Publishing IMS and DB2 Data using WebSphere Information Integrator: Configuration and Monitoring Guide

- WebSphere Information Integrator Classic Event Publisher architecture
- WebSphere Information Integrator Replication for z/OS
- Insurance industry business scenario

Nagraj Alur
Neale Armstrong
Tadakatsu Azuma
Christian Daser
Manuel Mueller

ibm.com/redbooks
First Edition (December 2005)

This edition applies to Version 8, Release 2 of WebSphere Information Integrator Classic Event Publisher (5665-M38) and Version 8, Release 2 of WebSphere Information Integrator Replication Edition (product number 5724-C74).

© Copyright International Business Machines Corporation 2005. All rights reserved.
Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
# Contents

Figures ................................................................. vii
Tables ................................................................. xi
Examples ............................................................... xiii
Notices ................................................................. xvii
Trademarks ........................................................... xviii
Preface ......................................................................... xix
The team that wrote this redbook ................................... xix
Become a published author ........................................... xxi
Comments welcome .................................................. xxi

Chapter 1. WebSphere Information Integrator Classic Event Publisher for IMS architecture ........................................ 1
  1.1 Introduction ........................................................ 2
  1.2 WIICEP for IMS architecture ..................................... 4
  1.3 WIICEP for IMS configuration .................................... 10
    1.3.1 STEP 1: Choose topology for the WIICEP for IMS environment ................................................ 11
    1.3.2 STEP 2: Install WIICEP for IMS ................................ 11
    1.3.3 STEP 3: Configure the WIICEP for IMS environment ............................................................. 11
  1.4 WIICEP for IMS processing flow .................................. 34
    1.4.1 Overview of WIICEP for IMS processing flow ............... 35
    1.4.2 Detailed description of WIICEP for IMS processing flow ....... 36
  1.5 WIICEP for IMS monitoring tools ................................ 42
    1.5.1 Monitor the correlation service .................................. 42
    1.5.2 Monitor the publication service ................................. 43
    1.5.3 Monitor and manage the CSA ................................. 43
  1.6 WIICEP for IMS recovery mode .................................. 44
    1.6.1 Recovery mode overview ....................................... 45
    1.6.2 Causes of recovery mode ....................................... 51
    1.6.3 Getting out of recovery mode ................................... 54
    1.6.4 Conditions when recovery is not possible ................. 68
    1.6.5 Methods to determine if a CCA has updated a monitored database .......... 68
    1.6.6 Best practices to minimize the occurrence of recovery mode .................. 69
    1.6.7 Detection of recovery mode ................................... 69

Chapter 2. WebSphere Information Integrator Event Publisher architecture ............................. 73
  2.1 Introduction ........................................................ 74
  2.2 WIEP architecture .................................................. 75
  2.3 WIEP processing flow .............................................. 80
    2.3.1 Q Capture ...................................................... 81
    2.3.2 WebSphere MQ ............................................... 82
  2.4 WIEP monitoring tools ............................................ 84
    2.4.1 Replication Center ............................................ 84
    2.4.2 Replication Alert Monitor ................................... 84

Chapter 3. Trustworthy Insurance Customer Information Facility (CIF) and Event Alert System scenario ............ 85
3.1 Business requirement ............................................................... 86
  3.1.1 Customer Information Facility (CIF) requirements .................. 86
  3.1.2 Event Alert System requirements .................................... 87
3.2 Environment configuration .................................................... 87
3.3 Step-by-step set up ............................................................... 90
  3.3.1 STEP 1: Design topology of the CIF and Event Alert System solution .................. 91
  3.3.2 STEP 2: Install prerequisite software .................................. 96
  3.3.3 STEP 3: Unload auto insurance and home insurance data ........... 96
  3.3.4 STEP 4: Stop WIICEP for IMS and WIIEP for DB2 UDB for z/OS .... 96
  3.3.5 STEP 5: Configure WebSphere MQ on the z/OS and Windows platforms .... 97
  3.3.6 STEP 6A: Configure WIICEP for IMS .................................. 107
  3.3.7 STEP 6B: Configure WIIEP for DB2 UDB for z/OS .................. 148
  3.3.8 STEP 7: Define and configure CIF ODS and install programs ........ 156
  3.3.9 STEP 8: Start WIICEP for IMS, WIIEP for DB2 UDB for z/OS and IMS .... 193

Chapter 4. Recovery mode scenarios ............................................ 195
  4.1 Introduction ........................................................................ 196
  4.2 Simple recovery mode scenario ........................................... 196
  4.3 Recovery mode resolution recommended approach ...................... 209
  4.4 Complex recovery mode scenario ......................................... 215

Appendix A. XML message structure and handling .......................... 231
  A.1 Types of XML messages for event publishing .......................... 232
  A.2 Processing XML event messages ......................................... 234
    A.2.1 WebSphere Business Integration Message Broker ............... 235
    A.2.2 WebSphere DataStage .................................................. 236
    A.2.3 Java with the XML Publishing Toolkit ........................... 236
    A.2.4 DB2 Message Queue Listener ....................................... 236
  A.3 RHFUTIL tool ............................................................... 237
  A.4 XML Publishing Toolkit ................................................... 238

Appendix B. REXX based tool to identify log files for the recovery change capture agent
              (CCA) ........................................................................ 241
  B.1 Problem domain ................................................................ 242
  B.2 REXX based tool overview .................................................. 242
    B.2.1 Inputs and outputs ....................................................... 244
    B.2.2 Functions .................................................................. 244
    B.2.3 Configuration file ....................................................... 244
  B.3 REXX based tool usage ........................................................ 245
    B.3.1 Invocation .................................................................. 246
    B.3.2 Sample output explanation ......................................... 253

Appendix C. Additional material ................................................... 283
  Locating the Web material ....................................................... 283
  Using the Web material ......................................................... 283
    System requirements for downloading the Web material ............ 284
    How to use the Web material .............................................. 284

Related publications ................................................................. 285
  IBM Redbooks .................................................................... 285
  Other publications .............................................................. 285
  Online resources .................................................................. 286
  How to get IBM Redbooks .................................................... 286
  Help from IBM ................................................................. 286
Index .................................................. 287
# Figures

