IBM®

developerWorks.

External application integration with IBM Content Manager through a custom event handler

Skill Level: Intermediate

Alan Yaung (ayaung@us.ibm.com)
Senior Software Engineer
IBM

09 Jul 2009

IBM® Content Manager, Version 8.4.1 supports an event infrastructure that enables the integration of external applications. A set of message formats are published for the event messages generated from an event monitor. The general integration for an external application is made possible by using a custom event handler. Because the event message informs the custom event handler of the content operations in the repository, the custom event handler can interact with external applications based on the content-aware business logic. This article provides an overview of the event framework and uses an example e-mail application to illustrate how to write a custom event handler for external application integration.

Introduction

IBM Content Manager, Version 8.4.1 supports an event framework for external application integration. Two types of integration are currently supported:

- Process integration, specifically for the integration with FileNet Business Process Manager
- General integration, for the integration with general purpose external applications

An event handler is provided with IBM Content Manager, Version 8.4.1 (hereafter referred to as CM8) for process integration. For integration with external applications other than FileNet® Business Process Manager, you can use the general integration option and develop a custom event handler to facilitate specific application logic in a

© Copyright IBM Corporation 2009. All rights reserved.

business solution. This article describes how to design a custom event handler for an external application in general integration.

The following sections provide an overview of the event framework and various aspects of a custom event handler. The example helps you write a custom event handler for an e-mail application with IBM Content Manager.

Architecture of an event framework

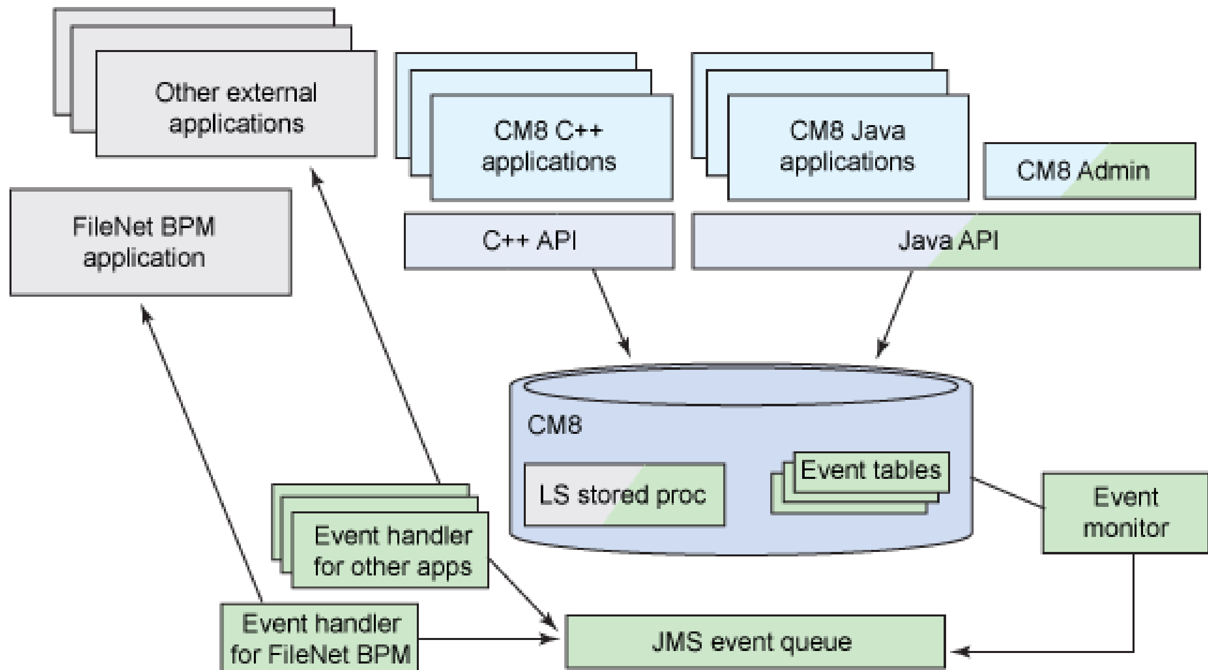
Figure 1 illustrates the architecture of an event framework that supports the integration of IBM Content Manager and external applications. Because the process integration has already been addressed with an event handler for FileNet Business Process Manager, this article focuses mainly on the integration with general purpose external applications.

The event framework provides the following three capabilities:

- Event subscription, which allows you to identify an item type and its associated events to be monitored.
- Event monitoring, which logs the events when they occur and generates event messages that are sent to a Java Message Service (JMS) message queue.
- Event processing, which retrieves the event messages from the JMS queue and processes these messages based on business logic.

Figure 1. Event framework architecture

© Copyright IBM Corporation 2009. All rights reserved.



During event subscription, the event subscription data is stored in the configuration tables in a CM8 library server database. The configuration data is configured by using the CM8 system administration client. Administrators subscribe the events to be monitored for a specific item type and its attributes.

The event monitoring part focuses on the event message generation. Through CM8 Java and C++ API interfaces, CM8 applications interact with the library server, generating event data in the database. The library server logs the events when they occur according to the configuration data. The logged events are placed into an event queue table in the library server database. An event contains event type, event ID, item information, and attribute data. The event monitor fetches event data from the event queue table based on the configuration data, converts these events into Java Message Service (JMS) messages, and puts these JMS messages on a JMS event queue.

The event processing part concentrates on the event message consumption. An event handler reads JMS messages, which carry the event data, from the JMS event queue and provides the ability to integrate application logic with CM8 document attributes. Even though an event handler for process integration is provided with the IBM Content Manager, Version 8.4.1 for FileNet Business Process Manager, custom event handlers would need to be developed for integration with other applications.

A scenario

The electronic mail has been a simple and effective way to collaborate in a business environment. The objective of the scenario is to illustrate how to incorporate the

© Copyright IBM Corporation 2009. All rights reserved.

e-mail notification capability into this event framework in a business context.

A fictitious XYZ Insurance Company requires the claim adjuster to be automatically notified through an e-mail as soon as an auto claim form is submitted for processing by an insurance agent. During the claim submission, an insurance agent provides the claim information, such as claim number, claim amount, and policy ID. In addition, he specifies a notification list of claim adjusters who are requested to review the claim.

Furthermore, after the claim adjuster receives the e-mail notification, he can follow through the URL link in the e-mail to view the claim information, including the claim form document. Next, he can update the status of the claim based on his assessment. For a claim adjuster, he just needs to open an e-mail and follow the URL to do the claim review. This is a common requirement with the e-mail integration.

The following scenario illustrates the integration of an e-mail application with IBM Content Manager that uses a document creation case as an example. (This scenario is used to take you through various steps in later sections.)

