the base table for the table joins.

5.2. Sample Usage

The following sample code shows how to create a query document, use the QueryService interface to get a Query object back and then execute the query. The sample counts the number of events for each MQPUT, MQPUT1 and MQGET. To assemble the query document the helper class QueryDoc is used.

```
// instantiate a new query document.
QueryDoc qdoc = new QueryDoc();

String[] apiCodes = { String.valueOf(MQDefs.MQPUT),
                      String.valueOf(MQDefs.MQPUT1),
```

```
                        String.valueOf(MQDefs.MQGET)};

// set the WhereClause of the QueryDocument to retrieve events
// containing a list of APIs, MQPUT, MQPUT1 and MQGET.
QueryDoc.WhereClause clause = new QueryDoc.WhereClause("mqputget",
                                            false,
                                            XPathConstants.APICODE,
QueryOp.EQ_QUERY_STRING,
                                            apiCodes,
                                            false);

// set the WhereClause into the QueryDoc.
qdoc.updateWhereClause(clause);

// select the fields to be retrieved in this case the program id.
String[] selects = { XPathConstants.PROGRAM_ID };
qdoc.insertSelect(selects);

// gets and execute the query.
Cursor queryCursor = customReportBean.getQueryResults(qdoc);

// map of API name versus event count for that API.
HashMap nameToCount = new HashMap();
int maxValue = 0;

// iterate through the query fetching the results from the database.
while (queryCursor.next()){

    String objValue = queryCursor.getValue(1,true);
    Integer count = (Integer)nameToCount.get(objValue);
    if (count == null)
    {
        count = new Integer(1);
        nameToCount.put(objValue,count);
    } else {
        int newValue = count.intValue() + 1;
        nameToCount.put(objValue,new Integer(newValue));
        if (newValue > maxValue)
            maxValue = newValue;
    }
}
// always close cursor at the end
queryCursor.close();
```

The method getQueryResults used in the above code snippet is as follows. This method gets the QueryService instance (QueryService is a singleton object per schema), gets an event list query object and executes the query, returning the result set cursor.

```
public Cursor getQueryResults(QueryDoc queryDoc) throws
DataManagerException
    {
    // get a reference to the singleton QueryService instance.
        QueryService queryServ = QueryService.instance(schemaName);

    // get a query object.
        Query queryObj = queryServ.getEventListQuery(dbConn,
queryDoc);

    // execute the query and return a result set cursor.
        return queryObj.execute();
```

```
}
```

5.3. Class QueryService

```
public class com.bristol.tvision.datamgr.query.QueryService
extends java.lang.Object
```

QueryService is the main interface to query the XDM tables. It is a singleton object that has methods that take a XML query document as the query definition and returns a query object. This query object can then be executed to obtain a cursor, which is then used in consecutive calls to retrieve data. All the methods in this interface that get a cursor or data from the database require a valid JDBC SQL connection handle. The methods throw a DataManagerException on an error condition occurring.

This interface defines the following methods.

Methods

- instance

```
instance
public static QueryService instance()
```

This method returns the singleton instance for the service.

Returns:

The return value is a reference to the singleton instance.

Example:

```
QueryService queryServ = QueryService.instance();
```

- getEventDetail

```
public org.w3c.dom.Document getEventDetail(ConnectionInfo con,
                                           EventID eventId,
                                           TypeConvService convSvr)
                                    throws DataManagerException
```

This method returns the event XML document for a given event.

Parameters:

con	The database connection to use
eventId	The specified event
convSvr	The TypeConvService allows fields like date and time formatting, time-zone and other conversions to be applied to the retrieved data. A value of null implies that no conversions are applied. Refer to the section on TypeConvService for more information on the supported conversions.

Returns:

The return value is an XML document containing event data.

Throws:

DataManagerException - if retrieving of the XML document fails

- getUserDataLength

```
public long getUserDataLength(ConnectionInfo con,
                              EventID eventId,
                              int dataNum)
                       throws DataManagerException
```

This method returns the length of a given message data segment for a given event. Typically, message data is segmented when a data collection filter using data ranges is used to collect data. In that case, this method allows you to get the size of a particular data segment.

Parameters:

con -	the database connection to use.
eventId	The event id the event that the message data belongs to.
dataNum	The segment number of the message data, where the first segment has index 0.

Returns:

The return value is the length of the message data segment.

Throws:

DataManagerException – occurs if the database operation fails.

- getUserData

```
public byte[] getUserData(ConnectionInfo con,
                          EventID eventId,
                          int dataNum,
                          int offset,
                          int length)
                   throws DataManagerException
```

This method returns a segment of a message data segment. This segment is specified by a starting offset (offset) and the length (length) to return.

Parameters:

con	The database connection to use.
eventId	The event id the user data belongs to.
dataNum	The segment number of the user data.
offset	The starting offset of the segment to retrieve.
length	The number of bytes to return.

Returns:

The return value is the message data part of the event of id eventId.

Throws:

DataManagerException - if database operation fails.

Example:

The following code retrieves the first (index 0) segment of the message data buffer into a byte array.

```
QueryService queryService =
QueryService.instance();
```

```
int dataLength =
(int)queryService.getUserDataLength(con, eventId, 0);
byte[] rawData =
queryService.getUserData(con,eventId,0,0,dataLength);
```

- getNextListDocument

```
public org.w3c.dom.Document getNextListDocument(Cursor cursor,
QueryResultPager pager, QueryResultFormatter formatter)
throws DataManagerException
```

This method Returns the next XML list document for a given query cursor and pager object that defines the page size.

Parameters:

cursor	The query cursor on the events
pager	The pager object (the same instance of the pager object has to be passed into consecutive calls of get
formatter	The formatter object to format the result

Returns:

The XML list document for the page.

Throws:

DataManagerException - if retrieving of the data or assembly of the XML document fails.

- getCorrelatedEventsQuery

```
public Query getCorrelatedEventsQuery(ConnectionInfo con,
       EventID eventId,org.w3c.dom.Document queryDoc)
       throws DataManagerException
```

This method creates a Query object to query the database for all events correlated to the event denoted by eventId. Execution of the query returns a Cursor that allows access to all columns specified in select section of the query, as well as to the column "confidence", “direction”, and “relation_type” of table event_relation.

Parameters:

con	The connection to use for executing the query
eventId	The eventID
queryDoc	The XML query document specifying the rows to include in the result. The WHERE clauses are ignored.

Returns:

A Query object ready for execution

Throws:

DataManagerException - if parsing the query doc or executing the query fails

- getQuery

```
public Query getQuery(Document queryDoc, boolean
useScrollableCursor)
       throws DataManagerException
```

This method returns a query that is executed across all analyzer schemas / databases. The results from each analyzer schema are merged together and returned.

Parameters:

queryDoc	The XML query document specifying the rows to include in the result. The WHERE clauses are ignored.
useScrollableCursor	Whether to use a scrollable cursor which allows forward and backward iteration

Returns:

A Query object ready for execution

Throws:

DataManagerException - if parsing the query doc or executing the query fails

- getQuery

```
public Query getQuery(ConnectionInfo conInfo, Document queryDoc,
boolean useScrollableCursor)
       throws DataManagerException
```

This method returns a Returns a query on a single schema (conInfo.schema), executed on a single database connection (conInfo.con).

Parameters:

conInfo	The database connection info, containing the database connection and the database schema name.
queryDoc	The XML query document specifying the rows to include in the result. The WHERE clauses are ignored.
useScrollableCursor	Whether to use a scrollable cursor which allows forward and backward iteration

Returns:

A Query object ready for execution

Throws:

DataManagerException - if parsing the query doc or executing the query fails

5.4. Class QueryDoc

```
public class com.bristol.tvision.shared.query.QueryDoc
extends com.bristol.tvision.shared.xml.XMLDocument
```

The QueryDoc class is a helper class that can be used to assemble a query document.

Constructors

- QueryDoc

```
public QueryDoc()
```

This constructor creates new QueryDoc with base document type “/Event”. The root element 'Query' is created automatically.

- QueryDoc

```
public QueryDoc(QueryDoc other)
```

This copy constructor creates a new QueryDoc from the given input QueryDoc.

Parameters:

other - QueryDoc instance used to create a new QueryDoc from.

- QueryDoc

```
public QueryDoc(java.lang.String baseDocType)
```

This constructor creates an empty QueryDoc for the specified base document type.

- toString

```
public String toString()
```

This method returns the XML query document as a string.

Returns:

The return value is a string of the XML document. Returns null on failure.

- insertSelect

```
public boolean insertSelect(java.lang.String[] xpaths)
```

This method sets an array of XPath expressions, which form the “SELECT” part of the query.

- updateWhereClause

```
public boolean updateWhereClause(WhereClause clause, int groupId)
```

Add a where clause under given query group. The groupId can be any integer > 0.For each ‘Group’ section in the query document use a different id.

Parameters:

clause	where clause
groupId	group id

Return:

true if operation succeeds.

- updateBufferClause

```
public boolean updateBufferClause(BufferClause clause,
     int groupId)
```

Add a buffer clause under given query group. The groupID can be any integer > 0.
Parameters:

clause	buffer clause
groupId	group id

Return:

true if operation succeeds.

- deleteWhereClauseByName

```
public void deleteWhereClauseByName(String name, int groupId)
```

Delete a where clause under the given query group. GroupId should be the ID of an existing query group.

Parameters:

name	where clause name
groupId	group id

- deleteBufferClause

```
public void deleteBufferClause(int groupId)
```

Delete a buffer clause under the given query group. GroupId should be the ID of an existing query group.

Parameters:

groupId - group id

- findWhereClauseByName

```
public WhereClause findWhereClauseByName(String name, int
groupId)
```

Retrieve the where clause of given name under given query group. GroupId should be the ID of an existing query group.

Parameters:

name	where clause name
groupId	query group id

Return:

WhereClause instance.

- getBufferClause

```
public BufferClause getBufferClause(int groupId)
```

Get buffer clause under given query group.

Return:

BufferClause instance

- getWhereClauseNames

```
public String[] getWhereClauseNames(int groupId)
```

Get all where clause names under given query group.

Return:

An array of where clause names.

- isLinearSearch

```
public boolean isLinearSearch()
```

Check if query document contains linear search clause.

Return:

true if query document is linear searching.

- isBufferSearch

```
public boolean isBufferSearch()
```

Check if query document contains buffer clause

Return:

true if there's at least one buffer clause

- equals

```
public boolean equals(QueryDoc d)
```

Check if two queries equal or not.

- groupCompare

```
public boolean groupCompare(QueryDoc d, int groupId1, int
groupId2)
```

Compare group of different query doc.

- printQueryDoc

```
public void printQueryDoc(OutputStream out)
```

Dump query document to given output stream.

Parameter:

`out` - output stream instance.

5.5. Class QueryDoc.WhereClause

This is an inner static class in the class QueryDoc. It is a utility class that helps to define the where condition of the query. This condition is the matching criteria for which events should be retrieved from the database.

Fields

- name

```
public java.lang.String name
```

Name of the where clause

- negated

```
public boolean negated
```

Whether the where clause has "not" condition

- xpath

```
public java.lang.String xpath
```

XPath for the where clause

- operator

```
public java.lang.String operator
```

Operator for the where clause

- values

```
public java.lang.String[] values
```

Values for the where clause

- isLinearCond

```
public boolean isLinearCond
```

  Specifies whether the “Where” clause is a linear search condition.

- needTranslate

```
public boolean needTranslate
```

  If true, causes all data in the <Value> subelement to be treated as an object name. The corresponding object ID will be looked up and used before submitting the query to the query engine.

- valueType

```
public java.lang.String valueType
```

- TYPE_BIN

```
public static final java.lang.String TYPE_BIN
```

- TYPE_TEXT

```
public static final java.lang.String TYPE_TEXT
```

- codePage

```
public java.lang.String codePage
```

Constructors

- QueryDoc.WhereClause

```
public QueryDoc.WhereClause()
```

  This constructor creates an empty object.

- QueryDoc.WhereClause

```
public QueryDoc.WhereClause(boolean Negated,
                            java.lang.String XPath,
                            java.lang.String Operator,
                            java.lang.String[] Values)
```

  This constructor creates a “WhereClause” object using the given data.

  Parameters:

  Negated - Whether the where clause has "not" condition

  XPath - XPath of where clause

  Operator - Operator of where clause

  Values - Values of where clause.

- QueryDoc.WhereClause

```
public QueryDoc.WhereClause(java.lang.String Name,
                            boolean Negated,
                            java.lang.String XPath,
                            java.lang.String Operator,
                            java.lang.String[] Values,
                            boolean IsLinearCond)
```

  This constructor creates a “WhereClause” object using the given data.

Parameters:

`Name` - Name of the where clause
`Negated` - Whether the where clause has "not" condition
`XPath` - XPath of where clause
`Operator` - Operator of where clause
`Values` - Values of where clause
`IsLinerCond` - True if where clause is at linear condition

- QueryDoc.WhereClause

```
public QueryDoc.WhereClause(java.lang.String Name,
                            boolean Negated,
                            java.lang.String XPath,
                            java.lang.String Operator,
                            java.lang.String[] Values,
                            boolean IsLinearCond,
                            java.lang.String valueType,
                            java.lang.String codePage)
```

This constructor creates a “WhereClause” object using the given data.

Parameters:

`Name` - Name of the where clause
`Negated` - Whether the where clause has "not" condition
`XPath` - XPath of where clause
`Operator` - Operator of where clause
`Values` - Values of where clause
`IsLinerCond` - True if where clause is at linear condition

valueType - Either of the following values. For display purpose only.

QueryDoc.WhereClause.TYPE_BIN

QueryDoc.WhereClause.TYPE_TEXT

codePage - The character set code page, that is used when converting the hexidecimal value string into text

Methods

- equals

```
public boolean equals(QueryDoc.WhereClause c)
```

This method compares two WhereClause objects.

Parameters:

`c` - another instance of `WhereClause`

Returns:

true if two are considered be equal

5.5.1. Example

The sample code below creates a query document with a “WhereClause” and a “SelectClause” using the methods updateWhereClause and insertSelect. The query condition is named “mqputget” and specifies to match all MQPUT, MQPUT1 and MQGET APIs. The data fetched out of the database is specified by the selects String array and contains the

XPath expressions for the fields entry time, exit time, API code, host id, program id, program instance id and sequence number.

```
QueryDoc qdoc = new QueryDoc();

String[] apiCodes = { String.valueOf(MQDefs.MQPUT),
                      String.valueOf(MQDefs.MQPUT1),
                      String.valueOf(MQDefs.MQGET)};
QueryDoc.WhereClause clause = new QueryDoc.WhereClause("mqputget",
                                                       true,
                                          XPathConstants.APICODE,
                                          QueryOp.EQ_QUERY_STRING,
                                          apiCodes,
                                          false);

String[] selects = { XPathConstants.PRIMARYTIME,
                     XPathConstants.APICODE,
                     XPathConstants.HOST_ID,
                     XPathConstants.PROGRAM_ID,
                     XPathConstants.PROGINST_ID,
                     XPathConstants.SEQUENCE_NO };

qdoc.updateWhereClause(clause, 1);
qdoc.addSelects(selects);
```

5.6. Interface Query

```
public interface com.bristol.tvision.datamgr.query.Query
```

This interface provides the functionality to run a query. This object is obtained from methods in the QueryService class.

Methods

- execute

```
public Cursor execute()
              throws DataManagerException
```

  This method executes the query and returns a Cursor object to be iterated over.

  Throws:

  DataManagerException - If executing the query fails

- close

```
public void close()
           throws DataManagerException
```

  This method closes the query and releases the database resources. The query can not be executed again once close has been called.

  Throws:

  DataManagerException - If release of the database resources fails

- cancel

```
public void cancel()
            throws DataManagerException
```

This method can be called from a different thread to cancel the current query execution.

Throws:

DataManagerException - If the cancel fails

5.7. Interface Cursor

public interface `com.bristol.tvision.datamgr.query.Cursor`

The cursor interface is used to iterate over data returned by a query.