1-1 WIICEP architecture ................................................................. 5
1-2 Metadata catalog creation using Data Mapper .................................. 9
1-3 Main steps in configuring your WIICEP for IMS environment ............. 10
1-4 WIICEP for IMS configuration steps ............................................. 11
1-5 ftp request to download the CAC.V8R2M0.SACCLNT(CAC82WN) member .... 13
1-6 Tools summary .......................................................................... 13
1-7 Sample IMS D21PART database .................................................. 15
1-8 WebSphere MQ Explorer ............................................................ 34
1-9 Processing flow overview .......................................................... 36
1-10 WIICEP for IMS components in detail ......................................... 36
1-11 Recovery CCA overview ........................................................... 47
1-12 Recovery CCA restart points ..................................................... 50
1-13 Recovery mode process overview ................................................. 55
1-14 Related IMS CCAs scenario ...................................................... 56
2-1 Event publication process .......................................................... 74
2-2 Simple configuration in event publishing ....................................... 76
2-3 Event publishing flow ............................................................... 80
3-1 Auto insurance (hierarchic IMS) and Home (relational) DB2 structures . 86
3-2 Trustworthy Insurance’s CIF and event alert system configuration ....... 88
3-3 Trustworthy Insurance’s CIF and event alert system technical components. 90
3-4 Overview of set up steps ........................................................... 91
3-5 Trustworthy Insurance’s CIF & Event Alert System topology ............. 92
3-6 Detailed view of WebSphere MQ objects used in the Trustworthy Insurance solution. 97
3-7 WebSphere MQ queues created ................................................... 104
3-8 WebSphere MQ channels ............................................................ 104
3-9 WebSphere MQ channels status .................................................. 105
3-10 WIICEP for IMS configuration steps for the CIF and Event Alert System solution. 108
3-11 Create a new repository ........................................................... 110
3-12 DataMapper auto.db repository ................................................ 110
3-13 Create a Data Catalog 1/2 ........................................................... 111
3-14 Create a Data Catalog 2/2 .......................................................... 111
3-15 AUTODB data catalog ............................................................. 111
3-16 Load DBD file 1/5 ................................................................. 112
3-17 Load DBD file 2/5 ................................................................. 112
3-18 Load DBD file 3/5 ................................................................. 113
3-19 Load DBD file 4/5 ................................................................. 113
3-20 Load DBD file 5/5 ................................................................. 114
3-21 Relationship between IMS physical segments and “logical” tables for CIF solution 115
3-22 Relationship between IMS physical segments and “logical” tables for EAS solution 115
3-23 List Tables 1/2 ................................................................. 116
3-24 List Tables 2/2 ................................................................. 117
3-25 List Owners 1/2 ................................................................. 117
3-26 List Owners 2/2 ................................................................. 118
3-27 Create a New Owner 1/3 .......................................................... 118
3-28 Create a New Owner 2/3 .......................................................... 119
3-29 Create a New Owner 3/3 .......................................................... 119
3-30 Create “logical” table CUSTOMER 1/9 ......................................... 119
3-31 Create “logical” table CUSTOMER 2/9 ......................................... 120
3-32 Create "logical" table CUSTOMER 3/9 .................................................. 120
3-33 Create "logical" table CUSTOMER 4/9 .................................................. 121
3-34 Create "logical" table CUSTOMER 5/9 .................................................. 121
3-35 Create "logical" table CUSTOMER 6/9 .................................................. 122
3-36 Create "logical" table CUSTOMER 7/9 .................................................. 122
3-37 Create "logical" table CUSTOMER 8/9 .................................................. 123
3-38 Create "logical" table CUSTOMER 9/9 .................................................. 123
3-39 Create "logical" table ADDRESS 1/6 .................................................... 124
3-40 Create "logical" table ADDRESS 2/6 .................................................... 124
3-41 Create "logical" table ADDRESS 3/6 .................................................... 125
3-42 Create "logical" table ADDRESS 4/6 .................................................... 125
3-43 Create "logical" table ADDRESS 5/6 .................................................... 126
3-44 Create "logical" table ADDRESS 6/6 .................................................... 126
3-45 Create "logical" table POLICY 1/5 ....................................................... 127
3-46 Create "logical" table POLICY 2/5 ....................................................... 127
3-47 Create "logical" table POLICY 3/5 ....................................................... 128
3-48 Create "logical" table POLICY 4/5 ....................................................... 128
3-49 Create "logical" table POLICY 5/5 ....................................................... 129
3-50 Create "logical" table CLAIMS 1/5 ...................................................... 129
3-51 Create "logical" table CLAIMS 2/5 ...................................................... 130
3-52 Create "logical" table CLAIMS 3/5 ...................................................... 130
3-53 Create "logical" table CLAIMS 4/5 ...................................................... 131
3-54 Create "logical" table CLAIMS 5/5 ...................................................... 131
3-55 Create "logical" table PREMIUMS 1/5 .................................................. 132
3-56 Create "logical" table PREMIUMS 2/5 .................................................. 132
3-57 Create "logical" table PREMIUMS 3/5 .................................................. 133
3-58 Create "logical" table PREMIUMS 4/5 .................................................. 133
3-59 Create "logical" table PREMIUMS 5/5 .................................................. 134
3-60 Create "logical" table QUIKCLM 1/4 .................................................. 134
3-61 Create "logical" table QUIKCLM 2/4 .................................................. 135
3-62 Create "logical" table QUIKCLM 3/4 .................................................. 135
3-63 Create "logical" table QUIKCLM 4/4 .................................................. 136
3-64 Generate USE Grammar for "logical" tables defined on AUTODB .......... 136
3-65 WIIP for DB2 UDB configuration steps for CIF and Event Alert System solution . 148
3-66 Windows DB2 Connect Personal Edition access to DB2 for z/OS .......... 149
3-67 DB2 Administration Client with DB2 for z/OS access via DB2 Connect server . 149
3-68 Create Q Capture control tables 1/4 .................................................. 150
3-69 Create Q Capture control tables 2/4 .................................................. 150
3-70 Create Q Capture control tables 3/4 .................................................. 151
3-71 Create Q Capture control tables 4/4 .................................................. 151
3-72 Create the Publication Queue Maps 1/3 .............................................. 152
3-73 Create the Publication Queue Maps 2/3 .............................................. 152
3-74 Create the Publication Queue Maps 3/3 .............................................. 153
3-75 Create publications 1/7 ................................................................. 153
3-76 Create publications 2/7 ................................................................. 153
3-77 Create publications 3/7 ................................................................. 154
3-78 Create publications 4/7 ................................................................. 154
3-79 Create publications 5/7 ................................................................. 155
3-80 Create publications 6/7 ................................................................. 155
3-81 Create publications 7/7 ................................................................. 156
3-82 CIF ODS environment set up steps ................................................... 156
3-83 CIF ODS database ....................................................................... 157
3-84 Sample CUSTLINK mapping table .................................................. 158
<table>
<thead>
<tr>
<th>Page</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-85</td>
<td>CIF Updater program structure</td>
<td>160</td>
</tr>
<tr>
<td>3-86</td>
<td>CIF Updater program functional details 1/3</td>
<td>160</td>
</tr>
<tr>
<td>3-87</td>
<td>CIF Updater program functional details 2/3</td>
<td>161</td>
</tr>
<tr>
<td>3-88</td>
<td>CIF Updater program functional details 3/3</td>
<td>161</td>
</tr>
<tr>
<td>3-89</td>
<td>Event notification program functional details 1/3</td>
<td>187</td>
</tr>
<tr>
<td>4-1</td>
<td>Simple recovery scenario</td>
<td>197</td>
</tr>
<tr>
<td>4-2</td>
<td>Recovery data set metadata</td>
<td>202</td>
</tr>
<tr>
<td>4-3</td>
<td>Recovery mode resolution recommended approach flowchart</td>
<td>210</td>
</tr>
<tr>
<td>4-4</td>
<td>Potentially different timelines of related CCAs updating SAVINGDB IMS database</td>
<td>211</td>
</tr>
<tr>
<td>4-5</td>
<td>Complex recovery mode scenario</td>
<td>216</td>
</tr>
<tr>
<td>A-1</td>
<td>Identify queue of interest</td>
<td>237</td>
</tr>
<tr>
<td>A-2</td>
<td>XML message content</td>
<td>238</td>
</tr>
<tr>
<td>B-1</td>
<td>REXX tool flowchart</td>
<td>243</td>
</tr>
<tr>
<td>B-2</td>
<td>Sample scenario to explain REXX based tool usage</td>
<td>248</td>
</tr>
</tbody>
</table>
## Tables

1-1 Supported IMS environments and database types ............................................. 3  
1-2 Parameters of the EXIT keyword ................................................................. 16  
1-3 Correlation service SIE parameters .............................................................. 24  
1-4 Publication service SIE parameters ............................................................... 26  
1-5 WIICEP configuration parameters ................................................................. 28  
1-6 IMS log records used by WIICEP for IMS ...................................................... 39  
1-7 Execution modes ............................................................................................. 46  
1-8 Recovery CCA control card format ................................................................. 47  
1-9 WTO messages of interest relating to recovery mode ....................................... 70  
2-1 Comparing the main features of event publishing and Q replication .............. 79  
3-1 System information ......................................................................................... 92  
3-2 IMS DB/DC information .................................................................................. 93  
3-3 WebSphere Information Integrator Classic Event Publisher for IMS information ........................................................................ 93  
3-4 DB2 UDB for z/OS information ....................................................................... 94  
3-5 WebSphere Information Integrator Event Publisher - DB2 UDB for z/OS information ................................................................. 94  
3-6 WebSphere MQ information ............................................................................ 94  
3-7 DB2 UDB CIF ODS information ..................................................................... 95  
3-8 CIF updater programs information ................................................................ 95  
4-1 IMS CCAs in the complex recovery mode scenario ......................................... 215  
A-1 XML Publishing Toolkit most useful classes ............................................... 239  
B-1 Main functions in the REXX-based tool ......................................................... 244
## Examples