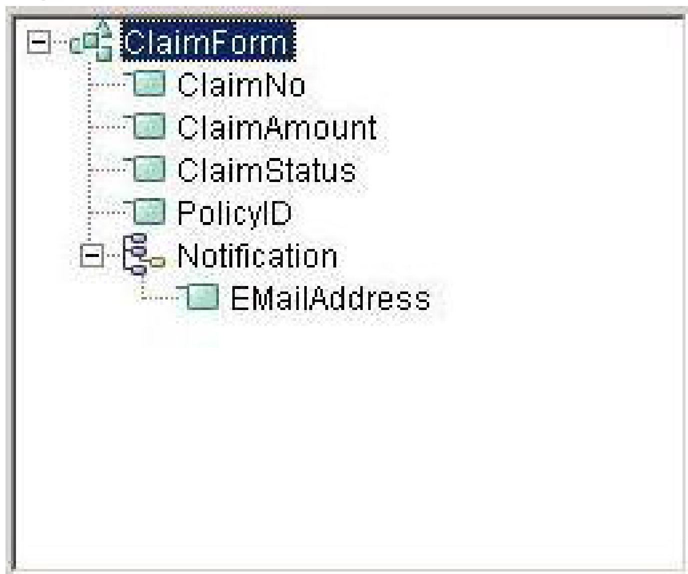
- A claim document of item type "ClaimForm" is created in IBM Content Manager by an insurance agent.
- Because the administrator configured this item type "ClaimForm" to monitor upon an "Add Item" event for document creation, the library server inserts a row of event data, including attribute values such as "ClaimNo", "ClaimAmount", and "Notification", into the event queue table.
- An event monitor scans the event queue table for committed rows and retrieves this "Add Item" event for document A. The event monitor also retrieves from a configuration table for configuration data. It converts the configuration information along with the event data retrieved from the event queue table into a self-describing event message in a common base event format (in other words, CBE format). The event message is then sent to a Java Message Service queue (in other words, JMS queue).
- A custom event handler, which is listening to the JMS queue, retrieves the document creation event from the queue. Next, it parses the event data from the CBE format and then uses the information to send an e-mail to the claim adjusters on the notification list.
- The claim adjuster receives the e-mail notification and reviews the claim document. He can then use a CM8 application to update the status of the claim.

A ClaimForm item type for auto claim processing is defined in IBM Content Manager. As shown in Figure 2, it consists of the following attributes:

© Copyright IBM Corporation 2009. All rights reserved.

- ClaimNo - Claim number as a unique identifier of the claim.
- ClaimAmount - The amount of the claim.
- claimStatus - The status of the claim.
- PolicyID - The policy ID associated with the claim.
- Notification - A notification list of e-mail addresses for the reviewers.

Figure 2. A claim form item type



It also has a document part, which contains the claim form (see Figure 3).

Figure 3. Document part definition

Part type	Access control list	Resource manager	Collection	Version
ICMBASE	DocRouteACL	rmdb	CBR.CLLCT001	Never create

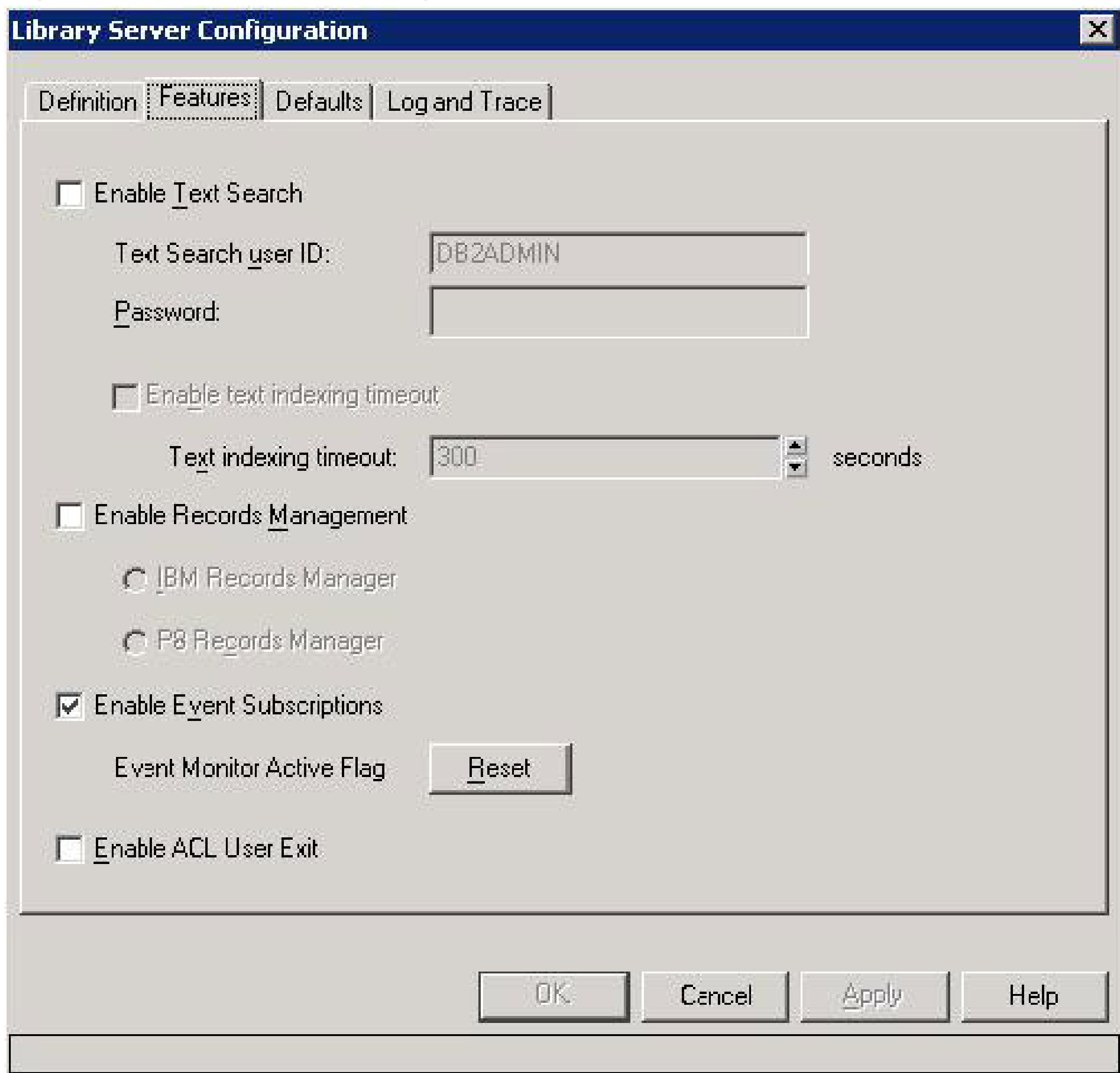
Event subscription for general integration

This section describes the steps for event subscription for general integration. The

© Copyright IBM Corporation 2009. All rights reserved.

event subscription can be enabled at two levels: library server level, and item type level for better granularity. You must select the "Enable Event Subscription" check box in the Library Server Configuration dialog (see Figure 4) to enable the logging of events for a library server instance.