Methods

- getRowCount

```
public int getRowCount()
```

This method returns the number of table rows in the query result, or -1 if this feature is not supported

Returns:

The number of rows

- getColumnCount

```
public int getColumnCount()
```

This method returns the number of columns in the query result

Returns:

The number of columns

- getColumnDescription

```
public java.lang.String getColumnDescription(int index)
```

This method returns the column description for the specified column. The index of the first column is 1.

Parameters:

index - The index of the column

Returns:

The column name

- getColumnName

```
public java.lang.String getColumnName(int index)
```

This method returns the database column name for the specified column. The index of the first column is 1.

Parameters:

index - The index of the column

Returns:

The column name

- getRow

```
public int getRow()
           throws DataManagerException
```

This method returns the current row for this cursor

Returns:

The current row, or 0 if there is no current row

- getValue

```
public java.lang.String getValue(int index)
                          throws DataManagerException
```

This method returns the value of the column as a String value. The index of the first column is 1.

Parameters:

index - The index of the column

Returns:

The value of the column, converted into a String

Throws:

DataManagerException - If getting the value from the underlying `ResultSet` fails

- getValue

```
public java.lang.String getValue(int index,
                                 TypeConvService convSvr)
                          throws DataManagerException
```

This method returns the value of the column as a String value (converted by the type conversion service). The index of the first column is 1.

Parameters:

index	The index of the column
convSvr	The type conversion service to use.

Returns:

The value of the column, converted into a String

Throws:

DataManagerException - If getting the value from the underlying ResultSet fails

- getIntValue

```
public int getIntValue(int index)
                throws DataManagerException
```

This method returns the value of the column as an integer value. The index of the first column is 1.

Parameters:

index - The index of the column

Returns:

The value of the column, converted into a integer

Throws:

DataManagerException - if getting the value from the underlying ResultSet fails

- getLongValue

```
public long getLongValue(int index)
                 throws DataManagerException
```

This method returns the value of the column as a long value. The index of the first column is 1.

Parameters:

index - The index of the column

Returns:

The value of the column, converted into a long

Throws:

DataManagerException - if getting the value from the underlying ResultSet fails

- getValue

```
public java.lang.String getValue(java.lang.String key)
                          throws DataManagerException
```

This method returns the value of the column as a String value. The column is identified by a key (XPath for XDM columns).

Parameters:

key - The key for the column

Returns:

The value of the column, converted into a String

Throws:

DataManagerException - If getting the value from the underlying ResultSet fails

- getValue

```
public java.lang.String getValue(java.lang.String key,
                           TypeConvService convSvr)
                           throws DataManagerException
```

This method returns the value of the column as a String value (possibly converted by the type conversion service). The column is identified by a key (XPath for XDM columns).

Parameters:

key	The key for the column
convSvr	The type conversion service to use

Returns:

The value of the column, converted into a String

Throws:

DataManagerException - If getting the value from the underlying ResultSet fails

- getIntValue

```
public int getIntValue(java.lang.String key)
                throws DataManagerException
```

This method returns the value of the column as an integer value. The column is identified by a key (XPath for XDM columns).

Parameters:

key - The key for the column

Returns:

The value of the column, converted into a integer

Throws:

DataManagerException - if getting the value from the underlying ResultSet fails

- getLongValue

```
public int getLongValue(java.lang.String key)
        throws DataManagerException
```

This method returns the value of the column as a long value. The column is identified by a key (XPath for XDM columns).

Parameters:

key - The key for the column

Returns:

The value of the column, converted into a long

Throws:

DataManagerException - if getting the value from the underlying ResultSet fails

- getValueMap

```
public java.util.Map getValueMap(TypeConvService convSvr)
                          throws DataManagerException
```

This method returns a Map object which contains a mapping from XPath to current column value, or null if this feature is not supported.

Parameters:

convSvr - The type conversion service to use.

Returns:

A Map object containing the values of the current row

Throws:

DataManagerException - if getting the values from the underlying ResultSet fails

- wasNull

```
public boolean wasNull()
                throws DataManagerException
```

This method reports whether the last column read with getValue() or getIntValue had a value of SQL NULL

Returns:

true if the last column value read was SQL NULL and false otherwise

Throws:

DataManagerException - if accessing the ResultSet fails

- next

```
public boolean next()
             throws DataManagerException
```

This method moves the cursor forward one row from its current position. A Cursor is initially positioned before the first row, calls to next() advance the cursor to the next row.

Returns:

true if the new current row is valid; false if there are no more rows

Throws:

DataManagerException - if moving the cursor in the underlying ResultSet fails

- previous

```
public boolean previous()
                 throws DataManagerException
```

This method moves the cursor backwards one row from its current position. A Cursor is initially positioned before the first row, calls to previous() advance the cursor to the previous row.

Returns:

true if the new current row is valid; false if there are no more rows

Throws:

DataManagerException - if moving the cursor in the underlying ResultSet fails

- absolute

```
public boolean absolute(int row)
             throws DataManagerException
```

This method moves the cursor to an absolute row position.

Parameters:

row - The row to position on

Returns:

true if the new current row is valid; false if cursor is not positioned on valid row

Throws:

DataManagerException - if positioning the cursor in the underlying ResultSet fails

- close

```
public void close()
           throws DataManagerException
```

This method closes the cursor and all with the cursor associated database resources

Throws:

DataManagerException - if closing the underlying JDBC resources fails

5.8. Class DataManagerException

```
public class DataManagerException
extends TVisionException
```

This exception class contains errors from the DataManager package.

Constructors

- DataManagerException

```
public DataManagerException()
```

This constructor creates new DataManagerException without a detail message string.

- DataManagerException

```
public DataManagerException(java.lang.Throwable t)
```

This method constructs a DataManagerException with the specified embedded Throwable.

- DataManagerException

```
public DataManagerException(java.lang.Object[] args)
```

This method constructs a DataManagerException with the specified logging arguments.

Parameters:

args - the logging arguments

- DataManagerException

```
public DataManagerException(java.lang.Throwable t,
                            java.lang.Object[] args)
```

This method constructs a DataManagerException with the specified embedded Throwable and the specified logging arguments.

Parameters:

t - the exception to chain

args - the logging arguments

Methods

- getSQLException

```
public java.sql.SQLException getSQLException()
```

This method returns the embedded exception as a SQLException if it is an instance of SQLException, null otherwise.

Returns:

The SQLException, or null if the embedded exception is not an instance of SQLException

- isUniqueViolationException

```
public boolean isUniqueViolationException()
```

  Returns true if the embedded exception is a SQLException indicating a violation of an unique constraint, false otherwise.

  Returns:

  true if exception is a unique constraint violation

- isOperationCanceledException

```
public boolean isOperationCanceledException()
```

  This method returns true if the embedded exception is an SQLException indicating that the executed SQL query has been canceled, false otherwise.

  Returns:

  true if exception is a SQL cancellation violation

6. Implementing User Events

This chapter contains the following sections:

TransactionVision supports accepting events created by user applications beyond those originating from the standard TransactionVision agents. In essence, you can add code in your application or by configuring the Java Agent points file to generate events in propriety format. This type of event is known as a user event. In general, your applications are also responsible for delivering the event to the Analyzer through the standard communication links. In some cases such as Java environment, the TransactionVision agent can be configured to generate and deliver user events automatically without custom coding. See the *Using Transaction Management* for information about enabling communication links to process user events in addition to standard TransactionVision events.

User events are defined as XML documents, and should conform to the standard imposed by the TransactionVision user event XML schema. Just like standard events, user events contain three sections: standard data, technology data, and user data. TransactionVision provides a Java User Event SDK for you to use to generate user events.

Each user event should reflect the transaction processing state at the point of origin. Events can be classified and associated with custom defined system resources (for example, database or FTP server).

Each user event should reflect the transaction processing state at the point of origin. Events can be classified and associated with custom defined system resources (for example, database or FTP server).

The TransactionVision Analyzer is capable of processing the standard header section automatically. By default, it does not perform any processing or analysis on the technology

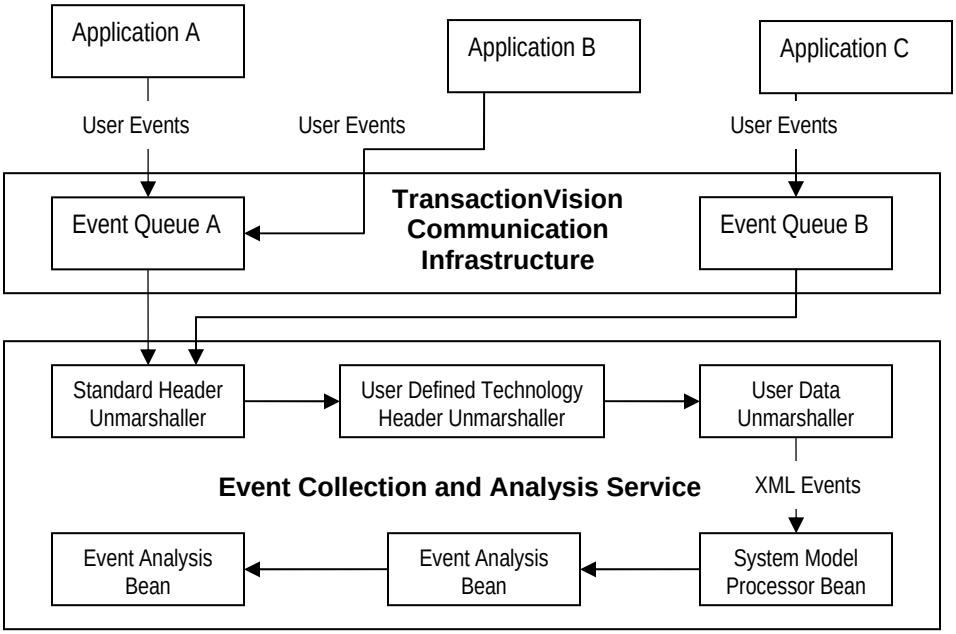
and user data section. However, you may implement additional beans for customizing the unmarshaling, modification, and database writing process.

User events should include data for event correlation, unit of work identification, and transaction classification. The XML rule engine in TransactionVision has been extended to also handle event-to-event correlation. It is also possible to implement event or transaction analysis beans for more complicated type processing.

6.1. Differences Between User Events and Standard Events

Standard TransactionVision events are distinguished from user events as follows:

- Standard events belong to technologies supported directly by TransactionVision. Sensor and analysis components are supplied. Events are generated automatically within applications by the Sensors.
- User events are designed and implemented by end users. They belong to the TransactionVision technology type "UserEvent". Events can be divided into different classes according to your design.



The following tables provide a comparison of standard and user events features:

Data Format:

	Standard Events	User Events
Standard data format	Binary	XML
Technology data format	Binary/XML	XML
User data format	BLOB/XML	XML
Event Correlation Data	Supported by TV	Defined by user
Transaction Data (e.g.: UOW id)	Supported by TV	Defined by user

Agent and Transportation Support:

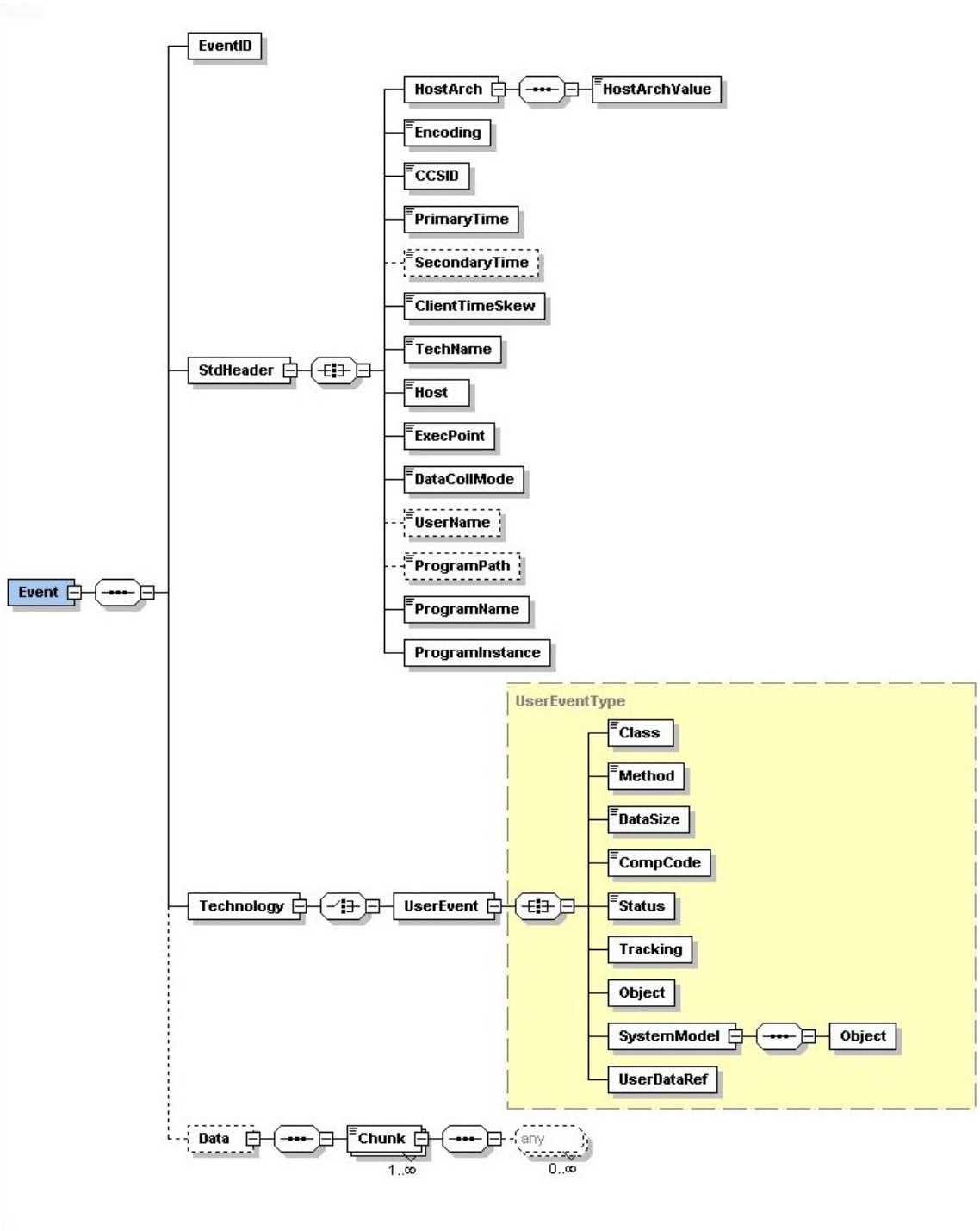
	Standard Events	User Events
Configuration Message	Yes	Yes (optional use subject to application)
Sensor Components	Yes	No
SDK Support	No	Yes
Data Collection Filtering	Supported	Supported
Event Generation Control	By data collection filters	By applications AND/OR data collection filters
Event Packaging	Supported	Supported

Event Analysis Support:

	Standard Events	User Events
Unmarshaling (Standard)	Supported by TV	Supported by TV
Unmarshaling (Technology)	Supported by TV	User Defined
Unmarshaling (User Data)	Supported by TV	User Defined
System Model Update	Supported by TV	TV/User Defined
Event Modification	TV/User Defined	TV/User Defined
Database I/O	Supported by TV	Supported by TV
Event Correlation	Supported by TV	TV(XML Rules)/User Defined
Local Transaction Analysis	Supported by TV	Supported by TV (data to be defined by users)
Business Transaction Analysis	Supported by TV	Supported by TV

6.2. User Event Data Model

The following is an example of user event XML content. User events should conform to the TransactionVision user event XML schema. The two schema files UserEvent.xsd and TechUserEvent.xsd can be found under the TransactionVision installation configuration directory **<TVISION_HOME>/config/xmlschema**:



The following shows a sample TransactionVision user event XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
  <Event>
    <EventID sequenceNum="631"/>
       <StdHeader minorVersion="1" version="5" uow=”12A2345”>
      <HostArch>
        <HostArchValue>0x80000380</HostArchValue>
      </HostArch>
      <Encoding>273</Encoding>
```