<table>
<thead>
<tr>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>DI21PART DBD definition with EXIT= keyword in SEGM definitions</td>
</tr>
<tr>
<td>1-2</td>
<td>Recovery data set allocation</td>
</tr>
<tr>
<td>1-3</td>
<td>Allocating the metadata catalog sequential data set</td>
</tr>
<tr>
<td>1-4</td>
<td>Allocate correlation service restart GDG dataset</td>
</tr>
<tr>
<td>1-5</td>
<td>SIEs of the correlation service and publication service &amp; a publication definition</td>
</tr>
<tr>
<td>1-6</td>
<td>Modified CACCS member</td>
</tr>
<tr>
<td>1-7</td>
<td>JCL to start WIICEP for IMS</td>
</tr>
<tr>
<td>1-8</td>
<td>Job output of WIICEP for IMS</td>
</tr>
<tr>
<td>1-9</td>
<td>Correlation service status</td>
</tr>
<tr>
<td>1-10</td>
<td>Publication service status</td>
</tr>
<tr>
<td>1-11</td>
<td>Correlation service report</td>
</tr>
<tr>
<td>1-12</td>
<td>Publication service report</td>
</tr>
<tr>
<td>1-13</td>
<td>CSA reporting and maintenance utility report</td>
</tr>
<tr>
<td>1-14</td>
<td>Release CSA job</td>
</tr>
<tr>
<td>1-15</td>
<td>Sample recovery CCA control card</td>
</tr>
<tr>
<td>1-16</td>
<td>Sample recovery CCA job with instream control cards with check agent</td>
</tr>
<tr>
<td>1-17</td>
<td>Recovery CCA check agent output</td>
</tr>
<tr>
<td>1-18</td>
<td>Sample recovery CCA control cards for log file recovery mode 'L'</td>
</tr>
<tr>
<td>1-19</td>
<td>Sample recovery CCA output for log file recovery mode</td>
</tr>
<tr>
<td>1-20</td>
<td>IMS Status control file</td>
</tr>
<tr>
<td>1-21</td>
<td>Set agent in recovery mode control cards</td>
</tr>
<tr>
<td>1-22</td>
<td>Set agent in recovery mode partial output</td>
</tr>
<tr>
<td>1-23</td>
<td>Sample DBRC LIST.LOG ALL command output</td>
</tr>
<tr>
<td>1-24</td>
<td>Sample log file recovery control cards</td>
</tr>
<tr>
<td>1-25</td>
<td>Sample log file recovery output</td>
</tr>
<tr>
<td>1-26</td>
<td>Sample set agent in active mode control cards</td>
</tr>
<tr>
<td>1-27</td>
<td>Sample set agent in active mode output</td>
</tr>
<tr>
<td>1-28</td>
<td>Job to create local queues, channel and transmit queue</td>
</tr>
<tr>
<td>1-29</td>
<td>Job to create send queues</td>
</tr>
<tr>
<td>1-30</td>
<td>Script file for creating WebSphere MQ objects</td>
</tr>
<tr>
<td>1-31</td>
<td>QMCIF.txt script output</td>
</tr>
<tr>
<td>1-32</td>
<td>Invoking amqsput on z/OS</td>
</tr>
<tr>
<td>1-33</td>
<td>amqsget script</td>
</tr>
<tr>
<td>1-34</td>
<td>Sample amqsget output</td>
</tr>
<tr>
<td>1-35</td>
<td>AUTODB DBD</td>
</tr>
<tr>
<td>1-36</td>
<td>AUTODB IMS SEGMENT COBOL copybooks</td>
</tr>
<tr>
<td>1-37</td>
<td>AUTOPSB1</td>
</tr>
<tr>
<td>1-38</td>
<td>AUTOPSB1</td>
</tr>
<tr>
<td>1-39</td>
<td>Generate USE Grammar</td>
</tr>
<tr>
<td>1-40</td>
<td>Modified AUTODB to support event publishing</td>
</tr>
<tr>
<td>1-41</td>
<td>AUTODB DBDGEN job</td>
</tr>
<tr>
<td>1-42</td>
<td>AUTODB PSBGEN job</td>
</tr>
<tr>
<td>1-43</td>
<td>AUTODB ACBGEN job</td>
</tr>
<tr>
<td>1-44</td>
<td>Copy staging libraries to active libraries</td>
</tr>
<tr>
<td>1-45</td>
<td>Recovery data set allocation information</td>
</tr>
<tr>
<td>1-46</td>
<td>DD card to add to SY1.PROCLIB(IMS810K) IMS started task JCL</td>
</tr>
<tr>
<td>1-47</td>
<td>Allocate metadata catalog for the CIF &amp; Event Alert System solution</td>
</tr>
<tr>
<td>1-48</td>
<td>Populate metadata catalog for the CIF &amp; Event Alert System solution</td>
</tr>
<tr>
<td>1-49</td>
<td>Allocate GDG data sets for the correlation service</td>
</tr>
</tbody>
</table>
4-39 Browse NALUR.LOG3 to determine system clock time.  
4-40 CAC.RECOVDS.CACBACH3 recovery data set restart information entry  
4-41 Recovery CCA ‘S’ mode (set agent in recovery mode) control cards  
4-42 Recovery CCA ‘S’ mode (set agent in recovery mode) output  
4-43 Recovery CCA ‘L’ mode (log file recovery) control cards  
4-44 Recovery CCA ‘L’ mode (log file recovery) output.  
4-45 Recovery CCA ‘A’ mode (place an agent in active mode) control cards  
4-46 Recovery CCA ‘A’ mode (place an agent in active mode) output  
A-1 TRANS message type XML message for event publishing.  
A-2 ROWOP message type XML message for event publishing  
B-1 Sample configuration file  
B-2 RUNZJCL job.  
B-3 runlinux.sh script  
B-4 DBRC LIST.LOG ALL output (partial).  
B-5 Sample graphic output of the REXX based tool.  
B-6 IMSK SSID graphic timeline.  
B-7 REXX based tool output log.  
B-8 REXX tool source code
Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

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

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

Ascential DataStage®  DB2®  Redbooks™
Ascential®  DRDA®  Redbooks (logo) ™
CICS®  @server®  WebSphere®
DataStage®  @server®  z/OS®
DB2 Connect™  IBM®  zSeries®
DB2 Universal Database™  IMS™

The following terms are trademarks of other companies:

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

Microsoft, PowerPoint, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

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

Other company, product, or service names may be trademarks or service marks of others.
Preface

This IBM® Redbook provides an overview of WebSphere® Information Integrator Classic Event Publisher for IMS™, and documents the procedures for implementing WebSphere Information Integrator Classic Event Publisher and WebSphere Information Integrator Q replication event publishing in a z/OS® environment.

It is aimed at an audience of IT architects and database administrators (DBA) responsible for managing event publishing using WebSphere Information Integrator Classic Event Publisher for IMS and WebSphere Information Integrator Q replication in the z/OS environment.

This redbook documents a step-by-step approach to implementing an event publishing scenario involving replication and event handling using the WebSphere Information Integrator family of products.

This book is organized as follows:

- Chapter 1, “WebSphere Information Integrator Classic Event Publisher for IMS architecture” on page 1 provides a detailed description of WebSphere Information Integrator Classic Event Publisher for IMS, its architecture and processing flow. Best practices recommendations for the area of key configuration options are also discussed.

- Chapter 2, “WebSphere Information Integrator Event Publisher architecture” on page 73 provides an overview of WebSphere Information Integrator Event Publishing, its architecture, and processing flow.

- Chapter 3, “Trustworthy Insurance Customer Information Facility (CIF) and Event Alert System scenario” on page 85 describes a step by step approach to implementing a customer information facility (CIF) and a risk-exposure event handling application for a fictitious insurance company using the event publishing capabilities of the WebSphere Information Integrator family of products.