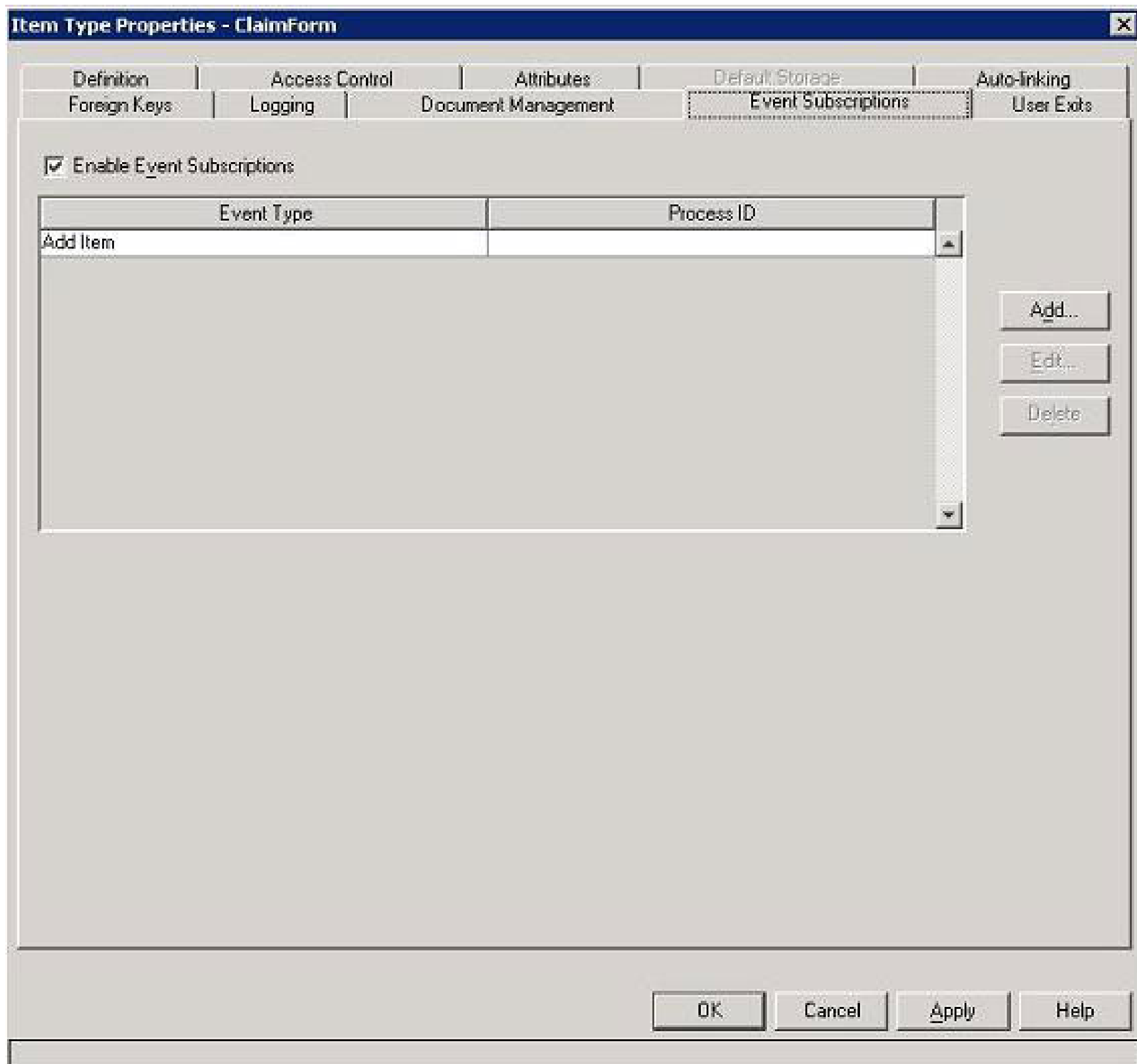
Figure 4. Library server configuration



At the item type level, Figure 5 shows an example of event subscription of an item type "ClaimForm". (The "ClaimForm" item type was described in Figure 3.) The event to be subscribed for this item type is an "Add Item" event. The "Event Subscription" page provides the support of the item level event subscription.

Figure 5. Event subscription

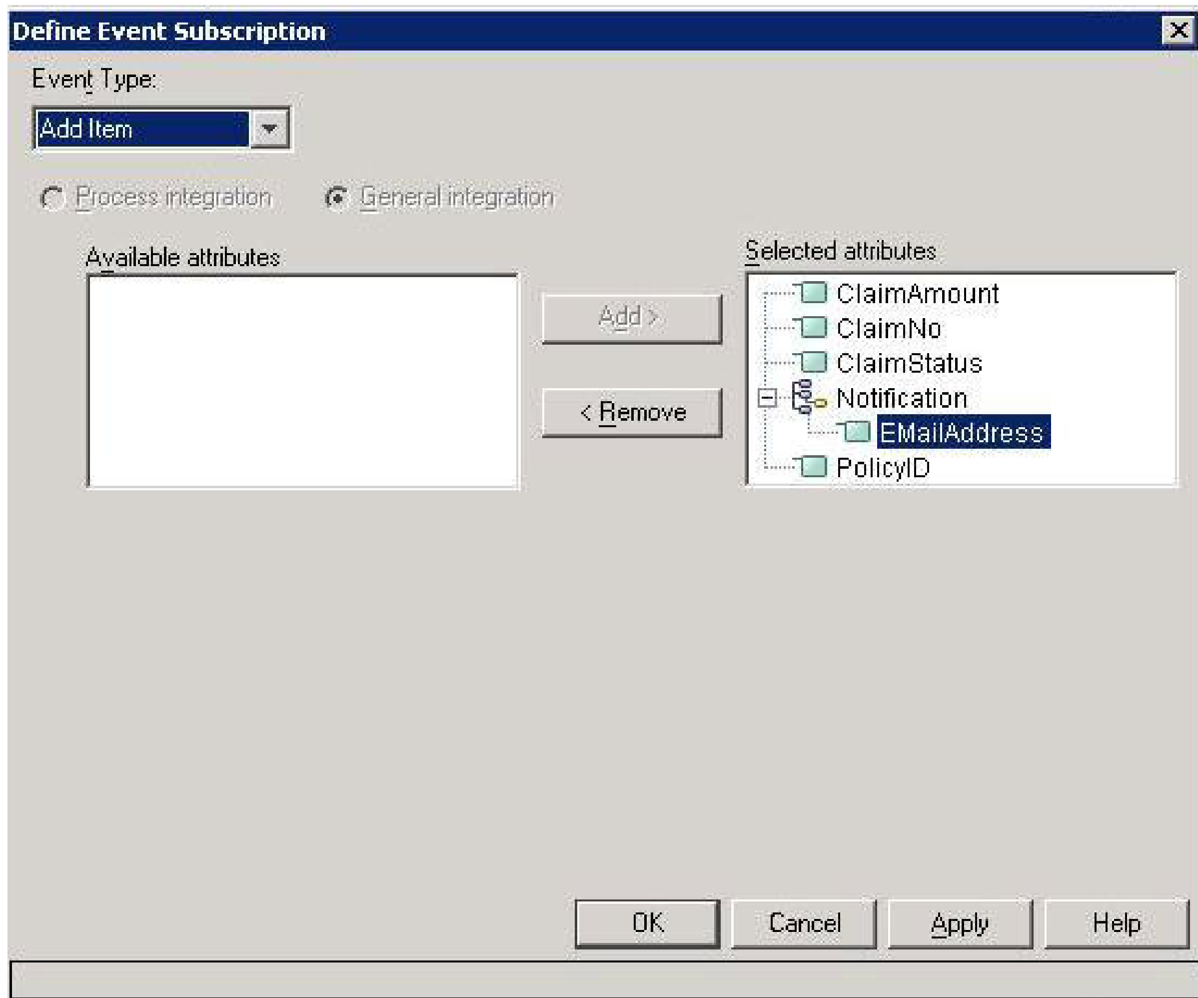
© Copyright IBM Corporation 2009. All rights reserved.