```
    <CCSID>1208</CCSID>
          <PrimaryTime>20030930155849620000</PrimaryTime>
       <SecondaryTime>20030930155849620000</SecondaryTime>
       <ClientTimeSkew>0</ClientTimeSkew>
       <TechName>UserEvent</TechName>
          <Host>bennytpc</Host>
       <ExecPoint>2</ExecPoint>
          <DataCollMode>7</DataCollMode>
          <ProgramName>ExecuteOrder </ProgramName>
       <ProgramInstance>
            <SensorStartTime>1060282060062</SensorStartTime>
            <ThreadStartTime>1060282060123</ThreadStartTime>
            <ThreadIdHash>231233575</ThreadIdHash>
          </ProgramInstance>
     </StdHeader>
         <Technology>
        <UserEvent>
          <Class>JDBC</Class>
          <Method>query</Method>
          <DataSize>1200</DataSize>
          <CompCode>0</CompCode>
          <Status>Normal</Status>
          <TrackingId>CA21B2335D3325</TrackingId>
        </UserEvent>
      </Technology>
         <Data>
        <Chunk seqNo="0" ccsid="1208" blobType="2" from="0" to="35">
             <Order>Buy 100 shares of IBM</Order>
           </Chunk>
         </Data>
</Event>
```

This section describes the user event XML elements and attributes. In many cases, TransactionVision defines the possible constant values for the elements and attributes through the following class com.bristol.tvision.userevents.Constants. There are also references to the User Event SDK provided by TransactionVision for assisting the user event generation process.

/Event (required): This is the root element for a single user event. Any user event must have one and only one instance of this element.

6.2.1. EventID

/Event/EventID (required): This element servers as the event identifier. It has one required attribute sequenceNum. This identifier must be unique in the application's thread of execution (program instance) space over the lifetime of the application.

Attributes:

Name	Type	Use	Description
sequenceNum	xsd:int	required	Uniquely identify the event among those belong to the same thread of execution.

6.2.2. Standard Section

/Event/StdHeader (required): This is the top level element for standard event data.

Attributes:

Name	Type	Use	Description
version	xsd:int	required	Major version number for the event. Should be set to Constants..EVENT_MAJOR_VERSION_LATEST.
minorVersion	xsd:int	required	Minor version number for the event. Should be set to Constants.EVENT_MINOR_VERSION_LATEST.
uow	xsd:string	optional	A string representing the local unit of work that the event participates in the application thread of execution. The analyzer uses this to group events to the same local transaction.

/Event/StdHeader/HostArch (required): This element describes the host machine architecture where the application runs: It contains one child element "HostArchValue" that contains a unique code defined by TransactionVision for identifying the host vendor and operating system. Java applications should use the User Event SDK to retrieve the host architecture value.

The host architecture code is a 32-bit integer that is divided into three separate subfields, these subfields identify

- The byte order (Big or Little Endian)
- The operating system
- The operator system vendor

In the following discussion, bit 0 is the most significant bit, and bit 31 is the least significant bit.

BYTE ORDER

Mask for binary-integer encoding.

This subfield occupies bit positions 0 through 7 and 24 through 31 within the host architecture value field. For big endian architecture, this should be set to 0x80000080. For little endian architecture, this should be set to 0x00000000.

VENDOR

Mask for vendor code.

This subfield occupies bit positions 8 through 15 within the host architecture value field. The possible values are as follows:

Vendor Name	Code Value
Microsoft	0
Sun Microsystem	1
Hewlett-Packard (HP)	2
IBM	3
Linux	4
Digital	5

OPERATING SYSTEM

Mask for operating system code.

This subfield occupies bit positions 16 through 23 within the host architecture value field. The possible values are as follows:

Operating System	Code Value
Microsoft Windows 3.1	0
Microsoft Windows 95	1
Microsoft Windows 98	2
Microsoft Windows 2000	3
Microsoft Windows NT	4
Sun Solaris	5
Hewlett Packard HP-UX	6
IBM AIX	7
IBM OS390 (CICS)	8
Linux	9
IBMOS390 (Batch)	10
IBM OS400	11
IBM OS390 (IMS)	12
Microsoft Windows ME	13
Tru64 UNIX	14
IBM Sun OS	15
Microsoft Windows XP	16
Microsoft Windows 2003	17

/Event/StdHeader/Encoding (required): This element contains an integer code for identifying the numerical encoding of the application environment (integer, floating point). Java applications should use the User Event SDK to retrieve this value.

The Encoding field is a 32-bit integer that is divided into four separate subfields; these subfields identify:

- The encoding used for binary integers
- The encoding used for packed-decimal integers
- The encoding used for floating-point numbers
- Reserved bits

Each subfield is identified by a bit mask which has 1-bits in the positions corresponding to the subfield, and 0-bits elsewhere. The bits are numbered such that bit 0 is the most significant bit, and bit 31 the least significant bit. The following masks are defined:

INTEGER_MASK

Mask for binary-integer encoding.

This subfield occupies bit positions 28 through 31 within the Encoding field.

DECIMAL_MASK

Mask for packed-decimal-integer encoding.

This subfield occupies bit positions 24 through 27 within the Encoding field.

FLOAT_MASK

Mask for floating-point encoding.

This subfield occupies bit positions 20 through 23 within the Encoding field.

RESERVED_MASK

Mask for reserved bits.

This subfield occupies bit positions 0 through 19 within the Encoding field.

/Event/StdHeader/CCSID (required): This element contains an integer code for identifying the character code set of the application environment. The User Event SDK defines the default value as UTF-8 for Java environment.

/Event/StdHeader/PrimaryTime (required): This element contains a string representing the primary time stamp of the event up to microseconds. The format of the string is yyyyMMddhhmmssuuuuuu, where yyyy is the 4-digit year field, MM is the 2 digit month field, dd is the 2-digit day field, hh is the 2-digit 24-hour based hour field, mm is the 2-digit minute field, ss is the 2-digit second field, and uuuuuu is the 6-digit microsecond fields. The User Event SDK provides support of retrieving and rendering the time stamp value.

For cases where a distinct entry and exit time stamps are to be associated with a single event, set this and the SecondaryTime element to the entry and exit time respectively.

/Event/StdHeader/SecondaryTime (required): This element contains a string representing the secondary time stamp for the event. The format is similar to that for the primary time element. As stated before, this can be set to hold the exit time stamp of an event. The secondary time is set to the primary time value if this element is absent.

/Event/StdHeader/ClientTimeSkew (optional): This element contains a long value defining the time skew (in milliseconds) between the two hosts where (a) the application runs and (b) the event queue/manager receiving the event resides. If the event queue/queue manager exists on the same host where the application runs, this should be set to zero. The time skew is assumed to be zero if this element is absent.

/Event/StdHeader/TechName (required): This element contains a string that identifies the technology. This should always be set to “UserEvent”. Java applications can retrieve this value through the constant variable TECH_NAME_USEREVENT defined in the Constants class.

/Event/StdHeader/Host (required): This element contains a string that identifies the host where the application runs.

/Event/StdHeader/ExecPoint (required): This element contains that an integer that is a numerical byte code defined by TransactionVision for identifying the monitored method execution point. For user events, this should always be set to the value Constants.EP_EXIT (=2).

/Event/StdHeader/DataCollMode (required): This element contains an integer that is a numerical byte code defined by TransactionVision for identifying the data collection mode. For user events, the two possible values are “Collect technology and user data” (Constants.DATA_COLL_MODE_ALL_MASK , value = 7) or “Collect technology data only, no user data” (Constatns.DATA_COLL_MODE_ARG_MASK, value = 3).

/Event/StdHeader/UserName (optional): This element contains a string that identifies the user context of the running application. This can be set to the user id running the application.

/Event/StdHeader/ProgramPath (optional): This element contains a string that defines the application path on the host. This is not needed for Java programs.

/Event/StdHeader/ProgramName (required): This element contains a string that defines the application name.

/Event/StdHeader/ProgramInstance (required): This element contains one or more child elements that together identify the runtime thread of execution (program instance) where the event occurs. For example, this can be threads in JVM on distributed platforms. The actual elements and their meaning are specific to the platform and environment.

For Java environment, the program instance identifier contains the following three elements:

1. /Event/StdHeader/ProgramInstance/SensorStartTime: This element contains a string that represents a timestamp in milliseconds. This should be a value unique across all application threads. The User Event SDK can automatically generate a value based on the time when the SDK singleton helper class is created. The application can choose to overwrite with another reference time stamp that is unique in the application space.
2. /Event/StdHeader/ProgramInstance/ThreadStartTime. This element contains a string that represents a timestamp in milliseconds. This should be a value unique in the application thread where the event happens. The User Event SDK can automatically generate this value. The application can choose to overwrite with another reference time stamp if it wishes.
3. /Event/StdHeader/ProgramInstance/ThreadIDHash. This element contains a string that represents a hash value for the Java thread ID. The User Event SDK can automatically generate this value. Unlike the other two attributes, the SDK does not allow user to overwrite the default value.

Important. Java applications should avoid generating this element directory and leverage the SDK support instead.

6.2.3. Technology Section

/Event/Technology: This is the top level element for technology data. It has exactly one child element /Event/Technology/UserEvent. All the standard child elements for the UserEvent element are optional. The technology section can contain any number of application defined child elements attached to the UserEvent element.

The standard elements are defined as follows:

/Event/Technology/UserEvent/Technology: This element contains a string that defines the fine grain technology category among different user event types.

/Event/Technology/UserEvent/Class: This element contains a string that describes the class or category type of information within the technology. ”.

/Event/Technology/UserEvent/Method: This element contains that a string that describes the method/API of the event (e.g.: insert/update/query for JDBC).

/Event/Technology/UserEvent/DataSize: This element contains an integer that represents the user data length.

/Event/Technology/UserEvent/CompCode: This element contains an integer representing the event completion (return) code. TransactionVision defines the possible values in the Constants class.

/Event/Technology/UserEvent/Status: This element contains a string that representing the status (reason) code supplementing the completion code. For example, this can be the SQL error code further explaining the JDBC operation result.

/Event/Technology/UserEvent/Tracking: This element contains a mandatory string attribute id that defines a unique ID for grouping events belonging to the same business transaction. In other words, events having the same tracking ID would be put into the same business transaction record in TransactionVision. This is different from the unit of work attribute (/Event/StdHeader/@uow) since the unit of work ID is used for grouping events into the same LOCAL transaction.

Attributes:

Name	Type	Use	Description
id	xsd:string	Required	A tracking string for correlating events belongs to the same business transaction across multiple applications and platforms.

/Event/Technology/UserEvent/UserDataRef: This element serves as a reference tag to the user data (if presented). It has a single mandatory integer attribute “chunk” that should always be set to the value 0.

Attributes:

Name	Type	Use	Description
chunk	xsd:int	required	A tracking string for correlating events belongs to the same business transaction across multiple applications and platforms.

/Event/Technology/SystemModel: A user event can be associated with a system resource. For example, a JDBC event can be associated with the database table that the JDBC call operates on. This element defines the model for any such system resource. The system model can have one or more child elements /Event/Technology/SystemModel/Object. Each object element contains a user defined system object. The object element has the following mandatory attributes:

Attributes for /Event/Technology/SystemModel/Object:

Name	Type	Use	Description
id	xsd:int	required	The object local ID in the system model and event context. The event element for the event system resource /Event/Technology/Object uses this ID for identifying the object.
typeId	xsd:int	required	The object type ID in the Analyzer project-wise system object model table. Any use defined object type should have a type ID value greater than a well-defined base value Constants.USEROBJECT_TYPE_BASE (= 100000).
name	xsd:string	required	The object name. In general, this should be a simple string.
sig	xsd:string	required	A string that services as the object unique signature in the Analyzer project system object table. The signature should have the format <typeId>/<signature name>. TypeId is the global object type ID, while “signature name” can be any string chosen by the user. For example, a database table MYTABLE can have the signature 100100/MYTABLE, where 100100 is the type ID for database tables.

/Event/Technology/Object: This element identifies the system resource object associated with this event. For example, this can refer to a database table for a JDBC insert event. The presence of this element implies that the application interacts with the system resource in the scope of the event lifetime.

Attributes:

Name	Type	Use	Description
id	xsd:int	required	The local object ID for this object, as defined in the local system model.
dir	xsd:int	required	An integer defining the direction of interaction of the application and resource object. The two possible values are Constants.USEREVENT_PATH_IN and Constants..USEREVENT_PATH_OUT. PATH_IN indicates that data flows from resource to application, and PATH_OUT implies the opposite. For example, if an event is supposed to represent a database query, the direction should be set to PATH_OUT, implying the application is retrieving data from the database (resource).
latency	xsd:string	optional	A long value representing the latency of the application-resource interaction in milliseconds. For example, for a database query event, this can represent the amount of time taken for the query to complete.

6.2.4. User Data Section

/Event/Data (optional): This is the top level element for storing any user payload data. User data should also be in XML format. There should be one child element /Event/Data/Chunk attached to the /Event/Data element. The user data document should then be attached to the Chunk element as its child. The following attributes should be set for the Chunk element:

Attributes:

Name	Type	Use	Description
seqNo	Xsd:int	required	This should always be set to 0.
blobType	Xsd:int	required	Identify the type of the data attached. This should always be set to Constants.XMLEVENT_BLOB_XML (=2).
ccsid	Xsd:int	required	This should be set to the character code set for the user data.
from	xsd:int	required	This should always be set to 0.
to	Xsd:int	required	This should be set to the user data length minus 1.

6.3. Analyzing User Events

This section discusses the specific customization or configuration for user event analysis. For information on correlating user events into transactions, see section 4.5. For information about extending the system model for user events, see section 4.6.

6.3.1. Event Unmarshalling

By default, TransactionVision Analyzers would extract the user event XML document from the event messages, and convert it into the internal XMLEvent class object (com.bristol.tvision.services.analysis.xml.XMLEvent). The XMLEvent class implements the interface org.w3c.dom.Document and can be manipulated like any other XML document. Since the user event is in XML format, minimal modifications are needed to the incoming document.

Should you decide to further customize the unmarshalling logic for the technology and user data section, you can elect to develop a bean implementing the com.bristol.tvision.services.analysis.unmarshal.IUnmarshal interface. There is one difference between unmarshalling user events and standard Sensor events. In the former case, the whole XML document has already been read off from the input stream and attached to the XMLEvent structure. Thus the unmarshalling logic for user event should not attempt to read from the event input stream. Instead, it should focus on modifying the XMLEvent document instead.

6.3.2. Local Transaction Analysis

TransactionVision implements a local transaction generation algorithm through the bean com.bristol.tvision.services.analysis.eventanalysis.UserEventLocal Transaction for all user events. To group user events into the same local transaction, this bean uses the user event unit of work ID (/Event/StdHeader/@uow) and the program instance identifier.

6.3.3. Business Transaction Analysis

By default, the Analyzer is capable of putting user events belonging to different local transactions into the same business transactions by either event relations or tracking ID.

In the first case, two local transactions containing user events are put into the same business transaction if at least one event relation exists between events from either local transaction.

In the second case, two local transactions containing user events are put into the same business transaction if at least one event from each local transaction shares the same tracking id (/Event/Technology/UserEvent/Tracking/@id).

6.3.4. Statistical Analysis

For user events that have an associated system object (resource), the Analyzer will generate aggregated latency statistics over fixed intervals. Individual latency statistics will be gathered for each application-system object pair with a particular flow direction.

For example, if there are two applications reading from and writing to five different database tables, a total of twenty (20) data sets will be created and updated for every application-resource combination in either flow direction (20 = 2 x 5 x 2).

The statistics computation and aggregation is handled by the Java bean com.bristol.tvision.services.analysis.statistics.UserEvent StatisticsBean. This can be enabled and disabled by modifying the corresponding entry in the Beans.xml file.

6.4. Tutorial: Generating User Events

This section provides a tutorial sample that demonstrates how to generate user events with the TransactionVision User Event SKD and helper classes.

The source code and build files for this tutorial are located in the TransactionVision **<TVISION_HOME>/samples/userevent/tutorial** directory. This directory contains the following files:

File	Description
build.xml	Ant build file
readme.txt	Readme file for the tutorial
SystemModelDefinition.xlm	System model definitions for this sample
TechUserEvent.xsd	XML schema for user events

TVisionUserEvent.java	Source code for the tutorial
tvUserEvent.bat	Script to run the sample on Microsoft Windows.
tvUserEvent.sh	Script to run the sample on UNIX platforms.
UserEvent.xsd	XML schema for user evnets

6.4.1. Sample Overview

This sample generates a single user event representing a JDBC query activity. It delivers the event message through WebSphere JMS. The user can define the destination WebSphere MQ queue manager and queue to receive the generated user event.

First, examine the main routine of this Java sample, found in TVisionUserEvent.java:

```
/**
 * Main routine for composing and delivering a single
 * TransactionVision user event to the analyzer through a WebSphere
 * MQ communication link.
 */
public static void main (String args[])
{
  /* Check command line arguments */
  . . . . . .