- Chapter 4, “Recovery mode scenarios” on page 195 describes our recommended approach for managing the occurrence of recovery mode in the WebSphere Information Integrator Classic Event Publisher for IMS environment, using a typical scenario involving multiple IMS active Change Capture Agents (CCAs). Two example scenarios are provided.

- Appendix A, “XML message structure and handling” on page 231 provides a brief overview of the XML Publication Toolkit.

- Appendix B, “REXX based tool to identify log files for the recovery change capture agent (CCA)” on page 241 provides a description of the sample code developed by the ITSO to identify the IMS logs required by the recovery CCA during the recovery process.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, San Jose Center.

Nagraj Alur is a Project Leader with the IBM International Technical Support Organization, San Jose Center. He holds a Masters Degree in Computer Science from the Indian Institute of Technology (IIT), Mumbai, India. He has more than 30 years of experience in DBMSs, and has been a programmer, systems analyst, project leader, independent consultant, and researcher. His areas of expertise include DBMSs, data warehousing, distributed systems
management, database performance, information integration, as well as client/server and Internet computing. He has written extensively on these subjects and has taught classes and presented at conferences all around the world. Before joining the ITSO in November 2001, he was on a two-year assignment from the Software Group to the IBM Almaden Research Center, where he worked on Data Links solutions and an eSourcing prototype.

**Neale Armstrong** is a DB2® consultant working for IBM UK. Neale has specialized in WebSphere Information Integration disciplines of Federation and SQL Replication for several years. More recently he has been supporting some of the early adopters of Q Replication in the UK.

**Tadakatsu Azuma** is a leading DPROPR specialist with IBM Japan. He has over 12 years of experience in the field of data replication, having implemented several major customer replication solutions. He has particular expertise in using DPROPR for replication to and from non-IBM databases. He was a major contributor to the previous IBM Redbooks™, including *Data Where you Need It, the DPropR Way!*, GG24-4492-00; and DPROPR Planning and Design Guide, SG24-4771-00. He also contributed to the Prentice Hall publication, *DB2 V5 Data Replication Certification Guide*, ISBN: 0130824240.

**Christian Daser** is an IT Specialist with IBM Software Group in Germany. He holds a Bachelor of Science with Honours, specializing in Information Technology from University of Cooperative Education in Stuttgart (Germany) and Open University, London. He has two years of experience in DBMSs working as a technical sales person for information management products. His areas of expertise includes WebSphere Application Server accessing DB2 using JDBC™ and SQLJ. Last year, he specialized in DB2 z/OS and the Information Integration portfolio on zSeries®, where he developed and taught workshops on WebSphere Information Integrator Classic Federation in Germany.

**Manuel Mueller** is an IMS specialist in technical presales in Ehningen, Germany. He studied Information Technology Management at the University of Cooperative Education (Berufsakademie) Stuttgart, Germany in a partnership program with IBM. His diploma thesis dealt with the comparison of hierarchical and relational databases. After graduating as a Diplom Ingenieur (BA) he joined IBM in 2003.

Thanks to the following people for their contributions to this project:

Paul Cadarette
Tom Jacopi
Bob Love
Gregg Upton
Anil Varkhedi
Shan (Sandra) Yu
**IBM Silicon Valley Laboratory**

Bart Steegmans
**IBM Belgium**

Henry Kiesslich
**IBM Germany**

Jouko Jantti
**IBM Finland**

Toshiaki Sota
**IBM Japan**
Steve Spirk  
Ingram Micro

Yvonne Lyon  
Paolo Bruni  
Emma Jacobs  
International Technical Support Organization, San Jose Center

Richard Conway  
International Technical Support Organization, Poughkeepsie Center

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:  
ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our Redbooks to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

▪ Use the online Contact us review redbook form found at:
  ibm.com/redbooks

▪ Send your comments in an e-mail to:
  redbook@us.ibm.com

▪ Mail your comments to:
  IBM Corporation, International Technical Support Organization  
  Dept. QXXE  Building 80-E2  
  650 Harry Road  
  San Jose, California 95120-6099
In this chapter we provide a detailed description of WebSphere Information Integrator Classic Event Publisher for IMS, its architecture and processing flow, and considerations in choosing between a distributed or consolidated server configuration for a given business requirement. Best practices recommendations for key configuration options area also discussed.

The topics covered are:

- WebSphere Information Integrator Classic Event Publisher (WIICEP) architecture
- WIICEP for IMS configuration
- WIICEP for IMS processing flow
- WIICEP for IMS monitoring tools
- WIICEP for IMS recovery mode
1.1 Introduction

The IBM WebSphere Information Integrator V8.2 portfolio provides the foundation for a strategic information integration framework that helps customers decrease time to market for new applications, get more value and insight from existing assets, and control IT costs. This platform provides a range of information integration capabilities — enterprise search, data federation, data transformation, data placement (caching and replication), and data event publishing — designed to meet a diverse range of data integration requirements for business intelligence and business integration.

Of particular interest in this redbook are the WebSphere Information Integrator Event Publisher (WIIEP) offerings on the z/OS platform, that capture database changes in DB2 UDB for z/OS, IMS, CA-IDMS™, Software AG Adabas™, or VSAM by reading the active or recovery log, format the changes into consistent relational XML messages, and publish them to WebSphere MQ. Any application or service that integrates either with WebSphere MQ directly or supports Java™ Message Service (JMS) can asynchronously receive the data changes as they occur.

WIIEP solutions enable database events to initiate business processes. For example, a change in an inventory value could be used to drive a product restocking workflow, or the addition of a new customer could initiate welcome e-mail, credit verification, and accounting updates. This creates an application-independent, loosely coupled integration that is more adaptable to changing application environments. For example, while multiple applications may impact the value of the inventory level, a single point of integration — the data items themselves — is driving the workflow. Changes to the applications that impact the inventory level can be made with no impact on the event driven integration.

Additionally, WIIEP solutions can deliver changes to Extract, Transform, and Load (ETL) tools, or custom-built processes for updating operational data stores, or data warehouses minimizing bandwidth requirements, keeping target databases more closely in sync. Thus, businesses can utilize information that is far more current for tactical and operational decision-making.

WIIEP solutions include the following products:

- IBM WebSphere Information Integrator Event Publisher for DB2 Universal Database™ for z/OS V8.2 supports the capture, formatting, and publishing of DB2 UDB for z/OS database events.

  Note: The event publisher function is also available as part of WebSphere Information Integrator Replication for z/OS (Program number 5655-L88) which includes Q replication, Event Publishing, and SQL replication.

  This product is not used in our business scenario, and is not discussed in this redbook.

- IBM WebSphere Information Integrator Classic Event Publisher (WIICEP) for IMS V8.2 supports the capture, formatting, and publishing of IMS database events. It supports IMS Version 7.1 or later.

- IBM WebSphere Information Integrator Classic Event Publisher for CA-IDMS™ V8.2 supports the capture, formatting, and publishing of CA-IDMS database events. It supports CA-IDMS r14 and r15.

  This product is not used in our business scenario, and is not discussed in this redbook.

- IBM WebSphere Information Integrator Classic Event Publisher for Software AG Adabas™ supports the capture, formatting, and publishing of Software AG Adabas database events. It supports Software AG Adabas 7.1.3 or 7.4.3.
This product is not used in our business scenario, and is not discussed in this redbook.

- IBM WebSphere Information Integrator Classic Event Publisher for VSAM V8.2 supports the capture, formatting, and publishing of VSAM events made through CICS®. It requires CICS Transaction Server for z/OS, V2.2, or V2.3.

This product is not used in our business scenario, and is not discussed in this redbook.