When you click on the **Add** button in the "Event Subscription", a "Define Event Subscription" dialog is displayed (see Figure 6). Select the **General integration** button since the scenario is related to a general purpose external application. All document attributes of this "ClaimForm" item type are selected to be monitored in the event monitoring for this "Add Item" event.

Figure 6. Add an event

© Copyright IBM Corporation 2009. All rights reserved.



Design of a custom event handler

Essentially, a custom event handler is a JMS application supporting the business logic described in the scenario. It uses an asynchronous messaging delivery mechanism, and a message listener is implemented to receive a newly arrived message in the JMS queue. The basic functions of a custom event handler are to consume the event messages in the JMS queue and to interact with external applications accordingly. According to the scenario, the handler must connect to an IBM Content Manager to retrieve the document attributes of the item referenced in the event message. Furthermore, the handler uses the JavaMail API to compose an e-mail notification, which is sent to the claim adjusters for review.

The code sample in Listing 1 shows the program structure of an implementation of the custom event handler for the scenario. First, the required packages that are imported are:

© Copyright IBM Corporation 2009. All rights reserved.

- Accessing a JMS queue - JMS and Java Naming and Directory Interface (JNDI) packages
- Sending an e-mail - the JavaMail package
- Parsing the event message - the SAX API package
- Retrieving content data - the Content Manager API package

The `myEventHandler` class implements two JMS listeners: the `onMessage` method for the javax.jms.MessageListener interface to listen to the JMS queue, and the `onException` method for the javax.jms.ExceptionListener interface to handle JMS exceptions.

Two inner classes are defined for parsing the event messages. The `myContentHandler` class extends the `DefaultHandler` class of SAX API to parse the event message, and the `myErrorHandler` class implements the ErrorHandler interface of SAX API.

The `sendMail` method constructs the e-mail to be sent to the claim adjuster by using JavaMail API. It is called by the `onMessage` method when an event message is received.

The `waitForQUIT` method prompts for the quit command from the user. The `myEventHandler` class exits when a quit command is received. The `waitForQUIT` method is called when the `myEventHandler` class is instantiated.

Listing 1. Program structure

```
// JMS API
import javax.jms.*;
// JNDI API
import javax.naming.*;
// Java IO and utility
import java.io.*;
import java.util.*;
// SAX and XML parsing
import org.xml.sax.*;
import org.xml.sax.helpers.*;
// JavaMail API
import javax.mail.*;
import javax.mail.internet.*;
// CM API
import com.ibm.mm.sdk.server.*;
import com.ibm.mm.sdk.common.*;

public class myEventHandler extends Thread
             implements javax.jms.MessageListener, javax.jms.ExceptionListener
{

    static class myContentHandler extends DefaultHandler
    {
        // methods in the myContentHandler class
    }
```


© Copyright IBM Corporation 2009. All rights reserved.

```
    static class myErrorHandler implements ErrorHandler
    {
        // methods in the myErrorHandler
    }

    public void run()
    {
        this.waitForQUIT();
    }

    private void waitForQUIT() throws Throwable
    {
        // prompt for the QUIT command to exit
    }

    public myEventHandler(String database, String userid, String pw) throws Throwable
    {
        // set up JMS connection
        // initialize SAX parser
        // start the thread for waiting the QUIT command to exit
    }

    public static void main(String[] argv)
    {
        // start the myEventHandler instance
        myEventHandler eh = new myEventHandler(argv[0], argv[1], argv[2]);
    }

    public void onMessage(javax.jms.Message message)
    {
        // listen to the JMS queue
        // retrieve a JMS message
        // parse the JMS message to obtain the event data
        // process the event data
        // compose an e-mail
        // call the sendMail function to send out an e-mail notification
    }

    public void onException(JMSException exc)
    {
        // print the error message when a JMS exception occurs
    }

    public void sendMail(String mailhost, String from, String[] to,
                         String subject, String text)
    {
        // send an e-mail notification to the claim reviewers
    }
}
```

The following sections describe the main portions of the sample program (included in the Download section):

- Connect to a JMS queue
- Retrieve the event data in an event message
- Parse the event data in an event message
- Retrieve content data referenced by the event from a CM8 server
- Compose an e-mail based on the content data

© Copyright IBM Corporation 2009. All rights reserved.

- Send the e-mail to the claim adjusters

Connect to a JMS queue

Listing 2 illustrates the steps for setting up the access to a JMS queue. First, the program retrieves a queue connection factory by looking up the context object with a queue connection factory name. From the queue connection factory, a queue connection is created. Next, a queue session is created from a queue connection, and a queue is retrieved by looking up the context object with a queue name. Then a queue receiver is created for a specified queue as a consumer of messages. The queue receiver listens to the queue with a message selector `APPLICATION = 'none'`. Only the messages with a message property APPLICATION and a property value of 'none' are retrieved from the JMS queue. `APPLICATION = 'none'` indicates that the message is from general integration.

After the queue receiver is available, the program sets the message listener in the queue receiver. The exception listener is set in the queue connection. Before starting the JMS connection, the program performs two additional tasks:

- Initializes the SAX parser for parsing the event message
- Connects to the Content Manager to retrieve the content data

The last step starts the JMS connection. After the JMS connection is started, the `onMessage` message listener and the `onException` exception listener are ready for the incoming messages from the JMS queue.