  /* Set up JMS(WebSphere MQ) connection for delivering user
     events to TransactionVision communication link. */
        . . . . . .

  /* Generate and delivery a user event */
  sendEvent();

  /* Shut down JMS connection */
  . . . . . .
}
```

The majority of the code in this method validates the command line arguments and handles the JMS connection for delivering the user event message. This code for this has been omitted in the above code snippet.

The sendEvent() method contains the code that generates and delivers the user event

In the following code segment, the UserEventHelper class records the start and end time of the JDBC activity. The sleep call simulates the elapsed time of a JDBC call.

```
/* We simulate a JDBC query call by sleeping for at least 1 second.
   Use the SDK helper function to get event start and end time. */
     TimeData startTime = UserEventHelper.getCurrentTime();
     Thread.currentThread().sleep(1000);
     TimeData endTime = UserEventHelper.getCurrentTime();
```

The following code segment creates a system object to represent the database named “tradedb01.” Note that the system object type identifier (100001) must be consistent with the data in the SystemModelDefinition file in the TransactionVision configuration directory.

```
/* Create a database system model object use this as the resource
   for the JDBC user event */

SystemModelObject dbObj = new SystemModelObject(
     1,            /* local object id in system model */
     100001,       /* user defined system object type id */
```

```
      "tradedb01",  /* object name */
      "tradedb01"); /* object signature string */
```

Next, the sample prepares the user data for a book order in XML format:

```
/* Compute the application-resource latency time by comparing the
   start and end time */
long latency = endTime.timeInMillis - startTime.timeInMillis;

/* Prepare user data for the event and record data length */
String userData = "<BookOrder><ISBN>0743267524</ISBN>" +
  "<Quantity>1</Quantity></BookOrder>";
int userDataSize = userData.length();
```

The following code uses the User Event SDK UserEventSkeleton class to generate the basic XML document for the user event standard header section. It uses several set methods in the UserEventSkeleton class to set the context data such as program name, event primary and secondary time, and unit of work identifier.

```
/* Create a TransactionVision user event skeleton through the SDK.
   Most environment settings such as host architecture, encoding,
   character code set are set by default. We only need to set the
   program name, timestamps, and technology attributes. */
UserEventSkeleton userEvent = new UserEventSkeleton();
userEvent.setProgramName("OrderProcessor");
userEvent.setPrimaryTime(startTime.strTime);
userEvent.setSecondaryTime(endTime.strTime);
userEvent.setUOW("A67524B2236"); /* unit of work chosen by user */
Next, the sample creates the Java string for storing the complete
user event XML document. Note the use of the UserEventSkeleton class
to help generating an event sequence number.
/* Start composing the user event with the XML header */
String strEvent = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>";
/* Create an event sequence number that should be unique in the
   application life time. The SDK skeleton class can supply such
   number. */
int eventSeqNo = userEvent.getEventIDSequenceNum();

/* Add the standard header. We can leverage the user event skeleton
   for the bulk of the standard header content.  */
strEvent += "<Event><EventID sequenceNum=\"" + eventSeqNo + "\"/>";
Next, the UserEventSkeleton class object generates the string for
the standard header:
strEvent += userEvent.getStandardHeader();
The following is the technology section:
/* Start the technology section */
strEvent += "<Technology><UserEvent>";
/* TransactionVision defines several completion code in the class
   com.bristol.tvision.userevents.Constants */
int compCode = Constants.USEREVENT_COMPCODE_OK;
strEvent += "<Class>JDBC</Class>";
strEvent += "<Method>query</Method>";
strEvent += "<DataSize>" + userDataSize + "</DataSize>";
strEvent += "<CompCode>" + compCode + "</CompCode>";
strEvent += "<Status>Normal</Status>"; /* status chosen by user */
```

Next, add a reference to the database system model object associated with the JDBC activity you are reporting. In this case, the sample reports an inbound activity (query), the application reading data from the database:

```
/* This event refers to the database resource we created above.
   Use the local object id as reference. Since the event represents
   a query action, the relation direction is set to point from the
```

```
   database object to the application (inbound).
   The direction constants are defined in the class
   com.bristol.tvision.userevents.Constants */
int pathDir = Constants.USEREVENT_PATH_IN;
strEvent += "<Object id=\"" + dbObj.id + "\" ";
strEvent += "dir= \"" + pathDir + "\" ";
strEvent += "latency= \"" + latency + "\" />";
```

The following code segment generates a unique transaction tracking ID for grouping events of the same business transaction:

```
/* Generate a unique tracking id for grouping events belonging to
   the same transaction together */
strEvent += "<Tracking id=\"1A2D357236B17925\"/>";
```

The following code writes details about the system model object for the database by using the serialization method toXML() of the system model object class:

```
/* Generate the system model for our database object. Use the
   toXML() method for serialization. */
strEvent += "<SystemModel>";
strEvent += dbObj.toXML();
strEvent += "</SystemModel>";
```

We add the UserDataRef element as a reference to the user data for this event:

```
/* Create a user data reference line */
strEvent += "<UserDataRef chunk=\"0\"/>";
```

We have now completed the technology section of the user event:

```
/* Close technology section */
strEvent += "</UserEvent></Technology>";
```

The following code segment attaches the user data to the event

```
document under the "/Event/Data/Chunk" node:
/* Create the user data section, make sure we set the type to
   com.bristol.tvision.userevents.Constants.XMLEVENT_BLOB_XML */
int toIndex = userDataSize - 1;
String chunkEleBegin =
  "<Chunk seqNo=\"0\" ccsid=\"1208\" blobType=\"" +
  Constants.XMLEVENT_BLOB_XML + "\"" +
  " from=\"0\" to=\"" + toIndex + "\">";
String chunkEleEnd = "</Chunk>";
strEvent += "<Data>" + chunkEleBegin + userData + chunkEleEnd +
            "</Data>";
```

The event document has now been completed:

```
/* Close the event */
strEvent += "</Event>";
```

Optionally, we can validate the document generated against the XML schema **UserEvent.xsd**. This sample has a copy of this schema file in the sample directory. This schema file is also available in the TransactionVision configuration directory. The validation code can be found in the tutorial Java source file:

```
/* Parse and validate the user event through XML parser */
if (validateEvent(strEvent) == false)
   return;
System.out.println("Event document comforms to XML schema.");
```

Finally, we are ready to deliver the user event through JMS. Note that since this sample uses JMS to deliver the user event, we may get TransactionVision JMS events for these activities if the sample is run in a Sensor-enabled environment. The SDK helper class allows you to

temporarily disable such JMS events generation through the method disableTVisionJMSSensor().

```
/* Send user event through JMS. It is possible that this application
   is monitored by TransactionVision JMS sensor. In order to avoid
   sending standard TV JMS events for our user event delivery calls,
   temporarily disable the JMS sensor. It may be necessary to add
   synchronization control in multi-thread environment. */
bSensorMode = UserEventHelper.disableTVisionJMSSensor();
bRestoreSensor = true;
TextMessage message = tvQSession.createTextMessage(strEvent);
```

All user event JMS messages should have the correlation ID set to the well-defined value Constants.TVISION_USEREVENTS_ID:

```
/* All user event messages should have the same correlation ID
   as defined in the com.bristol.tvision.userevents.Constants
   class. */
message.setJMSCorrelationID(Constants.TVISION_USEREVENTS_ID);
tvQSender.send(message);
tvQSession.commit();
System.out.println("Send user event to TransactionVision event queue.");
```

Now that we have delivered the message, we can restore the normal behavior of the JMS Sensor with restoreTVisionJMSSensor().

```
/* We can now resume JMS sensor collection */
bRestoreSensor = false;
UserEventHelper.restoreTVisionJMSSensor(bSensorMode);
```

6.4.2. Building the Tutorial Sample

Use the included ant file to build this tutorial. Make sure you update the **build.xml** file so that the following directory properties are set according to your local environment:

Property	Description
mq.dir	The WebSphere MQ installation directory
tvision.dir	The TransactionVision installation directory

This sample uses the XML schema files **UserEvent.xsd** and **TechUserEvent.xsd** for validating the user event XML document generated. These two files can also be found in the config/xmlschema directory under the TransactionVision installation directory.

This sample makes use of the user event SDK tvisionuserevents.jar under the TransactionVision installation (**<TVISION_HOME>/java/lib/tvisionuserevents.jar**).

6.4.3. Running the Tutorial Sample

To set up a TransactionVision project, run the tutorial sample, and collect user events, perform the following steps:

1. Merge the provided SystemModelDefinition.xml file with the one in the TransactionVision installation under the directory <TVISION_HOME>/config/sysmodel.
2. Create a TransactionVision Analyzer with one communication link.
3. Make sure the communication link is created with User Event Processing support enabled. This option is available in the Miscellaneous Information section of the communication link editing user interface.

4. This sample makes use of JMS (WebSphere MQ) for delivering the user event messages. Make sure the event queue created can be accessed by the sample through MQ SERVER connection.
5. Before running the sample, make sure you turn on Analyzer collection.
6. Use the script **tvUserEvent**.[bat|sh] to run the sample. The script takes two required command line arguments which specify the event queue name and queue manager name respectively (in the specified order).

   For example, if the event queue has name TVISION.EVENT.QUEUE on the queue manager trading, run the sample as follows:

```
% tvUserEvent.sh TVISION.EVENT.QUEUE TRADING
```

After a successful run of the sample, you should find a single user event in the project database.

6.5. Configuring the Java Agent Points File

You can include custom company methods as part of the TransactionVision Transaction path to be presented as events. These methods are not normally included in the Event History unless they are part of the standard Java application framework such as JMS, EJB, Servlets, JSP, etc.

To include these custom events, modify the auto_detect.points file and either create a TransactionVision only point or modify an existing Diagnostics point to specify a TransactionVision user event by specifying the tv:user_event tag on the details line

For example:

```
#
# GENERIC EVENT
#
[GenericEvent]
class      = !com.company.importantclass
method     = !.*
signature  = !.*
detail     = tv:user_event
```

For more information about creating and modifying existing instrumentation points, see advanced instrumentation in the *HP Diagnostics Installation and Configuration Guide*.

7. Database Schema

This chapter contains the following sections:

- 8.1. System Model Object tables
- 8.2. Event Tables
- 8.3. Event Relationship Tables
- 8.4. Transaction Tables
- 8.5. Statistics Tables
- 8.6. RUM processing Tables
- 8.7. Other internal tables

7.1. System model object tables

The System Object Model tables are used to store all the System Model objects and the relationships between them. System model objects include general resources as well as technology-specific resources.

7.1.1. Object Types

As such, different technologies will be assigned different ranges of object types. This is described in the table below.

Object Types

Value (range)	Description
0 – 999	Basic System Model Objects (hosts, technologies, Program Instances, etc.)
1	Host
2	Not used
3	Program
4	Program Instance
5	z/OS Jobname

6	z/OS Jobstep
7	z/OS CICS Region
8	z/OS CICS Transaction
9	z/OS IMS ID
10	z/OS IMS Region Type
11	z/OS IMS Region ID
12	z/OS IMS Transaction
13	z/OS IMS PSB
14	OS400 Jobname
15	z/OS CICS Task
16	User Name
17	Proxy
18	Statistics
100	Transaction Class
510	CICS Task
511	CICS Transaction
512	CICS Region
520	Batch TCB
521	Batch Jobstep
522	Batch Job
1000-1999	MQSeries Objects
1000	Unknown type
1001	None
1002	Queue
1003	Local Queue
1004	Model Queue
1005	Alias Queue
1006	Remote Queue
1007	Cluster Queue
1008	Local Cluster Queue

1009	Alias Cluster Queue
1010	Remote Cluster Queue
1011	Namelist
1012	Process
1013	Queue Manager
1014	Distribution List
1015	Cluster
1016	WBI Message Flow
1017	WBI Broker
1018	Connection Name
1019	Cluster Name
1020	ReplyTo Queue
1021	ReplyTo Queue Manager
2000	Proxy Object
3000-3100	Servlet Objects
3000	Server
3001	UI/Job Server
3002	Servlet
3003	Internet
3004	JSP
3005	EJB
3006	EJB Method
3007	Probe
3008	Probe Group
3101-3199	JMS Objects
3101	Topic
3102	Queue
3103	Connection Name
3104	Tibco Global Queue
3105	Tibco Global Topic

3106	Sonic Broker URL
3107	Sonic Node
3108	Sonice Topic Routing Definition
3109	Sonic Queue Routing Definition
3110	Sonic Cluster
3111	Sonic Cluster Queue
3112	Sonic Cluster Topic
3113	BEA Server
3114	Sonic Domain
4000-4999	CICS Objects
4001	SYSID
4002	APPLID
4003	`TREMID`
4004	File
4005	TD Queue
4006	TS Queue
4007	TD Alias Queue
5000-5999	User Event Objects
5001	User Event Class
5002	User Event Method
5003	User Event Status
5004	User Event Technology
6000-6999	JDBC Objects
6001	Database
6002	Schema
6003	Table
6004	View
6005	Alias
6006	SQL
6007	DB Object Group

6008	SQL Statement
6009	Procedure
6010	Database URL
7000-7999	RUM Objects
7001	End User Name
7002	Country
7003	State
7004	City
7005	End User Group
8000-8999	Tuxedo Objects
8001	Queue
8002	Queue Space
8003	Service

7.1.2. Signatures

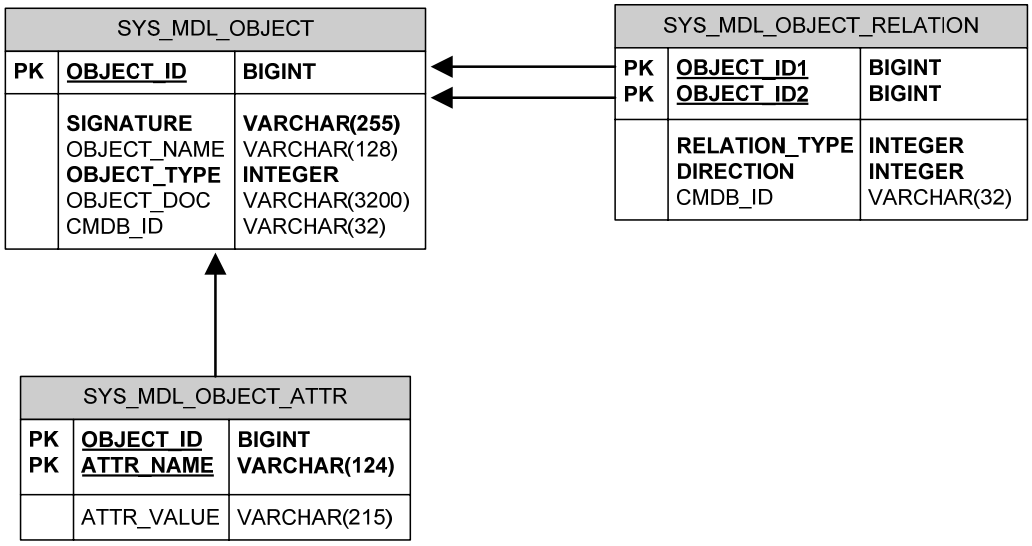
Each System Model Object has a unique object id that is assigned when the object is inserted into the table. In addition to this unique identifier, each object can be considered to have a signature that identifies that object uniquely. The signature of the object can be generated from event data and looked up in the SYS_MDL_OBJECT table to find the corresponding unique object id. The signature can be uniquely generated from the attributes of the object in an event.

The general format for a signature is a list of all the successor objects from left (highest) to right (the final object), separated by forward slashes. In addition, the object type identifier (see table above) is a prefix to the signature since two objects of different types might otherwise have the same signature.

Signature Examples

Object Type	Example Signature
Host	1/macbeth (Object type/hostname)
Program Instance (Unix/NT)	4/U/2001080617592300000/132/1 (Object type/platform id/start time/process id/thread id)
Program Instance (CICS – z/OS)	2/C/CICS/ABCD/A0F1 (Object type/platform id/CICS region/transaction id/task id)
MQSeries Queue Manager	1001/qm1 (Object type/queue manager name)

MQSeries Queue (local)	1002/qm1/LOCAL.QUEUE (Object type/queue manager/queue)
MQSeries Queue (alias)	1003/qm1/ALIAS.QUEUE (Object type/queue manager/queue)



7.1.3. System Model Relationships

The following table shows the relationship between system model objects:

Relation Type	Relation Name	Examples of the Relationship
1	OWNS	• A host owns all the programs it hosts. • A program owns its program instances. • A queue manager owns all the queues it hosts. • A host owns all application servers it hosts. • An application server owns all web (enterprise) applications. • A web application owns all servlets, JSP, and EJB it contains. • An EJB owns all the methods it defines. • An IMS control region job owns transaction. • A z/OS job owns all its job steps. • A TIBCO connection owns TIBCO targets.
2	CONTAINS	A name List and its contents.

Relation Type	Relation Name	Examples of the Relationship
3	USES	• A queue uses a connection name. • A program uses a queue. • A program uses its EJB and servlets. • A CICS transaction uses programs. • A CICS program uses CICS PC programs. • A CICS program uses CICS files. • A CICS program uses CICS TD queues.
4	RESOLVETO	• An alias queue and the base queue it refers to • A remote queue and the queue it refers to • A model queue and the dynamic queue generated from it • A CICS TD queue and indirect queue
5	ABSTRACTS	• Cluster name and cluster object • Cluster object and cluster queue
6	ALIAS	• Program instance and MQSI message flow • Program instance and MQSI broker
7	ONE_TO_ONE	EJB entity beans relationship
8	ONE_TO_MANY	EJB entity beans relationship
9	MANY_TO_ONE	EJB entity beans relationship
10	MANY_TO_MANY	EJB entity beans relationship
11	STARTS	Two CICS transactions; one starts the other
12	BRIDGE_TO	TIBCO bridge source and target
13	ROUTE_TO	TIBCO route source and target
14	ROUTE_TO_FROM	TIBCO route source and target
15	DEPENDS_ON	Transaction class and program

7.1.4. System Model Attributes

For each system model object an arbitrary number of additional attributes can be stored in the SYS_MDL_OBECT_ATTR table. Each row in the table contains the object id of the corresponding system module object, and a name/value pair for the attribute and its value.

7.2. Event Tables

Data in the event tables is split up into three basic sections:

- The core event data
- The user data
- Lookup tables

The core event data contains a unique compound key identifying that event and an XML document, which contains the entire event data (minus user data which was not unmarshalled into XML.) The XML data gets stored in LOB columns. For performance reasons, the Analyzer can be configured to store the XML data into a VARCHAR column instead. Should the event XML data exceed the maximum size of this VARCHAR column, a separate row will be inserted into the EVENT_OVERFLOW table, which defines the event_data as LOB. To configure the Analyzer to use VARCHAR, edit the DatabaseDef.xml file in <TVISION_HOME>/config.datamgr and replace:

For ORACLE:

```
<Table name="EVENT">
        <Column name="event_data" type="CLOB" size="1M"/>
```

with the following:

```
<Table name="EVENT">
            <Column name="event_data" type="LONGVARCHAR"/>

If the Oracle datatype LONG cannot be used (it has officially been
deprecated by Oracle and is subject to certain restrictions), the
event data can alternatively be split up into two or more VARCHAR2
columns that have to be named data1, data2, …, dataN:

<Table name="EVENT">
            <Column name="data1" type="VARCHAR" size=”4000”/>
            <Column name="data2" type="VARCHAR" size=”4000”/>
            <Column name="data3" type="VARCHAR" size=”4000”/>
            <Column name="data4" type="VARCHAR" size=”4000”/>

The analyzer will split up the XML event document into several parts
and store each part into one of those columns. Whenever the XML
document is retrieved within TransactionVision, the full document is
reassembled again.
```

For DB2 and SQL Server:

```
<Table name="EVENT">
        <Column name="event_data" type="CLOB" size="1M"/>
```

with the following:

```
<Table name="EVENT">
            <Column name="event_data" type="VARCHAR" size="15000"/>
```

Note: This change will only improve performance if most of the events will fit into the LONGVARCHAR/VARCHAR column (thus minimizing the need to use the overflow table). In DB2 the maximum size for the VARCHAR is dependent on the database tablespace page size and should be determined by a DBA.

The PARTIAL_EVENT table is a temporary container for Entry- or Exit only events. If the corresponding partial event arrives in the Analyzer within a defined time interval, a

matching thread running in the Analyzer will merge those events and store them in the EVENT table as usual.

User data that was not unmarshalled into XML is stored in the USER_DATA table in the raw format (no data conversion). As with the XML event data, the Analyzer can be configured to use VARCHAR instead of BLOB columns. Edit the **DatabaseDef.xml** file in **<TVISION_HOME>/config.datamgr** and replace:

For ORACLE:

```
<Table name="USER_DATA">
<Column name="user_data" type="BLOB" size="10M"/>
```

with the following:

```
<Table name="USER_DATA">
<Column name="user_data" type="LONGVARBINARY"/>
```

Or alternatively (similar to EVENT) you can define the table with multiple RAW columns:

```
<Table name="USER_DATA">
<Column name="data1" type="VARBINARY" size=”2000”/>
<Column name="data2" type="VARBINARY" size=”2000”/>
<Column name="data3" type="VARBINARY" size=”2000”/>
<Column name="data4" type="VARBINARY" size=”2000”/>
```

For DB2 and SQL Server:

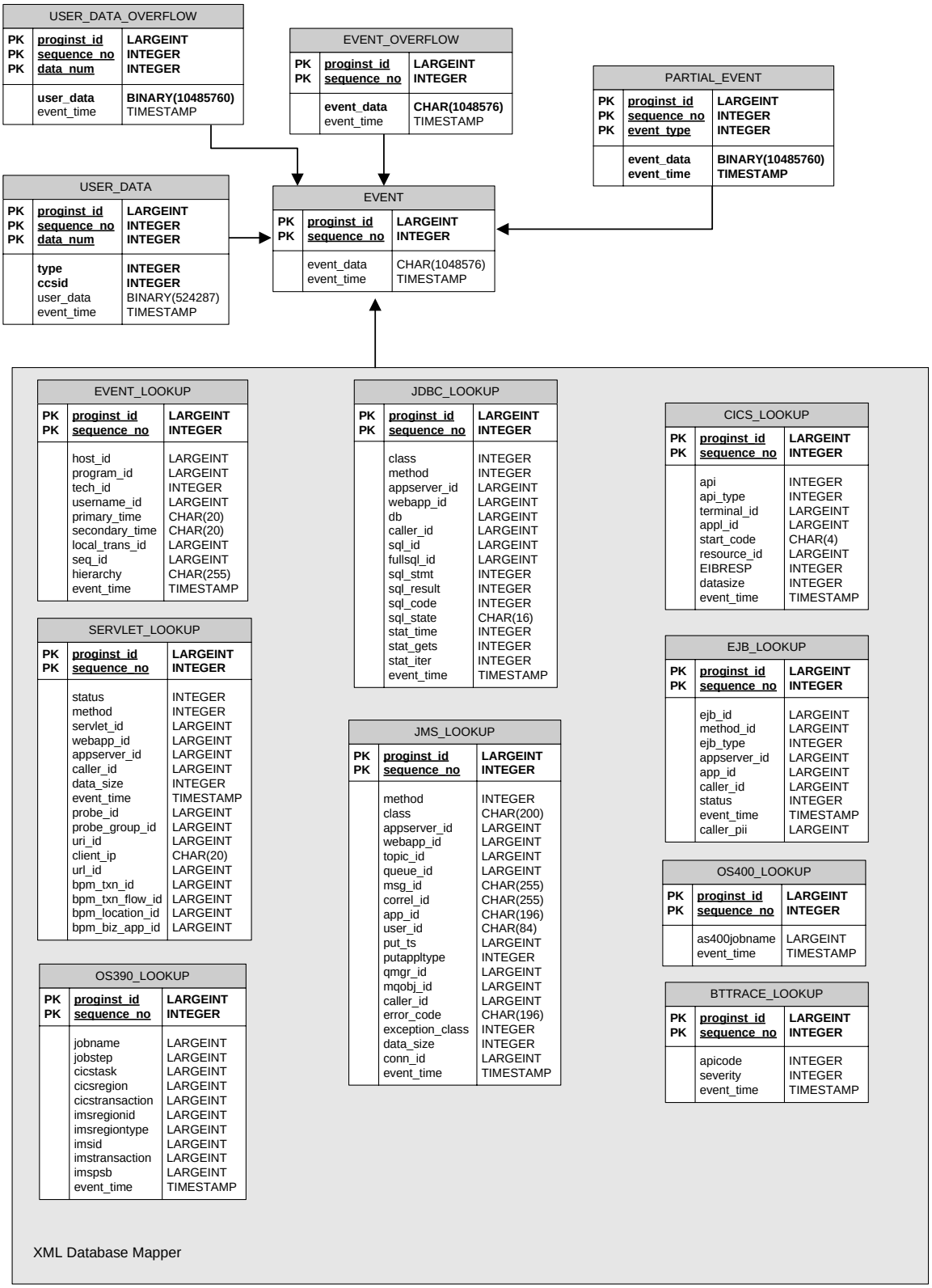
```
<Table name="USER_DATA">
<Column name="user_data" type="BLOB" size="10M"/>
```

with the following:

```
<Table name="USER_DATA">
<Column name="user_data" type="VARBINARY" size="15000"/>
```

Note that if any values or data types are changed in DatabaseDef.xml, the corresponding tables must be dropped and then re-created for the changes to take effect.

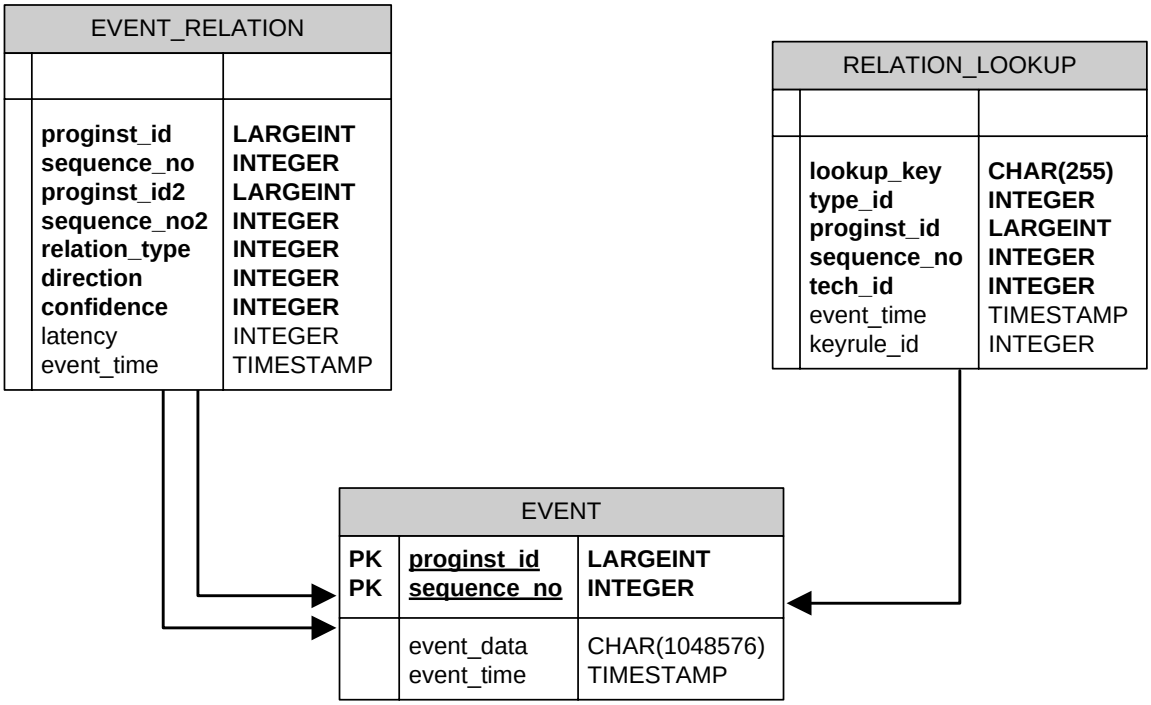
The lookup tables are used to store fields for quick searching; all columns in these tables are indexed. The XML to Database Mapping (XDM) file uses XPath statements to identify which data items are to be extracted from the XML event data and placed into the lookup tables. Lookup tables for the basic event data and the technology/platform specific event data are shown in the following figures.


USER_DATA_OVERFLOW
PK proginst_id LARGEINT
PK sequence_no INTEGER
PK data_num INTEGER
user_data BINARY(10485760)
event_time TIMESTAMP
EVENT_OVERFLOW
PK proginst_id LARGEINT
PK sequence_no INTEGER
event_data CHAR(1048576)
event_time TIMESTAMP
PARTIAL_EVENT
PK proginst_id LARGEINT
PK sequence_no INTEGER
PK event_type INTEGER
event_data BINARY(10485760)
event_time TIMESTAMP
USER_DATA
PK proginst_id LARGEINT
PK sequence_no INTEGER
PK data_num INTEGER
type INTEGER
ccsid INTEGER
user_data BINARY(524287)
event_time TIMESTAMP
EVENT
PK proginst_id LARGEINT
PK sequence_no INTEGER
event_data CHAR(1048576)
event_time TIMESTAMP
EVENT_LOOKUP
PK proginst_id LARGEINT
PK sequence_no INTEGER
host_id LARGEINT
program_id LARGEINT
tech_id INTEGER
username_id LARGEINT
primary_time CHAR(20)
secondary_time CHAR(20)
local_trans_id LARGEINT
seq_id LARGEINT
hierarchy CHAR(255)
event_time TIMESTAMP
JDBC_LOOKUP
PK proginst_id LARGEINT
PK sequence_no INTEGER
class INTEGER
method INTEGER
appserver_id LARGEINT
webapp_id LARGEINT
db LARGEINT
caller_id LARGEINT
sql_id LARGEINT
fullsql_id LARGEINT
sql_stmt INTEGER
sql_result INTEGER
sql_code INTEGER
sql_state CHAR(16)
stat_time INTEGER
stat_gets INTEGER
stat_iter INTEGER
event_time TIMESTAMP
CICS_LOOKUP
PK proginst_id LARGEINT
PK sequence_no INTEGER
api INTEGER
api_type INTEGER
terminal_id LARGEINT
appl_id LARGEINT
start_code CHAR(4)
resource_id LARGEINT
EIBRESP INTEGER
datasize INTEGER
event_time TIMESTAMP
SERVLET_LOOKUP
PK proginst_id LARGEINT
PK sequence_no INTEGER
status INTEGER
method INTEGER
servlet_id LARGEINT
webapp_id LARGEINT
appserver_id LARGEINT
caller_id LARGEINT
data_size INTEGER
event_time TIMESTAMP
probe_id LARGEINT
probe_group_id LARGEINT
uri_id LARGEINT
client_ip CHAR(20)
url_id LARGEINT
bpm_txn_id LARGEINT
bpm_txn_flow_id LARGEINT
bpm_location_id LARGEINT
bpm_biz_app_id LARGEINT
JMS_LOOKUP
PK proginst_id LARGEINT
PK sequence_no INTEGER
method INTEGER
class CHAR(200)
appserver_id LARGEINT
webapp_id LARGEINT
topic_id LARGEINT
queue_id LARGEINT
msg_id CHAR(255)
correl_id CHAR(255)
app_id CHAR(196)
user_id CHAR(84)
put_ts LARGEINT
putappltype INTEGER
qmgr_id LARGEINT
mqobj_id LARGEINT
caller_id LARGEINT
error_code CHAR(196)
exception_class INTEGER
data_size INTEGER
conn_id LARGEINT
event_time TIMESTAMP
EJB_LOOKUP
PK proginst_id LARGEINT
PK sequence_no INTEGER
ejb_id LARGEINT
method_id LARGEINT
ejb_type INTEGER
appserver_id LARGEINT
app_id LARGEINT
caller_id LARGEINT
status INTEGER
event_time TIMESTAMP
caller_pii LARGEINT
OS390_LOOKUP
PK proginst_id LARGEINT
PK sequence_no INTEGER
jobname LARGEINT
jobstep LARGEINT
cicstask LARGEINT
cicsregion LARGEINT
cicstransaction LARGEINT
imsregionid LARGEINT
imsregiontype LARGEINT
imsid LARGEINT
imstransaction LARGEINT
imspsb LARGEINT
event_time TIMESTAMP
OS400_LOOKUP
PK proginst_id LARGEINT
PK sequence_no INTEGER
as400jobname LARGEINT
event_time TIMESTAMP
BTTRACE_LOOKUP
PK proginst_id LARGEINT
PK sequence_no INTEGER
apicode INTEGER
severity INTEGER
event_time TIMESTAMP
XML Database Mapper

7.3. Event Relationship Tables

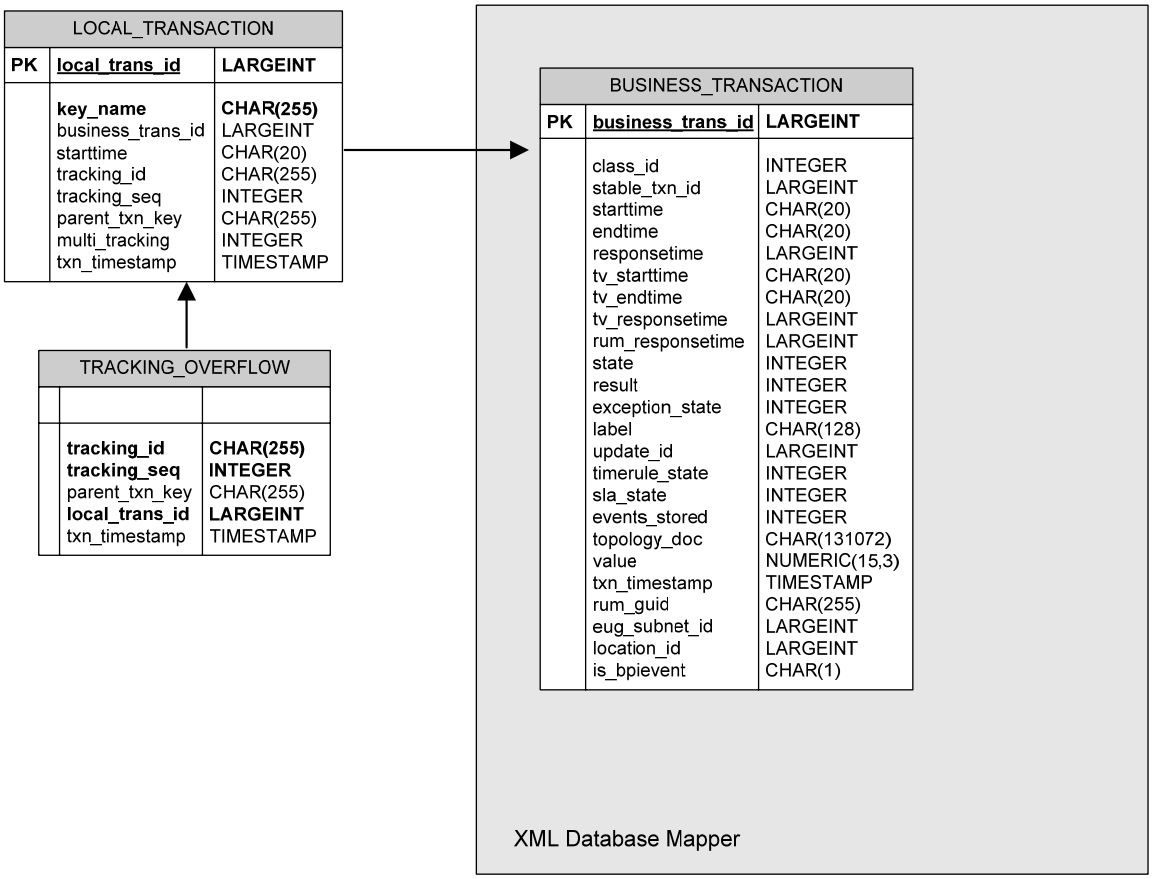
EVENT_RELATION table stores the relationship between two events determined by technology specific event correlation logic. If the relationship type is defined as BIDIRECTION, there will be two entries in this table: event1 -> event 2 and event2 -> event1. If the logic determines the two events are correlated in certain way with 100% certainty, the confidence factor is set to STRONG_RELATION, otherwise WEAK_RELATION.

RELATION_LOOKUP table stores a correlation lookup id for each event. The logic to generate this lookup id is specific to the technology used by this event.



7.4. Transaction Tables

Local and Business Transactions are created and updated during the Event Analysis phase in the Analyzer. The local transaction analysis populates the LOCAL_TRANSACTION table and links the event data to the corresponding transaction through the column local_trans_id in the table EVENT_LOOKUP. The BUSINESS_TRANSACTION table is defined through an XDM file and populated during business transaction analysis.



7.5. Statistics Tables

The statistics tables contain data used by various Reports in the TransactionVision UI/Job Server. The data in the TOPOLOGY_STATS and JDBC_STATS is collected by the Analyzer and used for the static Topology View and as a Datasource for event based reports. The BAC_SAMPLE_STATISTICS is used for delivering data samples to BSM.

7.5.1. Physical model

TOPOLOGY_STATS		
PK	**start_time**	**TIMESTAMP**
PK	**end_time**	**TIMESTAMP**
PK	**source_objid**	**LARGEINT**
PK	**dest_objid**	**LARGEINT**
PK	**link_objid**	**LARGEINT**
PK	**tech_id**	**INTEGER**
	msg_success	INTEGER
	msg_warn	INTEGER
	msg_error	INTEGER
	putget_success	INTEGER
	putget_warn	INTEGER
	putget_error	INTEGER
	byte_success	LARGEINT
	byte_warn	LARGEINT
	byte_error	LARGEINT
	min_latency	INTEGER
	max_latency	INTEGER
	avg_latency	REAL
	latency_count	INTEGER
	type	INTEGER

JDBC_STATS		
PK	**start_time**	**TIMESTAMP**
PK	**end_time**	**TIMESTAMP**
PK	**source_objid**	**LARGEINT**
PK	**dest_objid**	**LARGEINT**
PK	**link_objid**	**LARGEINT**
PK	**tech_id**	**INTEGER**
	min_latencycur	INTEGER
	max_latencycur	INTEGER
	avg_latencycur	REAL
	latency_countcur	INTEGER
	cursor_iter	INTEGER
	cursor_get	LARGEINT
	total_time	LARGEINT

BAC_SAMPLE_STATISTICS		
PK	**sample_timestamp**	**TIMESTAMP**
PK	**class_id**	**INTEGER**
	start_time	TIMESTAMP
	end_time	TIMESTAMP
	seq_no	INTEGER
	sum_response_time	LARGEINT
	sum_tv_response_time	LARGEINT
	sum_rum_response_time	LARGEINT
	max_response_time	LARGEINT
	max_response_time_txnid	INTEGER
	exp_tx_count	INTEGER
	late_tx_count	INTEGER
	failed_tx_count	INTEGER
	tx_count	INTEGER
	total_tx_value	LARGEINT
	total_failed_tx_value	LARGEINT
	total_exp_tx_value	LARGEINT
	total_late_tx_value	LARGEINT

7.6. RUM processing Tables

These tables are used for processing RUM events. The analyzer will process events sent by the RUM engine and update the TransactionVision business transaction data with the available RUM data (end user group information, end-to-end response times, etc). The RUM_LOOKUP table is defined via XDM and populated during RUM event processing. RUM_EVENT_RECOVERY and RUM_BUFFER_TABLE are used internally.

RUM_EVENT_RECOVERY		
	insertion_time	TIMESTAMP
	id	INTEGER
	data	BINARY(1048576)
	compressed	INTEGER

RUM_BUFFER_TABLE		
	rum_guid	CHAR(255)
	session_id	CHAR(50)
	timestamp	LARGEINT
	subnet_id	LARGEINT
	location_id	LARGEINT
	pagetime	LARGEINT
	creation_time	TIMESTAMP

RUM_LOOKUP		
PK	**proginst_id**	**LARGEINT**
PK	**sequence_no**	**INTEGER**
	client_starttime	CHAR(20)
	client_responsetime	LARGEINT
	session_id	CHAR(50)
	client_timestamp	TIMESTAMP

7.7. Other internal tables

These tables are used internally by the analyzer. The SCRATCH table is used for storing the analyzer recovery status, the ID_TABLE is used for id generation, the SEQUENCE_MAP and DIRTY_BIT tables for system model processing, and the BPI_BUFFER_TABLE for BPI event processing. The SCHEMA_VERSION table contains the the TransactionVision schema version number.

SCRATCH		
PK	**key_name**	**CHAR(30)**
	bvalue cvalue lvalue	BINARY(1048576) CHAR(1048576) LARGEINT

SEQUENCE_MAP		
PK	**sequence**	**CHAR(255)**
	id	**INTEGER**

ID_TABLE		
PK	**key_name**	**CHAR(30)**
	id	**INTEGER**

DIRTY_BIT		
PK	**module_id**	**INTEGER**
	status	**INTEGER**

BPI_BUFFER_TABLE		
	id **txn_id** expire_time type xml_data	**INTEGER** **LARGEINT** TIMESTAMP INTEGER CHAR(10240)

SCHEMA_VERSION		
	version	**INTEGER**

8. Event XML Schema

This section describes the various XML documents stored in TransactionVision database tables. XML schemas are used to describe TransactionVision data.

This chapter contains the following sections:

- 9.1. Basic Types
- 9.2. Event Schema Description

8.1. Basic Types

Basic types are technology specific data types and are described using schema tags xsd:simpleType or xsd:complexType. For example, MQMD belonging to the MQSeries technology may be described in a schema as:

```
<xsd:complexType name="MQMD">
      <xsd:sequence>
            <xsd:element name="StrucId" type="MQCHAR4"/>
            <xsd:element name="Version" type="MQLONG"/>
            <xsd:element name="Report" type="MQLONG"/>
            <xsd:element name="MsgType" type="MQLONG"/>
            <!-- and so on… -- >
      </xsd:sequence>
</xsd:complexType>
```

and the basic types MQCHAR4 and MQLONG are:

```
<xsd:simpleType name="MQCHAR4">
      <xsd:restriction base="xsd:string">
            <xsd:length value="4" fixed="true"/>
      </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="MQLONG">
      <xsd:restriction base="xsd:long"/>
</xsd:simpleType>
```

Similarly, all data types in a particular technology need to be described as above.

Technology specific methods such as MQGET, MQPUT etc. extend the “API” base type.

```
<xsd:element name="MQPUT">
      <xsd:complexType>
                  <xsd:sequence>
                        <xsd:element name="Hconn" type="MQHCONN"/>
                        <xsd:element name="Hobj" type="MQHOBJ"/>
                              <xsd:element name="pMsgDesc"
type="PMQMD"/>
                              <xsd:element name="BufferLength"
type="MQLONG"/>
                              <xsd:element name="pCompCode"
type="pMQLONG"/>
                              <xsd:element name="pReasonCode"
type="pMQLONG"/>
                        </xsd:sequence>
            </xsd:complexType>
```

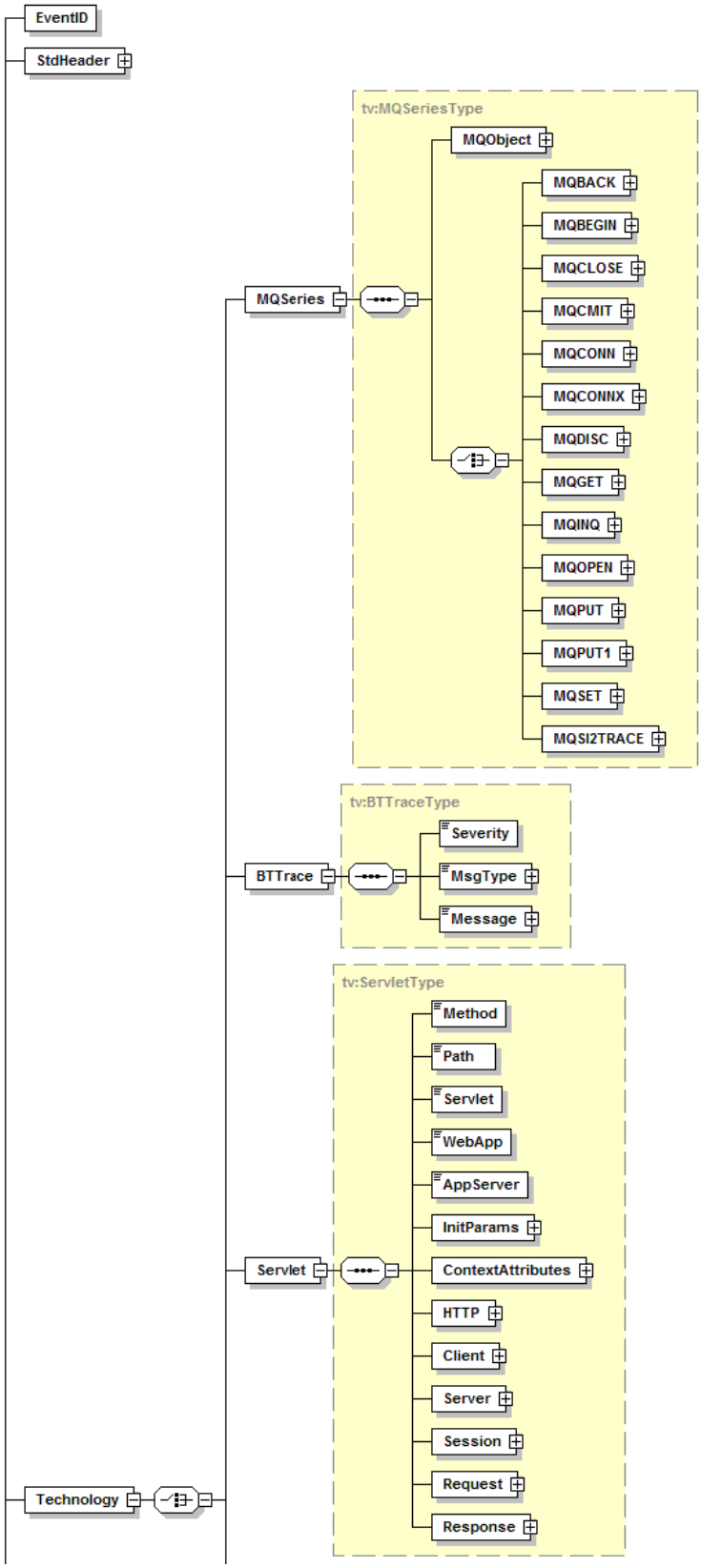
```
                    </xsd:element>
```

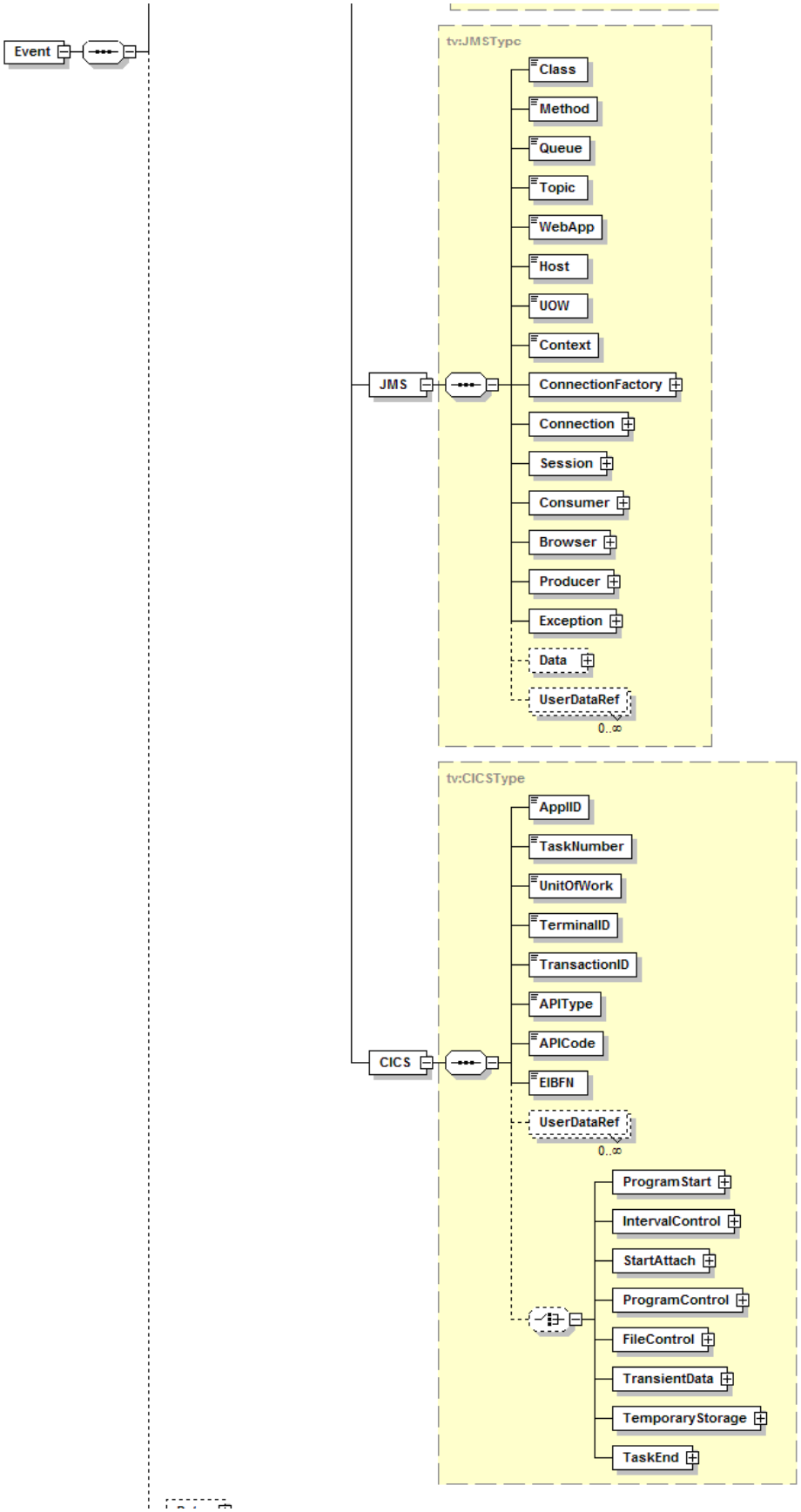
8.2. Event Schema Description

An event packet saved in the database would have the following layout: Detailed Schema definition can be found under **<TVISION_HOME>/config/xmlschema/Event.xsd**.