**Note:** WIICEP works on z/OS, Version 1.4 or later; as well as WebSphere MQ for z/OS, Version 5.3.

The application that receives messages from WIICEP must use one of the following applications — WebSphere MQ, Version 5.3; or WebSphere Business Integration Event Broker, Version 5.0

The main features of WebSphere Information Integrator Classic Event Publisher for IMS include the following capabilities:

- It captures database changes in IMS environments by reading the recovery log, formats the changes into XML messages in UTF 8 format, and publishes them to WebSphere MQ.

**Note:** In PTF UK09686 released on December 5th 2005, messages may also be published in native EBCDIC format.

Supported environments and program types are listed in Table 1-1.

<table>
<thead>
<tr>
<th>IMS environment</th>
<th>Databases supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB Batch</td>
<td>Full function</td>
</tr>
<tr>
<td>DB/DC</td>
<td>Full function and DEDB</td>
</tr>
<tr>
<td>DBCTL</td>
<td>Full function DEDB</td>
</tr>
</tbody>
</table>

Changes can be captured for the following full function databases:

- HISAM
- HDAM
- HiDAM
- SHISAM
- DEDB
- PHDAM
- PHIDAM

**Note:** Data capture is supported only when the batch job allocates a non-dummy IEFRDER DD statement.

The IEFRDER is a special DD statement that is used to accept input from the operator console when issuing a START or MOUNT command. The value specified from a MOUNT or START command overrides or adds to the source IEFRDER DD statement in the cataloged procedure.

It also supports updates from CICS applications, WebSphere Information Integrator Classic Federation, or ODBA clients using DRA.
It supports an IMS data sharing environment.

Each message contains changes from a single type of data source (for example, only IMS changes, VSAM changes, or CA-IDMS changes). Each message can contain an entire transaction or only a row-level change.

Data can be formatted into either transaction messages or row messages depending upon the needs of the application.

- With transactions a one or more messages (depending upon the setting of the MAX TRANSPORT MESSAGE SIZE configuration parameter) is published for each committed transaction that affects the source table. The message contains all of the changes made to the logical source table by the transaction, and also includes all the changes made to any other logical source tables updated in the same transaction that are published through the same message queue.

- With row messages, a message is published for each committed row operation on the source table.

Database changes being published can be filtered such that only the data that the receiving application is interested in are published. You can control which fields within IMS segments, VSAM files or CA-IDMS files will be monitored for changes using a metadata catalog to identify specific data items to be captured and published.

This metadata catalog also defines how the individual data items are to be reformatted into relational data types. This relational mapping results in “logical” IMS, VSAM and CA-IDMS tables.

Messages can be tailored to include only the changed data that the application requires.

Messages can be tailored to include the previous and current values for all columns included in the message. This only applies to update operations.

Easy-to-use wizards are provided to simplify the definition of event publishing.

WIICEP components (more later) may be installed on a single z/OS image or spread over two z/OS images.

The following sections describe in greater technical detail the architecture and main features of WebSphere Information Integrator Classic Event Publisher (WIICEP) for IMS:

- WIICEP for IMS architecture
- WIICEP processing flow
- WIICEP monitoring tools
- WIICEP for IMS recovery mode

1.2 WIICEP for IMS architecture

Figure 1-1 shows an overview of WIICEP architecture.
The various components in this architecture are:

- **Data sources**, which may be IMS, VSAM, and IDMS:
  Changes are captured from the log buffers and log files for IMS and IDMS, while they are captured from the CICS logs in the case of VSAM.
  Additional sources will be supported in future offerings.

- **Change Capture Agents (CCA):**
  A CCA is a module¹ that runs on the same machine as the database. A CCA is associated with each data source, and it intercepts changes occurring in these data sources and forwards them to the correlation service.

  A CCA monitors the database, looking for actions that affect data. When a table is affected by an INSERT, DELETE, or UPDATE operation, the change-capture agent checks² the WebSphere Information Integrator Classic Event Publisher metadata catalog to determine if the data affected is in monitored tables, and if so, sends the raw data for the candidate change to the correlation service module.

  There are two change-capture agents per database environment:
  - **Active change capture:**
    This agent captures changes directly from the log buffer as it is created.
  - **Recovery change capture:**
    This agent captures changes from the log files during recovery mode (more on this later).

  This module can be configured to run either in the database, where the recovery agent will be run as a batch job, or as a service that runs within the correlation service. The configuration of this module varies depending upon the database (such as IMS, CA-IDMS, or Software AG Adabas) being monitored.

¹ A piece of code
² This does not apply to IMS, where no filtering is done and all changed data is sent to the correlation service.
Correlation service:

The correlation service receives raw data for database changes from the CCAs (one or more CCAs can be associated with a single correlation service), matches the changes against user-mapped tables in the WebSphere Information Integrator Classic Event Publisher metadata catalog, and then sends the data to the publication service after the changes are committed. If the “logical” table that was affected is mapped in the metadata catalog and is flagged for data capture, the correlation service stores the data until it receives a COMMIT statement.

**Note:** You add information to the metadata catalog using the Data Mapper and the metadata utility.

Each piece of change information that the correlation service stores is kept until a COMMIT or ROLLBACK action is performed on the unit of work:

- At COMMIT, the correlation service creates one or two SQLDAs for each change to a monitored table. These SQLDAs are sent to the publication service. The information contained in the SQLDAs that are passed to the publication service is described in more detail in Section 1.4, “WIICEP for IMS processing flow” on page 34.
- If a ROLLBACK is detected for a unit of work, all data currently stored for that unit of work is deleted.

The correlation service can be warm started or cold started by modifying configuration parameters during startup:

- A warm start (default) retains the state of all known CCAs at the time the correlation service was last shut down. The restart information is stored in GDG files.
- A cold start discards all recovery information and places all known agents in active mode (more on this later).

**Attention:** The correlation service resides on the same machine as the CCA in a different address space. On z/OS, the CCAs and the correlation service must be on the same LPAR.

All this processing is handled in the correlation service, so that the CCAs have minimal impact on the database environment.

Publication service:

Each correlation service has one corresponding publication service and vice versa.

The publication service supports full logging. A publication service performs the following tasks:

a. Receives data in SQLDA format about changes to source tables from the correlation service.

b. Matches the data against the publications configured.

c. Converts the SQLDA data into messages in an XML format.

d. Puts the XML messages on the local, persistent WebSphere MQ message queues specified for your publications.

The publication service uses a local persistent WebSphere MQ message queue called the in-doubt resolution queue which ensures that recovery point information for the publication service is not lost if the correlation service is shut down. The in-doubt resolution queue is used to record the last unit of recovery (UOR) on a per CCA basis that is processed by the publication service. Once all of the changes for a UOR have been written to the
WebSphere MQ message queues, the in-doubt resolution queue is updated with information about the transaction that has just been processed, and a WebSphere MQ commit is performed. If the WebSphere MQ commit succeeds, the transaction is assumed to have been processed successfully and a confirmation message is sent back to the correlation service. If a write to a WebSphere MQ queue fails or the WebSphere MQ commit fails, a recovery situation has been encountered and the contents of the in-doubt resolution queue is used to identify where processing needs to begin in the IMS log files.

- **Metadata catalog:**
  The metadata catalog is used by a data server to facilitate translation of the data from the non-relational data structure into relational columns. It maps a non-relational data structure such as an IMS segment to an equivalent “logical” table that will be published. The Data Mapper tool described in the following list facilitates the creation of the metadata catalog.

  The metadata catalog identifies the data that is to be captured. A table must be ALTERed to turn on data capture in order for updates to those databases and segments to be processed by the correlation service and sent to the publication service. Changes received for other databases/segments are simply discarded by the correlation service.

- **WebSphere MQ:**
  WebSphere MQ is the transport mechanism for messages originating in the data sources that need to be delivered to target applications.

- **XML messages:**

  **Note:** In PTF UK09686 released on December 5th 2005, messages may also be published in native EBCDIC format.

  XML messages may be data messages, informational messages or control messages. Full details of the structure of XML messages for event publishing are available in *IBM DB2 Information Integrator Q Replication and Event Publishing Guide and Reference Version 8.2*, SC18-7568.