Listing 2. Connect to a JMS queue

```
        // obtain the queue connection factory
        factory = (QueueConnectionFactory)ctx.lookup(_qcf_name);
        // create a queue connection
        connection = factory.createQueueConnection();
        System.out.println("Queue connection created");
        // create a queue session
        qSession = connection.createQueueSession(false,
                                         javax.jms.Session.AUTO_ACKNOWLEDGE);
        System.out.println("Queue session created");
        // create a queue
        testQueue = (javax.jms.Queue)ctx.lookup(_queue_name);
        System.out.println("Queue = [" + _queue_name + "]");
        // create a consumer listening to the specified queue
        // for message property APPLICATION = 'none' (general integration)
        qReceiver = (QueueReceiver)qSession.createConsumer(testQueue,
                                                   "APPLICATION = 'none'");
        // set the message listener
        qReceiver.setMessageListener(this);
        // set the exception listener
        connection.setExceptionListener(this);

        // initialize the SAX parser
        try {
```


© Copyright IBM Corporation 2009. All rights reserved.

```
            _parser = initParser(_dh);
        } catch (SAXException se) {
            System.out.println("SAXException - " + se.getMessage());
            throw se;
        }

        // connect to the Content Manager
        dsICM = new DKDatastoreICM();
        dsICM.connect(database, userid, pw, "");
        System.out.println("datastore connected.");

        // start the JMS connection
        connection.start();
        System.out.println("Queue connection started");
```

Retrieve the event data

After the JMS connection is started, the onMessage message listener retrieves the messages from the JMS queue based on a first-in/first-out manner. A JMS message is first cast to a TextMessage object. As illustrated in Listing 3, the program retrieves the message string from the TextMessage object. In addition, six message properties for the message are retrieved:

- DATABASE - The database name indicates the name of the CM8 server generating the event.
- TYPE - The event type indicates the type of the event (for example, TYPE = "item-create").
- LAUNCH - The launch indicator indicates whether the message requests to launch a workflow or not. Its value can be either LAUNCH = "true" or LAUNCH = "false". For general integration, the value of LAUNCH is "false".
- ID - The message ID is the identifier of the event, such as "AA1234567890123456789012345678901234567890123000".
- APPLICATION - The application type indicates the application associated with a workflow launch. For process integration with FileNet Business Process Manager (BPM), the value is APPLICATION = "BPM". For general integration, the value is APPLICATION = "none".
- COUNT - The data element count indicates how many data elements are contained in the event. If the length of the event is less than or equal to 1024, only one context data element exists in the event. Otherwise, if the count is greater than 1, the custom event handler needs to concatenate the multiple context data elements together.

Next, the program parses the message string, which is represented in a common base event format (for example, CBE format). The following information is retrieved from the CBE-formatted message string:

© Copyright IBM Corporation 2009. All rights reserved.

- Global instance ID - The `strGlobalInstanceId` variable indicates the global instance ID of the event, such as "AA1234567890123456789012345678901234567890123000", which is the same as the message ID.
- Event data - The `strReturnMsg` variable contains the event data from the CM8 content repository.
- Event creation time - The `strReturnTime` variable contains the creation time of the event in GMT, such as "2008-03-19T01:03:11.256Z".
- Application name - The `strApplication` variable indicates the name of the application generating the event. That is, "Content Manager Event Monitor".
- Component name - The `strComponent` variable indicates the component generating the event, such as "Content Manager 8.4.01.000".
- Execution environment - The `strExecutionEnvironment` variable shows the execution environment (OS and architecture), such as "Windows XP[x86]".
- Host location - The `strLocation` variable indicates the hostname and IP address of the event monitor, such as "myhost/1.11.22.33".
- SubComponent name - The `strSubComponent` variable shows the name of subcomponent. That is, "Event Monitor".

Listing 3. Retrieve the event data

```
// cast the message to the TextMessage object
msg = (TextMessage) message;
// retrieve the raw string from a JMS message
rawString = msg.getText();

System.out.println("Reading message: " + rawString);

// retrieve the message properties from a JMS message
msgDatabase = msg.getStringProperty("DATABASE");
msgType = msg.getStringProperty("TYPE");
msgLaunch = msg.getStringProperty("LAUNCH");
msgID = msg.getStringProperty("ID");
msgApplication = msg.getStringProperty("APPLICATION");
msgCount = msg.getStringProperty("COUNT");

// create an input source from the raw string
ByteArrayInputStream bais = new ByteArrayInputStream
                            (rawString.getBytes("UTF-8"));
InputSource is = new InputSource(bais);
bais.close();

// parse the string
try {
  _parser.parse(is);
} catch (SAXParseException se) {
  System.out.println("Encountered a SAX parser error"
                     + " - " + se.getMessage());
```


© Copyright IBM Corporation 2009. All rights reserved.

```
        se.printStackTrace();
        throw se;
      }

      // retrieve the information from a CBE-formatted string
      strGlobalInstanceId = ((myContentHandler)_dh).getGlobalInstanceId();
      strReturnMsg = ((myContentHandler)_dh).getReturnMsg();
      strReturnTime = ((myContentHandler)_dh).getReturnTime();
      strApplication = ((myContentHandler)_dh).getApplication();
      strComponent = ((myContentHandler)_dh).getComponent();
      strExecutionEnvironment = ((myContentHandler)_dh).getExecutionEnvironment();
      strLocation = ((myContentHandler)_dh).getLocation();
      strSubComponent = ((myContentHandler)_dh).getSubComponent();
```

Parse the event data

Listing 4 describes an implementation of simple parsing code to parse the event data retrieved from the message string The objective of this code is to get the PID string and ITEMTYPE name from the event data. The PID string and ITEMTYPE name are used later to retrieve the content data and compose an e-mail notification.

A `StringTokenizer` object is instantiated with the event data and the delimiter ";". A `while` loop iterates through the tokens until either the "ICMEMEND" is found or no more token is available. Each token is represented as a 3-tuple element with <tag>, =, and <value>. The token with the ITEMTYPE tag contains the name of the item type of the document referenced in the event data. Similarly, the token with the PID tag carries the persistent ID string of the document referenced in the event data.

Listing 4. Parse the event data

```
      // Use the StringTokenizer to parse the event message
      StringTokenizer tokenizer = new StringTokenizer(strReturnMsg, ";");

      // variable for PID
      String pid = "";
      // variable for itemtype
      String itemtype = "";

      String strPart = "";
      int pos = 0;
      String tagPart = "";

      // retrieve pid and itemtype from the event message
      while (true) {
        strPart = tokenizer.nextToken();
        if (strPart.equalsIgnoreCase("ICMEMEND")
            || strPart == null || strPart.equals("")) break;

        pos = strPart.indexOf('=');
        tagPart = "";

        if (pos > 0) {
            tagPart = strPart.substring(0, pos);
        }
        else continue;

        if (tagPart.equalsIgnoreCase("ITEMTYPE")) {
```


© Copyright IBM Corporation 2009. All rights reserved.

```
                    pos = strPart.indexOf('=');

                    if (pos > 0) {
                        itemtype = strPart.substring(pos + 1);
                    }
                }
                else {
                  if (tagPart.equalsIgnoreCase("PID")) {
                      pos = strPart.indexOf('=');

                      if (pos > 0) {
                          pid = strPart.substring(pos + 1);
                      }
                  }
                }
              } // end while
```

Retrieve the content data

Given the PID string retrieved from the event data, the document metadata can be retrieved from a content manager repository. Listing 5 illustrates how to retrieve a DDO object (in other words, dynamic data object) and its attributes from a CM8 server. First, a DDO object is created with the PID string from the event data. A retrieve call populates the DDO object with attribute values.

Next, the following attributes are retrieved:

- ClaimNo - Claim number as a unique identifier of the claim, like "308825".
- ClaimAmount - Amount of the claim, like "2000".
- PolicyID - Policy ID associated with the claim, like "8-112258".
- Notification - Child component, which contains a list of e-mail addresses with an `EMailAddress` attribute for the reviewers, like "user@host.com".