```
<?xml version="1.0" encoding="UTF-8"?>
<Event>
  <EventID programInstID="642" sequenceNum="7"/>
  <StdHeader minorVersion="1" uow="…" version="5">
    <HostArch>
      <OS>AIX</OS>
      <Vendor>IBM</Vendor>
      <HostArchValue>0xFFFFFFFF80030780</HostArchValue>
    </HostArch>
    <Encoding>273</Encoding>
    …
  </StdHeader>
  <Technology>
    <MQSeries API="MQPUT" … >
      <MQPUT>
        <MQPUTEntry>
          <HConn>0x5</HConn>
          <HObj>0x200EC268</HObj>
          <MQMD parameterName="MsgDesc" pointerValue="0x2FF22288">
            <StrucId>MQMD_STRUC_ID &quot;MD&quot;</StrucId>
            <Version>MQMD_VERSION_1 1</Version>
            <Report>MQRO_NONE 0</Report>
            <MsgType>MQMT_DATAGRAM 8</MsgType>
            …
          </MQMD>
          <MQPMO parameterName="PutMsgOpts" pointerValue="0x2FF223F8">
            <StrucId>MQPMO_STRUC_ID &quot;PMO&quot;</StrucId>
            <Version>MQPMO_VERSION_1 1</Version>
            <Options>MQPMO_NONE 0x0</Options>
            …
          </MQPMO>
          <BufferLength>25</BufferLength>
          <Buffer pointerValue="0x2FF2253C">
            <UserDataRef chunk="0"/>
          </Buffer>
          <CompCode pointerValue="0x2FF224FC">N/A</CompCode>
          <ReasonCode pointerValue="0x2FF22500">N/A</ReasonCode>
        </MQPUTEntry>
        <MQPUTExit>
          <HConn>0x5</HConn>
          <HObj>0x200EC268</HObj>
          …
        </MQPUTExit>
      </MQPUT>
    </MQSeries>
  </Technology>
  <Data>
    <Chunk blobType="0" ccsid="0" from="0" seqNo="0" to="24"/>
  </Data>
</Event>
```

The diagram below shows the basic structure of the type hierarchy of objects used to describe an event.


EventID
StdHeader
tv:MQSeriesType
MQObject
MQBACK
MQBEGIN
MQCLOSE
MQCMIT
MQCONN
MQCONNX
MQDISC
MQGET
MQINQ
MQOPEN
MQPUT
MQPUT1
MQSET
MQSI2TRACE
MQSeries
tv:BTTraceType
Severity
MsgType
Message
BTTrace
tv:ServletType
Method
Path
Servlet
WebApp
AppServer
InitParams
ContextAttributes
HTTP
Client
Server
Session
Request
Response
Servlet
Technology


Event
tv:JMSTypc
Class
Method
Queue
Topic
WebApp
Host
UOW
Context
JMS
ConnectionFactory
Connection
Session
Consumer
Browser
Producer
Exception
Data
UserDataRef
0..∞
tv:CICSType
AppID
TaskNumber
UnitOfWork
TerminalID
TransactionID
APIType
APICode
CICS
EIBFN
UserDataRef
0..∞
ProgramStart
IntervalControl
StartAttach
ProgramControl
FileControl
TransientData
TemporaryStorage
TaskEnd

9. The Data Manager

This chapter contains the following sections:

9.1. Using the DataManager to Access the Database

Custom beans and reports that need to access the database may use the service interface of the DataManager class to conveniently perform tasks which otherwise would have to be coded on the JDBC level.

A reference to the DataManager object can be obtained with the instance() method.

If the DataManager instance is used outside of the TransactionVision application context (for example, in a standalone Java application), the first call into the DataManager must be

```
DataManager.instance().init()
```

Beans and reports that run within the TransactionVision application are not required to do this; they can expect the instance to be successfully initialized.

Custom beans running within the TransactionVision Analyzer Framework will usually get the current database connection passed in as a parameter of class ConnectionInfo, which encapsulates the JDBC connection handle and the database schema name for the current processed event:

```
public class ConnectionInfo {

    /** The database connection */
    public Connection con;
    /** The database schema */
    public String schema;

    public ConnectionInfo(Connection con, String schema) ;
```

```
}
```

In cases where the custom code needs to obtain its own database connection, the DataManager offers three different methods for this purpose:

- getThreadConnection() will return a connection for the current thread. If this is the first time the thread calls into this method, a new connection to the database is returned. Every following call from the same thread will return the same connection, until it is getting released with releaseThreadConnection().
- getConnection() will always create and return a new connection to the database. This connection will get released with a call to releaseConnection(Connection con).

Interface

```
init
public static DataManager instance()
```

Returns the DataManager Singleton instance

Methods

Here is the complete list of the methods that make up the supported DataManager interface.

- init

```
public void init(java.lang.String propertyFile)
          throws com.bristol.tvision.datamgr.DataManagerException
```

  Initializes the DataManager according to the settings in the specified properties file. NOTE : This method has to be called before any other method.

  Parameters:

  dbProperties - The Database.properties file containing the db settings

  Throws:

  com.bristol.tvision.datamgr.DataManagerException - If initialization fails

- init

```
public void init()
          throws com.bristol.tvision.datamgr.DataManagerException
```

  Initializes the DataManager with the default properties file (Database.properties)

  Throws:

  com.bristol.tvision.datamgr.DataManagerException - If initialization fails

- getThreadConnection

```
public java.sql.Connection getThreadConnection()
                                          throws
com.bristol.tvision.datamgr.DataManagerException
```

  Returns the database connection for the current thread. If there is no connection stored in the connection map for this thread, a new connection is established by calling into the configured ConnectionSource, and this connection will be returned for all following calls.

  Returns:

  The database connection for the current thread

Throws:

com.bristol.tvision.datamgr.DataManagerException - if getting a new connection from the ConnectionSource fails

- releaseThreadConnection

```
public void releaseThreadConnection()
                                throws
com.bristol.tvision.datamgr.DataManagerException
```

Releases (closes) the connection for the current thread.

Throws:

com.bristol.tvision.datamgr.DataManagerException - if closing the connection fails

- getConnection

```
public java.sql.Connection getConnection()
                                     throws
com.bristol.tvision.datamgr.DataManagerException
```

Returns a new database connection which is not cached, which means every call into this method will obtain a new connection from the configured ConnectionSource.

Returns:

The database connection

Throws:

com.bristol.tvision.datamgr.DataManagerException - if getting a new connection from the ConnectionSource fails

- releaseConnection

```
public void releaseConnection(java.sql.Connection con)
                          throws
com.bristol.tvision.datamgr.DataManagerException
```

Close the connection which has been obtained from a call to getConnection.

Throws:

com.bristol.tvision.datamgr.DataManagerException - if closing the connection fails

- commitTransaction

```
public void commitTransaction(java.sql.Connection con)
                          throws
com.bristol.tvision.datamgr.DataManagerException
```

Performs a commit on current the database transaction

Parameters:

con - The connection holding the transaction to commit

Throws:

com.bristol.tvision.datamgr.DataManagerException - if the commit fails

- rollbackTransaction

```
public void rollbackTransaction(java.sql.Connection con)
                         throws
com.bristol.tvision.datamgr.DataManagerException
```

Performs a rollback on the current database transaction

Parameters:

con - The connection holding the transaction to roll back

Throws:

com.bristol.tvision.datamgr.DataManagerException - if the rollback fails

9.2. XML-Database Mapping Using XDM Files

The TransactionVision database schema is made extensible through the XML to Database Mapping (XDM) files. XDM is a generic way to describe the mapping of values contained in XML documents onto table columns in the database and allows fast, indexed XML data retrieval by the database engine.

The XML mapping is implemented by the class XMLDatabaseMapper and is used in TransactionVision to store the event and transaction data into lookup tables for fast retrieval. This class is also accessible from custom beans and reports and allows user written code to map basically any XML data to the database.

XML mappings are grouped into different ‘document types’. Each document type is defined by the root tag value for its documents and describes a mapping from XML to a set of database tables that logically belong together. These tables must share the same primary key, and the join across all these tables represents the mapped XML data for one XML document. In TransactionVision there are three predefined document types:

/Event

This document type consists of all event based XML mappings, including standard header event data, technology specific event data, and platform specific event data.

/Transaction

This document type maps data for the transaction analysis to the database tables.

/EventStatistics

This document type contains mappings for event statistics that are used for the topology view and various reports.

9.3. The XDM Syntax

XML mappings are defined in XDM files in the <TVISION_HOME>/config/xdm directory. The XML schema format of XDM files is defined in <TVISION_HOME>/config/xmlschema/XDM.xsd. Each XDM file defines a mapping of XML data to a particular database table. The syntax to describe this mapping is as follows:

<Mapping documentType="/Event">

Defines the document type for this mapping. This mapping is only valid for XML documents that have the same root tag as “documentType”.

<Mapping documentType="/Event" dbschema="SCHEMA1,SCHEMA2">

The dbschema attribute can specify one schema (or a list of schemas) for which the mapping is valid. The data insertion and retrieval methods of the XMLDatabaseMapper will not use this mapping if the supplied database schema parameter does not match. If this attribute is missing, the mapping is valid for all schemas. The <DBSchema> syntax of previous versions is still supported.

```
<Key name="proginst_id" type="BIGINT"
description="ProgramInstanceId">
<Path>/Event/EventID/@programInstID</Path>
</Key>
<Key name="sequence_no" type="INTEGER" description="SequenceNumber">
<Path>/Event/EventID/@sequenceNum</Path>
</Key>
```

Defines the primary key for the database table. All XDM mappings of the same document type must have the same key definition. There may be multiple key tags, in which case a compound primary key will get created. The structure of the key tag is similar to the Column tag and will be described there.

```
<Table name="EVENT_LOOKUP" categoryPath="COMMON">
```

Specifies the database table for the mapping. For mappings of the document type “/Event”, the XDM mappings can be technology or platform specific. The categoryPath attribute on the Table tag contains either “COMMON” to indicate that this table contains data common to every event and should be written for every event going through the Analyzer, or it can contain an XPath to the event document which is used as a criteria to decide if the mapping is applicable to the current event. If the “categoryPath” attribute contains an XPath, the attribute “categoryValues” contains a list of qualifying values from the event data. The standard event XDM mappings use XPaths to the event technology and to the event platform in the categoryPath attribute.

Examples:

```
<Table name="EVENT_LOOKUP" categoryPath ="COMMON">
...
</Table>
<Table name="MQSERIES_LOOKUP"
categoryPath="/Event/StdHeader/TechName" categoryValues ="MQSERIES">
...
</Table>
<Table name="OS390_LOOKUP" categoryPath=”
/Event/StdHeader/HostArch/OS”  categoryValues
="OS390_BATCH,OS390_CICS,OS390_IMS">
...
</Table>
```

If the categoryPath attribute is missing, the mapping is applicable to all events. Note that there has to be exactly one XDM mapping with categoryPath = “COMMON” for each document type.

```
<Column name="host_id" type="BIGINT" description="Host"
isObject="true">
<Path>/Event/StdHeader/Host/@objectId</Path>
</Column>
```

Each table mapping consists of several Column definitions that describe which XML value has to be mapped onto which database table column. The name attribute specifies the

column name, and the type attribute specifies the column type, which can be one of the following:

- INTEGER
- BIGINT
- FLOAT
- DOUBLE
- DECIMAL
- CHAR
- VARCHAR
- DATE
- TIMESTAMP

Both name and type are required. Types CHAR and VARCHAR require an additional attribute size.

Type DECIMAL requires additional attributes precision and scale.

The unicode attribute specifies that the character column should be generated in the database with the number of bytes defined for ‘unicode_bytes_per_character’ in Database.properties for each character. Default value if missing: ‘false’.

The subtype attribute can further refine the type of the column, Currently the only supported subtype is CURRENCY, which has to include the currency code, e.g. ‘subtype=CURRENCY(USD)’. See 10.3.1 for details on using currency values.

The description attribute specifies the name of the tag containing the value for that column in the query result document returned by the QueryService. Required.

The isObject attribute for a Column tag in the above XDM file refers to that column being an identifier for an object in the system model table. This allows to use the object name instead of the numerical, system generated object id in XDM based queries. Possible values: ‘true/false’. Default value if missing: ‘false’.

The generated attribute for a Column tag means that the column value will be generated by the DataManager.. Possible values: ‘true/false’. Default value if missing: ‘false’.

The conversionType attribute for a Column tag means that field requires a formatting conversion after reading from the database. The TypeConvService is called into after reading that field from the database. This is typically used for writing enumeration fields (conversionType=’enum’). Refer to the TypeConversionService for more information on how values are converted.

Additionally, an XDM column definition can be assigned a parameter named decimalFormat using a Param tag with a value set to a pattern of how to display a numeric value. When this column is read from the database and conversion is used, it will format a number according to the pattern given here. This pattern can be any pattern of the form supported by the java.text.DecimalFormat class. For example:

```
<Column name="value" type="DOUBLE" description="Value">
      <Param name="decimalFormat" value="$#.00"/>
      <Path>/Transaction/Value</Path>
</Column>
```

The indexed attribute specifies if a database index should be created for this column for faster query access. Possible values: ‘true/false’. Default value if missing: ‘true.

The complex attribute specifies that the Xalan XPath engine should be used instead of the built-in one for the document lookup. The built-in XPath search implementation is very efficient, but supports only a subset of the standard XPath syntax (see section 4.2 for details). If full XPath support is needed for a certain column, this attribute can be set. Note: the Xalan XPath implementation is much slower than the internal one and might slow down the analyzing process. Possible values: ‘true/false’. Default value if missing: ‘false’.

The xml attribute specifies that the XPath is pointing to an XML sub tree. The XMLDatabaseMapper will store the complete subtree as a full XML document into the corresponding column. Possible values: ‘true/false’. Default value if missing: ‘false’. Note: on ORACLE, LOB types are not supported for XDM column types. Use 'VARCHAR' or 'LONGVARCHAR' instead.

<Path> contains the XPath of the document value to write into the table column. The XMLDatabaseMapper will extract the value form the XML document and insert it into the database. Note that only XPaths pointing to Text nodes and attribute values are valid. If a value specified by the XPath does not exist in the XML document, a NULL value is inserted to the database.

A column can map to multiple XPath expressions as in the sample code below. The XPath expressions are evaluated in a sequential order and the first value found will get inserted into the database.