  **Attention:** WIICEP for IMS only publishes data messages. Schema messages are generated by the correlation service, and used by the publication service, but they are not propagated further by the publication service.

- **XML receiver process or stored procedure:**
  The XML messages generated by WIICEP may be consumed using a variety of techniques and tools including WebSphere Business Integration Message Broker, IBM WebSphere DataStage® (formerly Ascential® Datastage), Java application with the XML Publishing Toolkit, and DB2 Message Queue Listener. The XML event message consumers generally tend to update target databases with the changed-data retrieved from the data sources.

  The XML Publishing Toolkit is a set of Java classes that perform a majority of the basic tasks required to process an XML event message. It is shipped in the ..\sqllib\samples\repl directory of the WebSphere Information Integrator products for Linux®, UNIX® and Windows® as asnqwxml.zip file.
The XML Publishing Toolkit comes with the following components:

- A readme.txt file that contains the installation instructions for the toolkit.
- A doc directory containing the Javadoc™ for the toolkit.
- A src.zip file containing the toolkit source code. The entire toolkit is considered a sample — the source is therefore available for reference and modification.
- A set of sample directories. The samples are documented in the original “Event Publishing made Easy” article at:

Control information:

WIICEP contains many control items to manage the normal operations and recovery operations (after a failure) of the IMS event publishing environment. A brief description of each of these items follows — a more detailed description of their individual roles is described in Section 1.4, “WIICEP for IMS processing flow” on page 34.

- CSA storage for state information:
  WIICEP uses CSA storage to maintain the state of the WIICEP environment, and as a vehicle for communication between the CCAs and the correlation service.

-Optional Recovery information data set for IMS control region:
  This is a data set where the CCA running in the IMS control region records a restart point when the IMS control region is started without the correlation service running.

  **Attention:** Even though this is an optional data set, we strongly recommend that it be used for effective recovery processing as described in Section 1.6, “WIICEP for IMS recovery mode” on page 44.

- Correlation service restart information GDG data sets:
  When the correlation service is shut down normally, it records the restart point for a subsequent warm restart of the correlation service in this GDG data set.

- Publication service in-doubt resolution WebSphere MQ queue:
  The publication service uses a local persistent WebSphere MQ queue to store appropriate information to re-synchronize the processing between the correlation service and the publication service after a failure.

Data Mapper:

The Data Mapper tool is a Microsoft® Windows application that automates many of the tasks required to create a typical relational table from non-relational data structures. It accomplishes this by creating metadata grammar from existing non-relational data definitions (COBOL copybooks, CA-IDMS schema and subschema definitions, and IMS DBDs). The metadata grammar is used as input to the metadata utility to create metadata catalogs.

Figure 1-2 provides an overview of the steps involved in creating a metadata catalog. IMS DBDs and COBOL copybooks are imported from the z/OS platform to the Microsoft Windows platform using Data Mapper import utilities. This IMS non-relational structure information is then mapped using the Data Mapper GUI into relational “logical” tables. USE Grammar is then generated which is file transferred to the z/OS platform. The metadata utility processes this USE Grammar to generate the metadata catalog.

“STEP 6A2: Use Data Mapper to generate USE Grammar for AUTODDB” on page 108 describes the steps in creating the metadata catalog for the Trustworthy insurance industry scenario using the Data Mapper tool.
Chapter 1. WebSphere Information Integrator Classic Event Publisher for IMS architecture

Figure 1-2  Metadata catalog creation using Data Mapper

Data Mapper functions include:

- Creating data catalogs, where a data catalog is a collection of tables for a particular non-relational database type, such as IMS DB or VSAM.
- Creating a table, where a relational table may map one or more data structures from a non-relational data structure into a single WebSphere Information Integrator Classic Federation for z/OS or WebSphere Information Integrator Classic Event Publisher table, referred to as a "logical" table.
- Creating a column (optional), where a column can represent one or more data items in the corresponding non-relational data structure.
- Loading DBDs for reference (IMS only). This allows the Data Mapper to use the information as a reference when creating relational data. The Data Mapper does not store IMS DBD source, so you must reload the source each time you open a repository.
- Loading schemas and subschemas for reference (CA–IDMS only). This allows the Data Mapper to use the information as a reference when creating relational data. The Data Mapper does not store CA–IDMS source, so you must reload the source each time you open a repository.
- Generating metadata grammar, which is also known as USE statements or USE Grammar, for all of the tables in a specific data catalog. After metadata grammar has been created, it must subsequently be transferred from the workstation to the mainframe.

The metadata grammar is supplied as input to the metadata utility that runs on the mainframe. The metadata utility uses the contents of the metadata grammar to create "logical" tables. Client applications use "logical" tables, which are non-relational-to-relational mappings, for SQL access non-relational data. From a WIICEP perspective, the "logical" tables to a large extent define the contents of the XML message. The columns in the "logical" tables define each user data item, and determine the data type of the changed data that is published in the XML message.

– File transfer of data from the workstation to the mainframe. The Data Mapper facilitates file transfers from the workstation to the mainframe through its built-in FTP facility. You can use the built-in FTP to transfer copybooks or DBD source and generated metadata grammar.

– Creating relational views. By transferring the metadata grammar to the host system and running the metadata utility with this input, you create a relational view of non-relational data. This relational view is ultimately used to enable SQL access to the mapped non-relational data structure and defines the format of the published data.

– Creating indexes. A logical index is a logical SQL index that maps an existing physical index on a target database or file system, such as IMS DB or VSAM.

Note: Index definitions are not required for event publishing purposes.


1.3 WIICEP for IMS configuration

Figure 1-3 describes the main steps in configuring your environment for WIICEP for IMS. These steps are described in further detail in this section.

Figure 1-3  Main steps in configuring your WIICEP for IMS environment
1.3.1 STEP 1: Choose topology for the WIICEP for IMS environment

There are two choices for configuring the WIICEP for IMS environment:

- Both the correlation service and publication service are collocated on the same z/OS Image. This option is generally chosen when:
  - The single z/OS image has sufficient CPU, I/O, and memory capacity to accommodate the additional resource demands of the publication service.
  - Software licensing costs can be reduced by leveraging an existing WebSphere MQ installation on another z/OS image.

- Publication service is located on a different z/OS image. This option is appropriate when the conditions for the single z/OS image do not apply.

1.3.2 STEP 2: Install WIICEP for IMS

Installation of WIICEP for IMS is a routine process and is therefore not covered here. Full details of installing WIICEP is described in the *IBM DB2 Information Integrator Installation Guide for Classic Federation and Classic Event Publishing Version 8.2*, GC18-9301-01, and the *Program Directory for DB2 Information Integrator Classic Event Publisher for IMS*.

1.3.3 STEP 3: Configure the WIICEP for IMS environment

A number of steps are involved in configuring the WIICEP for IMS environment, as shown in Figure 1-4.

![Figure 1-4 WIICEP for IMS configuration steps](image-url)

Each of the steps shown in Figure 1-4 is described briefly in the following sections.
STEP 3a: Download and install the Data Mapper client on Windows

WIICEP provides a runtime client and the Data Mapper administration tool. The following three components may be installed:

- **ODBC Client:**
  
  The ODBC client only works with WIICEP metadata catalog tables. It does not support SQL access to the “logical” tables defined in the metadata catalog. The WebSphere Information Integrator Classic Federation product supports SQL access to the “logical” tables. The ODBC client setup registers the WebSphere Information Integrator Classic Federation ODBC driver in Windows administrative settings.

- **JDBC Client:**
  
  The JDBC client is similar to the ODBC client in functionality. The JDBC driver is simply a .jar file (cacjdbc21.jar) that is copied to the file system. This .jar file is included at setup time in the CLASSPATH of the environment variables.

- **Data Mapper administration tool:**
  
  The Data Mapper administration tool is used to populate the metadata catalog with definitions of “logical” tables for use as change capture sources. These “logical” tables are mapped from the underlying non-relational structures such as hierarchical database structures in the case of IMS.