The attribute values from the DDO object are used to compose an e-mail notification later.

Listing 5. Retrieve the content data

```
              // retrieve the DDO object with the pid string
              DKDDO ddo  = (DKDDO)dsICM.createDDO(pid);
              ddo.retrieve();
              // retrieve the ClaimNo attribute
              short dataId = (short)ddo.dataId("ClaimNo");
              String claimid = (String)ddo.getData(dataId);
              // retrieve the ClaimAmount attribute
              dataId = (short)ddo.dataId("ClaimAmount");
              float claimamount = ((Double)(ddo.getData(dataId))).floatValue();
              // retrieve the PolicyID attribute
              dataId = (short)ddo.dataId("PolicyID");
              String policyid = (String)ddo.getData(dataId);
              // retrieve the Notification component
              DKChildCollection notification = (DKChildCollection) ddo.getData(
```


© Copyright IBM Corporation 2009. All rights reserved.

```
                (short)ddo.dataId(DKConstant.DK_CM_NAMESPACE_CHILD, "Notification"));
       dkIterator iterNotification = notification.createIterator();
       int count = notification.cardinality();
       // variable for Notification
       String[] notificationList = new String[count];
       int index = -1;
       // construct the array of e-mail addresses
       while (iterNotification.more()) {
         DKDDO ddoEmail = (DKDDO) iterNotification.next();
         ddoEmail.retrieve();
         dataId = (short) ddoEmail.dataId("EMailAddress");
         String email = (String) ddoEmail.getData(dataId);
         if (email == null || email.trim().equals("")) continue;
         else {
             index++;
             notificationList[index] = email.trim();
         }
       }

       // variable for the e-mail addresses
       String[] TargetEmailAddr = null;
       if (index > -1) {
           TargetEmailAddr = new String[index + 1];
           for (int k = 0; k <= index; k++) {
               TargetEmailAddr[k] = notificationList[k];
           }
       }
```

Compose an e-mail notification

For an illustrative purpose, IBM Web Interface for Content Management (WEBi) is selected as the client for these two tasks in the scenario:

- A customer service representative submits an auto claim form.
- A claim adjuster reviews the claim and updates the claim status after being notified.

A special URL is constructed to allow the claim adjuster direct access to the WEBi client from an e-mail to review and update the claim form.

Listing 6 describes the steps to compose an e-mail notification containing event information, document information, and the URL link to the WEBi client. First, the program initializes a `StringBuffer` object to store the URL link. The required information for constructing a WEBi URL are:

- Host name of a WEBi server (for example, www.myhost.com)
- Port number of a WEBi server (for example, 9082)
- Path of WEBi launch JSP (for example, wccommonservices/launch.jsp)
- Content server name (for example, wc_server=CMServer)
- Content server type (for example, wc_serverType=cm for Content

© Copyright IBM Corporation 2009. All rights reserved.

Manager)

- Template name (for example, wc_template_name=ClaimForm)
- Document ID, which can be constructed with a PID string, such as: docid=88%203%20ICM8%20ICMNLSDB9%20ClaimForm59%2026%20A1001001A09E

Next, a WEBi document ID is constructed by:

- Creating a DKPidICM object from the PID string
- Setting the version to 0 for the latest version in the PID string
- Replacing blanks in the updated PID string with "%20"

Then a valid WEBi URL is constructed by appending all the required elements to the `StringBuffer` object, as shown in the example.

The final step is to compose event information, document information, and the WEBi URL link. The following information is passed to the `sendMail` method, which sends an e-mail notification:

- STMP host name - Replace www.smtpserver.com in the `defaultSMTP` variable with a valid SMTP server.
- From address - Replace www.myhost.com in the `defaultSender` variable with a valid e-mail address as the sender address.
- To addresses - Include an array of recipient addresses.
- Subject - Provide a subject line.
- E-mail content - Include host name of database, event type, message ID, itemtype name, claim number, claim amount, policy ID, and a WEBi URL link to the claim form.

Listing 6. Compose an e-mail notification

```
// initialize s string buffer for the WEBi URL
StringBuffer webiURL = new StringBuffer();
// variable for the WEBi host
String webiHost = "www.myhost.com";
// variable for the port number
String webiPort = "9082";
// variable for the path of  WEBi launch jsp
String webiLaunchJSP = "wccommonservices/launch.jsp";
// variable for the name of the content server to be accessed by WEBi
String wcServer = "CMServer";
// variable for the server type
String wcServerType = "cm";
// variable for the template (itemtype)
String wcTemplate = itemtype;
// variable for the document id
String wcDocid = "";
```


© Copyright IBM Corporation 2009. All rights reserved.

```
        // construct a DKPidICM from the pid string
        DKPidICM  pidICM = new DKPidICM(pid);
        // set the version to 0 for the latest version
        pidICM.setVersionNumber("0");
        // replace blanks with %20 and assign the pid as the WEBi document ID
        wcDocid = pidICM.pidString().replaceAll(" ", "%20");

        // construct a valid WEBi URL to be embedded in the e-mail
        webiURL.append("http://").append(webiHost).append(":")
               .append(webiPort).append("/").append(webiLaunchJSP)
               .append("?").append("wc_server=").append(wcServer)
               .append("&").append("wc_serverType=").append(wcServerType)
               .append("&").append("wc_template_name=").append(wcTemplate)
               .append("&").append("docid=").append(wcDocid);

        // send an e-mail notification
        sendMail(defaultSMTP,
                defaultSender,
                TargetEmailAddr,
                "Event notification",
                "This is an event notification.\n\n" +
                "Host = " + strLocation + "\n" +
                "DATABASE = " + msgDatabase + "\n" +
                "EVENT TYPE = " + msgType + "\n" +
                "Message ID = " + msgID + "\n" +
                "Item Type  = " + itemtype + "\n" +
                "Claim No = " + claimid + "\n" +
                "Claim Amount = " + claimamount + "\n" +
                "Policy ID = " + policyid + "\n\n" +
                "Please review the claim at " + webiURL.toString() + "\n\n");
```

Send the e-mail notification

Listing 7 illustrates how to use JavaMail API to send an e-mail using the simple mail transport protocol (in other words, SMTP) and displays an excerpt of the `sendMail` method in the program. First, a SMTP server name is specified and a session object is instantiated. Next, the "From" address and an array of "To" addresses are set. The program also sets the subject of the e-mail, the mailer, and the sent date. Finally, the program calls the `Transport.send()` method to send the e-mail to the specified SMTP server to deliver the e-mail to the recipients.