```
<Column name="msgid" type="CHAR" size="72" description="MessageID">

<Path>/Event/Technology/MQSeries/MQPUT/MQPUTExit/MQMD/MsgId</Path>

<Path>/Event/Technology/MQSeries/MQPUT1/MQPUT1Exit/MQMD/MsgId</Path>

<Path>/Event/Technology/MQSeries/MQGET/MQGETExit/MQMD/MsgId</Path>
</Column>
```

In addition to the <Path> element, a column definition can contain a <Join> definition like in the following example:

```
<Column name="class_id" type="INTEGER" description="ClassId">
<Path>/Transaction/ClassId</Path>
  <Join documentType="/TransactionClass"</Join>
</Column>
```

Join definitions offer a way to link two different document types together in order to use column definitions of both document types in one query. Internally this will generate a database join between the column of the current table and the primary key of the other table.

It is possible to store multiple values for one event into the database by defining an XDM table definition with the attribute type=”MultiValueExtension”. The table definition also requires an attribute basePath that specifies the base XPath for the event values that should be stored in the table. Let’s take the following event as an example:

```
<Event>

[..]

  <Data>
        <Chunk>
              <Account number=’1234’>
                    <Name>Miller</Name>
```

```
                    <Value>3340</Value>
                </Account>
                <Account number='4421'>
                    <Name>Smith</Name>
                    <Value>19000</Value>
                </Account>
                […]
            </Chunk>
    </Data>
</Event>
```

To store all account numbers for each event , the following XDM mapping has to be created:

```
<Mapping documentType="/Event">
  <Key name="proginst_id" type="BIGINT"
description="ProgramInstanceId">
        <Path>/Event/EventID/@programInstID</Path>
  </Key>
  <Key name="sequence_no" type="INTEGER"
description="SequenceNumber">
        <Path>/Event/EventID/@sequenceNum</Path>
  </Key>
  <Table name="ACCOUNTS" type="MultiValueExtension"
basePath="/Event/Data/Chunk/Account">
        <Column name="account_number" type="VARCHAR" size="20"
description="Account">
            <Path>/@number</Path>
        </Column>
  </Table>
</Mapping>
```

This will define a table 'ACCOUNTS' into which all account numbers found at the XPath /Event/Data/Chunk/Account/@number will be stored. The difference to a regular XDM table is that there can be multiple entries for a certain event in the table, the proginst_id/sequence_no columns are not a primary key any more.

NOTE: Multi-valued XDM tables are only useful for saving the data into the database tables, for later retrieval by custom SQL code. The TransactionVision query engine currently does not support any queries containing columns of multi-valued XDM mappings.

9.3.1. Currency columns

To handle monetary values accurately, it is recommended to use a DECIMAL data type with subtype CURRENCY in the XDM column definition. The subtype value indicates the default currency for the monetary column value with the three letter ISO-4217 currency code, e.g.:

```
<Column name="amount" type="DECIMAL" precision="15" scale="3"
description="OrderAmount" subType="CURRENCY(USD)">
<Path>/Transaction/OrderAmount</Path>
<Param currencyCodeClassXPath="/TransactionClass/CurrencyCode"/> <!-
- optional -->
 <Param currencyCodeXPath="/Transaction/OrderCurrencyCode"/> <!--
optional -->
</Column>
```

The currency for the column can be defined in three different ways:

- The currency can be defined by the value of another column in the transaction instance. This can be specified by using the parameter 'currencyCodeXPath' in the column

definition, which points to the XPath of the column containing the currency code for the transaction instance

- The currency can be defined by the value of a transaction class attribute. This can be specified by using the parameter 'currencyCodeClassXPath' in the column definition, which points to the transaction class attribute containing the currency code for the transaction instance. Note that every class has a default "currency_code" attribute which can be used for this purpose.
- If none of the above definitions exist, the currency code will be taken from the 'CURRENCY(…)' value in the column definition

The currency code for a business transaction instance can be used programmatically for the following purposes:

- if the value of a column with subtype CURRENCY is retrieved via a cursor with conversion service, the cursor will convert the currency code to the currency symbol (if available).
- the currency code determined for a transaction instance can be obtained in code (e.g. in a java action) via

  ```
  public String getCurrencyCode(String XPath) throws
  DataManagerException;
  ```

  in class XMLTransaction

- The value of a curreny column can automatically be converted from one currency to another via classification, by using a 'CurrencyConversionAction'. Here are the setup steps required for using this feature:
  1. Manually insert up-to-date conversion factors into the table CURRENCY_CONV in schema TVISION. Each row in this table contains a 'From' code, a 'To' code, and the factor to convert from 'From' to 'To'. Note that the factors do not work in reverse, so if you e.g. have a row ('USD', 'EUR', 0.79), and you also need conversion from EUR -> USD, you will also need to enter a row ('EUR', 'USD', 1.26)
  2. In the Transaction Definition Editor, define a currency action on the transaction attribute whose value is supposed to get converted, with the following properties:

     ```
     Action class name:
     com.bristol.tvision.services.analysis.actions.CurrencyConver
     sionAction
     ```

     Reason: the 3-letter currency code from which to convert

Now, whenever the attribute value is set during classification, the following will happen:

- The value determined by the value rule will be interpreted to be of the currency defined in 'reason'
- The action rule will use the conversion factors in table CURRENCY_CONV to convert the attribute value into the currency defined by XDM definition (as described above)
- The transaction attribute will be set to the converted value

Example:

A custom ‘OrderAmount’ business transaction attribute has been defined which is supposed to track monetary values in US dollar. The value will be set from a field in the XML payload of the event via a classification rule, but this value is based on Euro. A currency conversion action is used to convert the amount from EUR to USD before the value is stored in the attribute:

XDM definition:

```
<Column name="amount" type="DECIMAL" precision="15" scale="3"
description="OrderAmount" subType="CURRENCY(USD)">
<Path>/Transaction/OrderAmount</Path>
<Param currencyCodeClassXPath="/TransactionClass/CurrencyCode"/>
</Column>
```

Attribute rule definition:

```
<Attribute>
  <Path>/Transaction/OrderAmount</Path>
  <ValueRule>
  <Value type="XPath">/Event/Data/Chunk/Order/Amount</Value>
  </ValueRule>
  </Attribute>
```

Action rule on attribute ‘OrderAmount’:

Class: com.bristol.tvision.services.analysis.actions.CurrencyConversionAction

Reason: EUR

Contents of the CURRENCY_CONV table:

‘EUR’, ‘USD’, 1.26

Result:

Once the attribute rule for ‘OrderAmount’ fires, the value will be retrieved from /Event/Data/Chunk/Order/Amount, converted to USD in the action rule by multiplying it with 1.26, and stored in the ‘amount’ column in the business transaction table.

Note: the table CURRENCY_CONV is only read once at analyzer startup, so the analyzer needs to be restarted if the table has been updated.

9.3.2. Creating the XDM Database Tables

One important aspect of the XDM framework is that the creation of the underlying database tables is entirely data-driven. The definitions in the XDM files are not only being used for updating or querying the XML data, but also as an input to the TransactionVision Table Manager, which is responsible for creating and dropping the project tables as projects in the Analyzer GUI get created and deleted. Thus there is no need to issue any SQL DDL calls to the database. Once the XDM file is placed into the proper directory, and provided the document type is registered with the Table Manager, the new tables defined in the XDM mapping get automatically created for a new project. The same holds true if the project tables get created or dropped by using the command line tool CreateSqlScript.

The registration with the Table Manager is only needed if the XDM mapping uses a new user defined document type. The only thing to do is to add the new document type to the following section of the DatabaseDefinition.xml in the **<TVISION_HOME>/config/datamgr** directory:

```
<XDM>
    <DocumentType>/Event</DocumentType>
    <DocumentType>/Transaction</DocumentType>
    <DocumentType>/EventStatistics</DocumentType>
    <DocumentType>/MyNewDocType</DocumentType>
```

```
</XDM>
```

9.3.3. Properties of the TransactionVision Document Types

The /Event Document Type

Event-based XDM files specify that when an XML event is written to the database by the DBWrite module in the Analyzer, these fields are extracted and written into the database columns defined by the XDM mappings. Similarly, when the database is queried to retrieve event based data in the Analyzer GUI, these XDM files are used to construct the corresponding SQL query. The XML document for each event gets stored in the database table EVENT.

The /Transaction Document Type

This mapping is used to write business transaction attributes during the transaction analysis phase in the Analyzer. One noticeable difference to the event-based mappings is that there is no XML document inserted into the database, all document values are always mapped to the database tables. Note that you can define XDM based queries that combine both transaction and event document types.

The /EventStatistics Document Type

This document type contains XDM mappings for the event statistics data generated during analyzer processing that is used for the static topology view and other various reports. Note that it is not possible to link this document type to the event or transaction document types.

9.4. The XMLDatabaseMapper Interface

The XMLDatabaseMapper can be used in 2 different ways: implicitly when writing custom bean code in the Analyzer bean framework or using the query facilities of the QueryService, or explicitly by obtaining a reference to an XMLDatabaseMapper instance and calling into one of the available service methods.

To obtain a reference to an instance, the instance() method has to be called with the particular schema as an argument, e.g.:

```
XMLDatabaseMapper xdm = XMLDatabaseMapper.instance(mySchema);
```

The interface contains methods for reading, inserting, updating, and deleting XML values. All methods take a parameter of class XMLDocument, which denotes the XML document containing the data. The XMLDocument class implements the org.wc3.dom.Document Interface and can be constructed in several ways: from an existing document using the constructor XMLDocument(org.w3.dom.Document doc), or entirely bypassing the generation of any XML objects and creating a ‘lightweight’ XMLDocument instance by using the constructor XMLDocument(java.util.Map).

The class contains an internal HashMap for caching XPath expressions to the corresponding values in the XML document. The key of the map entry is an XPath expression, the value of the map entry is the value in the XML document corresponding to that XPath. If an instance is created by using the latter constructor, then any value lookup on the document translates into a simple HashMap lookup, whereas a lookup on an instance created with the first constructor is performed by executing an XPath search on the XML document (unless the corresponding XPath is already in the cache). This is implemented transparently for the caller by the following method of XMLDocument:

```
public String getDocumentValue(String xpath) throws XMLException;
```

If there is a value for the given XPath in the HashMap, the stored value is returned. Otherwise an XPath search on the document is performed.

With these ‘lightweight’ XML documents it is possible to provide data to the XMLDatabaseMapper without having to make expensive XML operations. The XMLTransaction class used in the transaction analysis is one example of such a ‘lightweight’ XML object.

Methods

Following is the list of available XMLDatabaseMapper methods.

- read

```
public void
read(com.bristol.tvision.datamgr.ConnectionInfo conInfo,
                 com.bristol.tvision.shared.xml.XMLDocument doc)
          throws com.bristol.tvision.datamgr.DataManagerException
```

  Reads all lookup table rows for the given key values and store the values in the attribute map of the XML document. The document passed in only needs to contain the key values.

  Parameters:

  con - The database connection to use

  doc - The document containing the key values

  Throws:

  com.bristol.tvision.datamgr.DataManagerException - Error while accessing the document or reading from the database tables

- write

```
public void
write(com.bristol.tvision.datamgr.ConnectionInfo conInfo,
                  com.bristol.tvision.shared.xml.XMLDocument doc)
           throws
com.bristol.tvision.datamgr.DataManagerException
```

  Writes the values of the mapped document elements to the lookup tables. For each mapped column defined in the xdm files, the value of the corresponding XPath expression is searched in the xml document and written to the table column defined in the mapping.

  Parameters:

  con - The database connection to use

  doc - The document to search

  Throws:

  com.bristol.tvision.datamgr.DataManagerException - Error while accessing the document or writing to the database tables

- update

```
public void
update(com.bristol.tvision.datamgr.ConnectionInfo conInfo,

com.bristol.tvision.shared.xml.XMLDocument doc)
            throws
com.bristol.tvision.datamgr.DataManagerException
```

Updates the values of the mapped document elements in the lookup tables. All columns that are defined by the document type will get updated. The rows to update are determined by the key values in the XML document.

Parameters:

con - The database connection to use

doc - The document containing the updated values

Throws:

com.bristol.tvision.datamgr.DataManagerException - Error while accessing the document or writing to the database tables

- delete

```
public void
delete(com.bristol.tvision.datamgr.ConnectionInfo conInfo,

com.bristol.tvision.shared.xml.XMLDocument doc)
            throws
com.bristol.tvision.datamgr.DataManagerException
```

Deletes rows in all lookup tables of the document type for the given key values in the XML document.

The document passed in only needs to contain the key values.

Parameters:

con - The database connection to use

doc - The document containing the key values

Throws:

com.bristol.tvision.datamgr.DataManagerException - Error while accessing the document or writing to the database tables

9.5. Extending the /Event Document Type

The XDM mappings of the /Event document type can be easily extended to map additional XML data to indexed database columns for faster retrieval. First, this can be done for XML values that are already present in the standard XML event data but which are not included in the default event based XDM mapping definitions. In this case the mapping for the desired values can be simply added (with its XPath and database column) to the corresponding XDM file (event.xdm. mqseries.xdm, etc.).

Second and more important, additional mappings can be defined for XML data that has been assembled from the contents of the user data buffer by an EventModifierBean (see chapter 3.2). Although this user defined XML data could also be mapped to the existing lookup tables (by simply modifying one of the existing XDM files), this is not advisable. For this purpose a new XDM file defining a mapping to a new table should be created. The mapping definition is required to have the document type /Event and the key columns proginst_id and sequence_no like all other event based XDM files. The column definitions should include all XDM values intended for display in the Analyzer GUI or queries through the query services. For steps to configure the Analyzer GUI to display these new columns see Chapter 3.

The TransactionVision DeleteEvents utility and job use an optimized fast deletion scheme based on timestamp columns if the –older option is used. To delete data in user-defined

XDM tables, the timestamp column must be present in any additional XDM mapping you define. Therefore, the following section is mandatory in the XDM file:

```
<Column name="event_time" type="TIMESTAMP" description="EventTime">
        <Path>/Event/EventTimeTS</Path>
</Column>
```

9.6. Extending the `/Transaction` Document Type

The /Transaction document type can be extended to add custom business transaction attributes to the transactional data in TransactionVision. See chapter 3.5.4 for details.

The TransactionVision DeleteEvents utility and job use an optimized fast deletion scheme based on timestamp columns if the –older option is used. To delete data in user-defined XDM tables, the timestamp column must be present in any additional XDM mapping you define. Therefore, the following section is mandatory in the transaction document type:

```
<Column name="endtime" type="CHAR" size="20" description="EndTime"
conversionType="Date">
        <Path>/Transaction/EndTime</Path>
</Column>
```