These components are shipped with the z/OS libraries, and must be downloaded to a Windows PC for installation. The Windows client tools are located in the PDS member CAC.V8R2M0.SCACCLNT(CAC82WN).

**Note:** The Windows client tools are also available for download from:

ftp://service2.boulder.ibm.com/

The customer needs to contact Technical Support to obtain login information.

The steps to install Data Mapper are as follows:

1. Download the components using your 3270 emulator tools (such as “IBM Personal Communications”) to a directory on your Windows machine. Ensure that the Transfer Type is binary from the drop down list and saved as CAC82WN.EXE in a temporary directory such as c:\code as shown in Figure 1-5.

**Note:** The same PDS contains a member called CACREAD which contains PDF manuals of all the WebSphere Information Integrator Classic Federation and Classic Event Publishing manuals. Download this (in binary) to a zip file, and extract the PDF manuals.

Click **Receive** to initiate the transfer.
2. Execute the downloaded file.

Run the installer by executing the cac82wn.exe. Accept the default installation location, and install all the product components. Click Next on the confirmation screen of the installation actions to proceed with the install as shown in Figure 1-6.
After the installation is completed, you can proceed to use the Data Mapper tool to map IMS database structures to “logical” tables.

**STEP 3b: Use Data Mapper to generate USE Grammar**

The Data Mapper tool assists in generating “logical” table definitions for the metadata catalog. The logical table definitions are referred to as USE Grammar because they define how the underlying hierarchical database will be used as a “logical” relational table for change capture.

The mechanics of USE Grammar are the same for WebSphere Information Integrator Classic Federation and WebSphere Information Integrator Classic Event Publishing. The differences are as follows:

- When a “logical” table is to be used for event publishing purposes, the “logical” table must be defined with the “Data Capture Changes” clause.
- A “logical” table must be defined for each SEGM in a DBD that has been augmented to generate data capture log records.
- The column definitions in the “logical” table for a SEGM must be consistent with the data that is being generated in the data capture log records. For example, for a child segment, if you are only capturing concatenated key and segment image data, the mapping can not contain a column that references non-key data in parent segments.

**Note:** Consistency checking between the table mapping and the data capture options is performed by the correlation service during initialization processing. Some mapping inconsistencies are tolerated with information that is captured when a cascade delete occurs. In these situations the mapping can reference non-key information in parent segments, and yet only capture concatenated key information when a cascade delete occurs.

The process of using the Data Mapper tool to create USE Grammar for an IMS database involves the following steps:

1. Create a new repository.
2. Load a DBD file.
3. Create Schema and IMS logical tables.
4. Generate USE Grammar.

An example of mapping a “logical” table using IMS database structures is described in “STEP 6A2: Use Data Mapper to generate USE Grammar for AUTODB” on page 108.

**STEP 3c: Modify IMS DBDs to support event publishing**

All updates to an IMS database are logged as “type 50” log records, which contain sufficient information to enable database restart and recovery operations. Event publishing requires additional information to be captured such as the complete “before image” of a record before and update or a delete.

Log records with this additional information is written as “type 99” on subsets of DBDs and/or SEGM for which event publishing is required. Such “type 99” log records are generated by IMS whenever the “EXIT=” keyword is specified on the DBD and/or SEGM definitions.
Figure 1-7 shows the IMS sample DI21PART database hierarchic structure.

The “EXIT=” keyword may be specified in the DBD definition, which would generate “type 99” log records for every change to the database, or specified on one or more SEGM definitions to limit the generation of “type 99” log records to only changes made to those segments. This is desirable only if a subset of the segments are required for change capture, because it avoids unnecessary “type 99” log record generation overheads for segments not needed in change capture.

For the DI21PART database, in order to only capture changes from the STOKSTAT segment, plus the concatenated key from the PARTROOT segment, the “EXIT=” keyword is specified on the PARTROOT (data changes) and STOKSTAT (data and key information) SEGM definition as shown in Example 1-1.

**Example 1-1  DI21PART DBD definition with EXIT= keyword in SEGM definitions**

<table>
<thead>
<tr>
<th>DBD</th>
<th>NAME=DI21PART, ACCESS=(HISAM, VSAM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATASET</td>
<td>DD1=DI21PART, DEVICE=3380, OVLFW=DI21PARO,</td>
</tr>
<tr>
<td></td>
<td>SIZE=(2048, 2048), RECORD=(678, 678)</td>
</tr>
<tr>
<td>SEGM</td>
<td>NAME=PARTROOT, PARENT=0, BYTES=50, FREQ=250,</td>
</tr>
<tr>
<td></td>
<td>EXIT=(*, NOKEY, DATA, NOPATH, (NOCASCADE), LOG)</td>
</tr>
<tr>
<td>FIELD</td>
<td>NAME=(PARTKEY, SEQ), TYPE=C, BYTES=17, START=1</td>
</tr>
<tr>
<td>SEGM</td>
<td>NAME=STANINFO, PARENT=PARTROOT, BYTES=85, FREQ=1</td>
</tr>
<tr>
<td>FIELD</td>
<td>NAME=(STANKEY, SEQ), TYPE=C, BYTES=2, START=1</td>
</tr>
<tr>
<td>SEGM</td>
<td>NAME=STOKSTAT, PARENT=PARTROOT, BYTES=160, FREQ=2,</td>
</tr>
<tr>
<td></td>
<td>EXIT=(*, KEY, DATA, NOPATH, (NOCASCADE), LOG)</td>
</tr>
<tr>
<td>FIELD</td>
<td>NAME=(STOCKEY, SEQ), TYPE=C, BYTES=16, START=1</td>
</tr>
<tr>
<td>SEGM</td>
<td>NAME=CYCCOUNT, PARENT=STOKSTAT, BYTES=25, FREQ=1</td>
</tr>
<tr>
<td>FIELD</td>
<td>NAME=(CYCLKKEY, SEQ), TYPE=C, BYTES=2, START=1</td>
</tr>
<tr>
<td>SEGM</td>
<td>NAME=BACKORDR, PARENT=STOKSTAT, BYTES=75, FREQ=0</td>
</tr>
<tr>
<td>FIELD</td>
<td>NAME=(BACKKEY, SEQ), TYPE=C, BYTES=10, START=1</td>
</tr>
<tr>
<td>DBDGEN</td>
<td>FINISH</td>
</tr>
<tr>
<td>END</td>
<td></td>
</tr>
</tbody>
</table>
The parameters associated with the EXIT= keyword are described in Table 1-2.

### Table 1-2 Parameters of the EXIT keyword

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exit-Name</td>
<td>In this parameter, you specify:</td>
</tr>
<tr>
<td></td>
<td>- Name of the DPropNR synchronous data capture exit, if there is one,</td>
</tr>
<tr>
<td></td>
<td>- &quot;*&quot; to indicate that there is no exit, or</td>
</tr>
<tr>
<td></td>
<td>- NONE to deactivate an exit routine on a SEGM statement</td>
</tr>
<tr>
<td></td>
<td>WIICEP for IMS does not use data capture exits, but co-exists if your site is using DPropNR, or if you have implemented your own exits at your site. If you do not have any data capture exits, set this parameter to *.</td>
</tr>
<tr>
<td>KEY</td>
<td>NOKEY</td>
</tr>
<tr>
<td>DATA</td>
<td>NODATA</td>
</tr>
<tr>
<td>PATH</td>
<td>NOPATH</td>
</tr>
<tr>
<td>CASCADE</td>
<td>NOCASCADE</td>
</tr>
<tr>
<td></td>
<td>- KEY</td>
</tr>
<tr>
<td></td>
<td>- DATA</td>
</tr>
<tr>
<td></td>
<td>- PATH</td>
</tr>
<tr>
<td>LOG</td>
<td>NOLOG</td>
</tr>
</tbody>
</table>

On the PARTROOT segment, the EXIT keyword parameters have the following meaning:

- "*" should be left as is for WIICEP for IMS.
- NOKEY specifies that the physical path concatenated key information should not be captured — pointless for the root segment.
- DATA specifies that physical segment data should be captured.
NOPATH specifies that the physical segment data should not be captured for parent segments.