Listing 7. Send the e-mail notification

```
    Properties props = System.getProperties();

    // use SMTP
    if (mailhost != null) props.put("mail.smtp.host", mailhost);

    // get a Session object
    javax.mail.Session session = javax.mail.Session.getInstance(props, null);
    if (debug) session.setDebug(true);

    // construct the message
    javax.mail.Message msg = new MimeMessage(session);
    // set From address
    if (from != null)
        msg.setFrom(new InternetAddress(from));
    else
```

 © Copyright IBM Corporation 2009. All rights reserved.

```
            msg.setFrom();

        InternetAddress[] toAddresses = new InternetAddress[to.length];
        for (int i = 0; i < to.length; i++) {
            toAddresses[i] = new InternetAddress(to[i]);
        }
        // set To address
        msg.setRecipients(javax.mail.Message.RecipientType.TO, toAddresses);

        // optional
        if (cc != null)
            msg.setRecipients(javax.mail.Message.RecipientType.CC,
                              InternetAddress.parse(cc, false));
        if (bcc != null)
            msg.setRecipients(javax.mail.Message.RecipientType.BCC,
                              InternetAddress.parse(bcc, false));
        // set Subject
        msg.setSubject(subject);
        // set e-mail text
        msg.setText(text);

        msg.setHeader("X-Mailer", mailer);
        msg.setSentDate(new Date());
        // send out the e-mail
        Transport.send(msg);
```

A custom event handler in action

Following the scenario, a customer service representative of a fictitious XYZ Insurance Company submits an auto claim form through IBM Web Interface for Content Management (WEBi) for his customer. As shown in Figure 7, he enters the claim number as 8-123456, the claim amount as $2000.00, and the policy ID as 30542285. A claim form TIFF file is also imported as a document part for this claim.

Figure 7. Import a claim form (part 1)

© Copyright IBM Corporation 2009. All rights reserved.

Add Document

File name: C:\testing\claim.tif Browse

Item Type ClaimForm

ClaimNo * 8-123456

ClaimAmount * 2000.00

ClaimStatus

PolicyID * 30542285

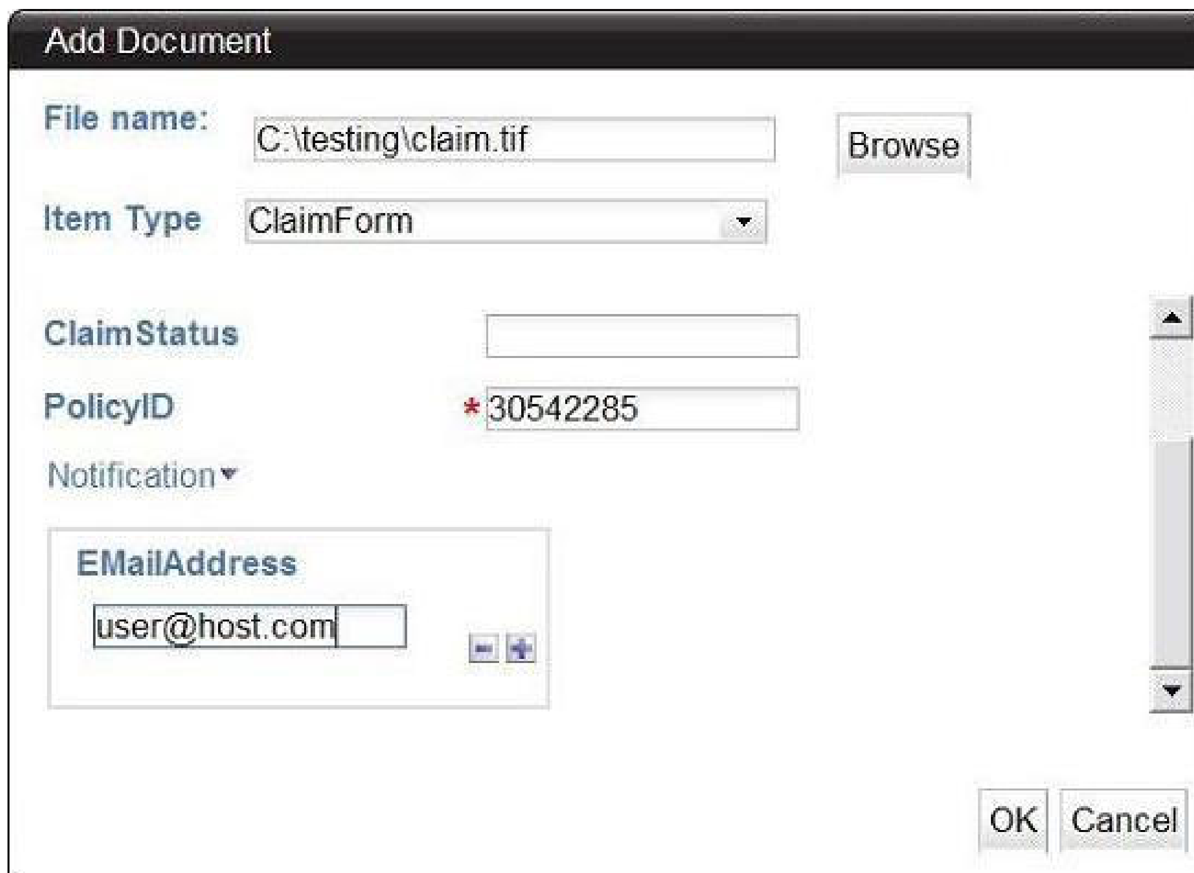
Notification

OK Cancel

As shown in Figure 8, he enters the e-mail address of the claim reviewer as user@host.com:

Figure 8. Import a claim form (part 2)

© Copyright IBM Corporation 2009. All rights reserved.

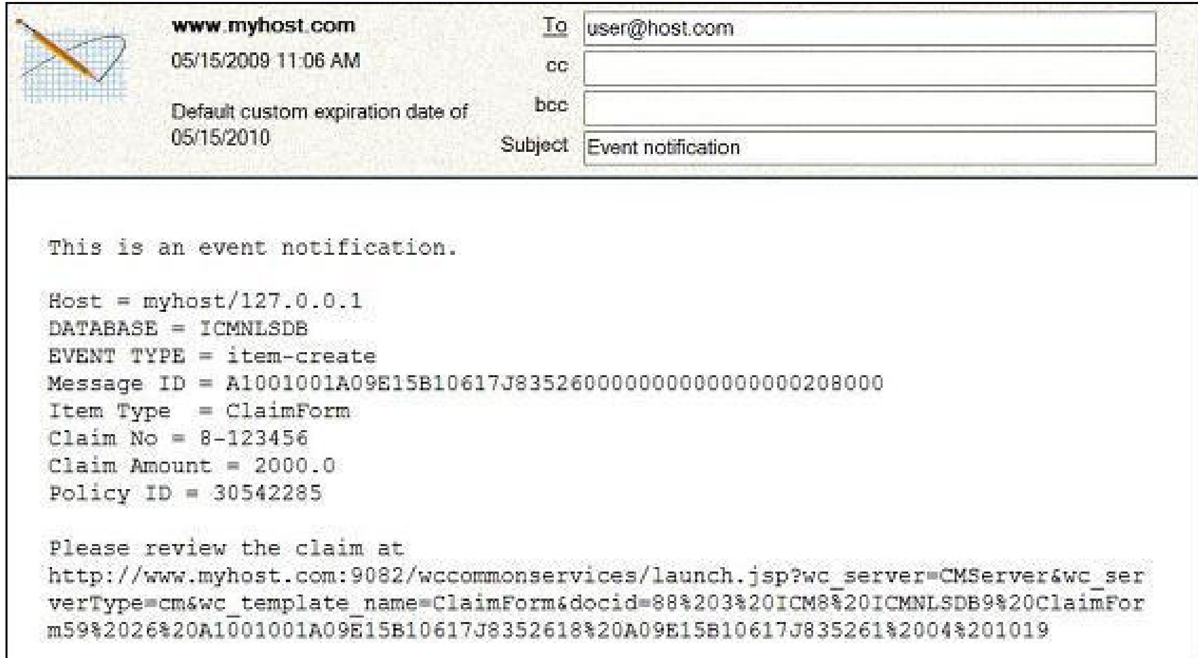


The event monitor and the custom event handler were started. Based on the event subscription for general integration in the scenario, the event monitor fetches the event data of the "Add Item" event, which was logged by the library server. A JMS message with the event data is generated and sent to the JMS queue. The custom event handler reads the JMS message from the JMS queue. The event data from the JMS message is parsed for composing the e-mail notification to the claim adjusters for review.

Figure 9 illustrates an example of an event notification, which is an e-mail received from the custom event handler. The event notification includes the following information: host name, database name, event type, message ID, item type, claim number, claim amount, policy ID, and the URL to the content to be reviewed.

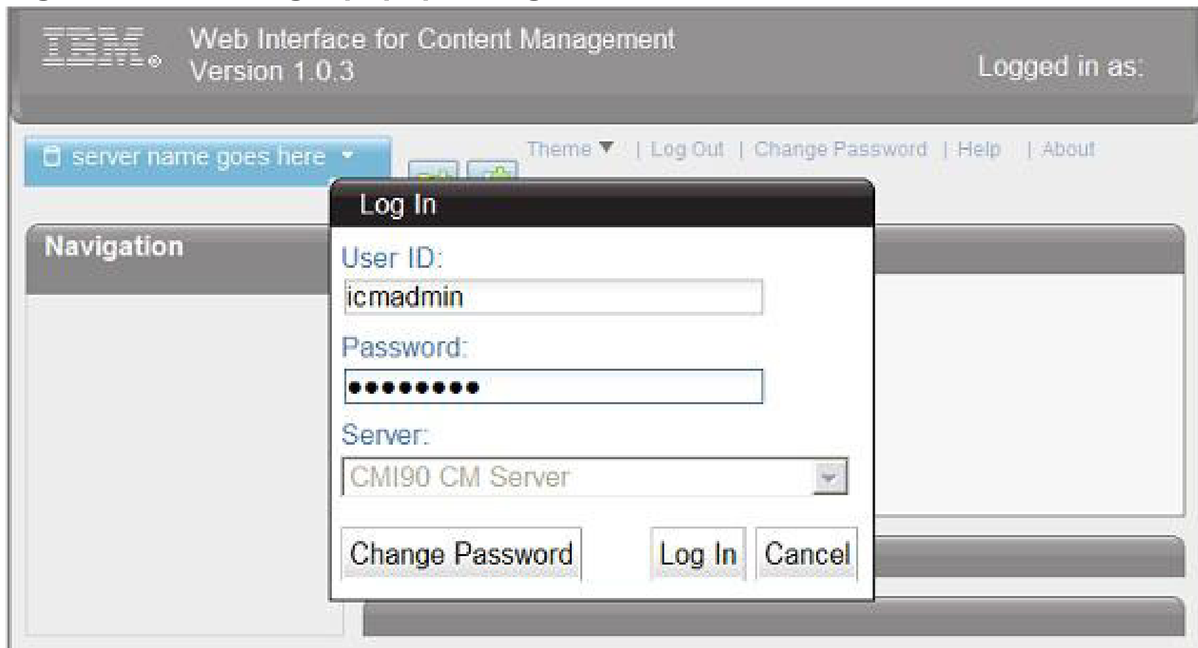
Figure 9. E-mail notification with a URL

© Copyright IBM Corporation 2009. All rights reserved.



When a claim adjuster receives the e-mail notification with the claim information, he can click on the URL link in the e-mail to launch WEBi in order to access the claim form. Figure 10 shows a WEBi login popup dialog, which requires user ID and password for authentication.

Figure 10. WEBi login popup dialog



After the claim adjuster is authenticated, the claim form item is displayed on the screen. As shown in Figure 11, the claim has claim number as 8-123456 and claim amount as 2000.0.

© Copyright IBM Corporation 2009. All rights reserved.

Figure 11. Retrieve the claim form from a URL

Next, the claim adjuster selects the check box and clicks on the view icon (the left-most icon on the tool bar) to view the claim form document. A viewer applet is started momentarily. Figure 12 illustrates a claim form, which is a TIFF image to be reviewed by the claim adjuster.

Figure 12. View the claim form

© Copyright IBM Corporation 2009. All rights reserved.

When the claim adjuster has reviewed the claim form document, he can click on the edit properties icon (the second icon from the left on the tool bar) to update the claim status with his decision. Figure 13 shows an edit properties dialog for the claim adjuster. Any change of the claim status is to be updated in the library server database.

Figure 13. Update the claim form

© Copyright IBM Corporation 2009. All rights reserved.

Edit Properties

ClaimForm

ClaimNo * 8-123456

ClaimAmount * 2000.0

ClaimStatus

PolicyID * 30542285

Notification

EMailAddress

user@host.com

OK Cancel

Conclusion

IBM Content Manager, Version 8.4.1 supports an event infrastructure that enables the integration of external applications. An external application can be integrated by using a custom event handler. This article has described a scenario and the design of a custom event handler. An example was provided to illustrate the interaction between IBM Content Manager and IBM Web Interface for Content Management through a custom event handler. With the capability of customizing event handlers, solution providers can develop versatile business applications in the enterprise content management area.

© Copyright IBM Corporation 2009. All rights reserved.

Downloads

Description	Name	Size	Download method
Sample program for this article	myEventHandler.zip	8KB	HTTP

Information about download methods

© Copyright IBM Corporation 2009. All rights reserved.

Resources

Learn

- "An event-driven framework for integrating IBM Content Manager with IBM FileNet Business Process Manager" (developerWorks, Jun 2009): Follow an auto claims scenario to see how to start a FileNet BPM process from Content Manager.
- IBM Content Manager: Learn more about IBM Content Manager products at ibm.com.
- ECM zone on developerWorks: Find developer resources, tutorials, and articles for Enterprise Content Management software.
- developerWorks Information Management zone: Learn more about Information Management. Find technical documentation, how-to articles, education, downloads, product information, and more.
- Stay current with developerWorks technical events and webcasts.

Get products and technologies

- Build your next development project with IBM trial software, available for download directly from developerWorks, or explore the online trials in the IBM SOA Sandbox and get your hands on application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.

Discuss

- Participate in developerWorks blogs and get involved in the developerWorks community.

About the author

Alan Yaung
Alan Yaung has extensive experiences in workflow and content management related products. He is the lead developer of the event monitor and the event handler for the integration of IBM DB2 Content Manager Version 8 and FileNet Business Process Manager.

© Copyright IBM Corporation 2009. All rights reserved.