NOCASCADE specifies that cascade delete changes should not be captured.

LOG specifies that “type 99” log records should be generated to the IMS log files.

On the STOKSTAT segment, the EXIT keyword parameters have the following meaning:

- "*" should be left as is for WIICEP for IMS.
- KEY specifies that the physical path concatenated key information should be captured — usually required for dependent segments.
- DATA specifies that physical segment data should be captured.
- NOPATH specifies that the physical segment data should not be captured for parent segments.
- NOCASCADE specifies that cascade delete changes should not be captured.
- LOG specifies that “type 99” log records should be generated to the IMS log files.

After these changes are saved, the following four jobs must be run to effect the changes made in the IMS system:

1. DBDGEN to change the DBD definitions
2. PSBGEN for the PSB definitions
3. ACBGEN to update the ACBs
4. Copy the staging libraries to the active libraries.

The job to copy staging libraries to active libraries is found in the installation verification program (IVP) libraries. For an IMS V8.2 system, the job name is INSTALIB(IV3E316J).

This concludes the modification of the DBD to generate the correct “type 99” log records for WIICEP for IMS.

**STEP 3d: Install WIICEP for IMS Active CCA**

The IMS active CCA is implemented as the IMS Logger Exit that has a hard-coded name of DFSFLGX0. IMS automatically invokes the IMS Logger Exit during IMS log file initialization processing if a module named DFSFLGX0 is found in the STEPLIB concatenation or the link-pack area.

Two methods are available to install the IMS active CCA, as follows. Each approach has its advantages and disadvantages. Generally you will decide which approach to take based upon how pervasive the use of Classic Event Publisher will be at your site.

- Copy module DFSFLGX0 from the Classic Event Publisher distribution libraries into the IMS SDFSRESL.

  This approach is appropriate and easiest when you are implementing a large-scale deployment indicating that you are either:
  - Planning to augment the majority of your IMS databases for change capture, or
  - Augmenting an IMS database for change capture that is updated by the majority of your IMS applications.

  If this approach is erroneously used in a smaller-scale deployment:
  - The correlation service still tracks all IMS control regions that are referencing the IMS SDFSRESL where the IMS active CCA is installed, even though many of these IMS applications do not update databases that are being monitored by WIICEP for IMS.
If these IMS active CCAs go into recovery mode (more on this later), you have to recover these failed agents, even though no IMS changes are being captured. This places an additional burden of management.

- Concatenate the Classic Event Publisher load library into your IMS batch jobs and started task procedures for the online DB/DC or DBCTL regions.

  This approach is appropriate for smaller-scale deployments where only a small number of IMS databases are monitored and updated by a small number of IMS applications.

  If this approach is erroneously used in a larger-scale deployment, you need to update each IMS batch job and DB/DC or DBCTL subsystems’ started task JCL to include a recovery data set (more later). If you forget to update one of your IMS applications that updates a monitored database, these changes are lost and the correlation service has no knowledge that this has occurred.

  For IMS jobs that have been updated to include the Classic load library in the concatenation, the JCL should also be updated to include the recovery dataset. The problem with this approach is that if you accidentally overlook a job, the CCA is never invoked and it is therefore as if the job never even existed from the WIICEP for IMS perspective.

**Attention:** The IMS Logger Exit is somewhat of an esoteric IMS system exit. IBM does not supply a sample for this exit, and it is therefore rarely used. However, in case you have implemented your own IMS Logger Exit or are using an exit from another company, the supplied version of the IMS Logger Exit does contain support for invoking an existing IMS Logger Exit. The SCACSAMP member CACIMLEX is a sample relink job that will create a backup of your Logger Exit, and then relink our version of the exit with yours. Your version of the IMS Logger Exit must be named DFSFLGX0 for the call to succeed.

WIICEP for IMS supplies a module called DFSFLGX0 in the CAC.V8R2M0.SCACLOAD library, which may be used in accordance with whichever method is used to install the IMS active CCA.

**STEP 3e: Allocate recovery data set and update IMS System Proc**

An optional modification to the IMS region JCL provides for recovery information. When an IMS region is started without a correlation service running, the CCA running in the region records a restart point to a data set if one is provided.

This 80-byte LRECL data set must be allocated and referenced by a CACRCV DD statement. The CACRCV DD statement must be added to the DB/DC or DBCTL started task JCL or into IMS batch job JCL.

A unique data set name must be created for each IMS job that a CCA will be active in.

Section 1.6.1, “Recovery mode overview” on page 45 provides a detailed description of the role of this recovery data set in recovery operations.

Example 1-2 shows the recommended allocation for the recovery data set named CAC.RECOVDS.IMS810K for an active CCA.
Example 1-2  Recovery data set allocation

Data Set Information

Command ==>  

Data Set Name . . . : CAC.RECOVDS.IMS810K

General Data                          Current Allocation
Volume serial . . . : TSTO10          Allocated cylinders : 1
Device type . . . . : 3390            Allocated extents . : 1
Organization . . . : PS
Record format . . . : FBS
Record length . . . : 80
Block size . . . . : 5120
1st extent cylinders: 1               Used cylinders . . . : 0
Secondary cylinders : 1
Creation date . . . : 2005/07/22
Referenced date . . : 2005/07/25
Expiration date . . : ***None***

After the recovery data set has been allocated, the IMS starter task JCL must be updated to add a DD card to reference this data set. The IMS starter task JCL is in SYS1.PROCLIB(IMS810K), and should contain a DD card (DSN name would vary depending upon your installation) as follows:

//CACRCV  DD DISP=SHR,DSN=CAC.RECOVDS.IMS810K

STEP 3f: Allocate and populate metadata catalog with USE Grammar

The WIICEP for IMS metadata catalog is stored in a VSAM data set which must be created and then populated as follows:

- Create the metadata catalog

  WIICEP for IMS provides a sample job in CAC.V8R2M0.SCACSAMP(CACCATLG) to allocate the metadata catalog.
  - Edit the JCL to provide system-specific values for the PROC variables HLQ, DISKU and DISKVOL.
  - Uncomment the CACCAT and CACINDX DD cards and specify data set names that you want.

  Example 1-3 shows a sample CACCATLG job for creating the metadata catalog.


Example 1-3  Allocating the metadata catalog sequential data set

//CACCATLG JOB 'ACCNT#',REGION=4M,NOTIFY=&SYSUID,MSGLEVEL=(1,1)
//********************************************************************
//*                                                                  *
//*   CACCATLG - SAMPLE JCL TO ALLOCATE META DATA CATALOGS           *
//*                                                                  *
//********************************************************************
//*ALLOC1   PROC CAC='CAC',         INSTALLED HIGH LEVEL QUALIFIER

Attention: The sample JCL in the manual creates a sequential metadata catalog. Although a VSAM linear dataset can be used for the metadata catalog, sequential metadata catalogs work just fine in an event publishing environment since the correlation service only reads the catalog during initialization processing and then caches all the tables marked for change capture.
Populate the metadata catalog

WIICEP for IMS provides a sample job in CAC.V8R2M0.SCACSAMP(CACMETAU) to populate the metadata catalog.

- Edit the JCL to provide system-specific values for the PROC variables CAC, IMS and GRAMMAR.
- Verify that STEPLIB and CTRANS DD cards point to the correct data sets
- Uncomment the CACCAT and CACINDX DD cards and specify data set names allocated earlier
- Edit the SYSIN DD card, to point it at the USE Grammar files produced earlier with Datamapper.

Example 3-20 on page 144 shows the CACMETAU job used to populate the metadata catalog used in the Trustworthy Insurance CIF scenario.

STEP 3g: Allocate GDG data sets to provide restart information for CS

The correlation service keeps track of its restart information in a GDG data set. WIICEP provides a sample job in CAC.V8R2M0.SCACSAMP(CACGDGA) to create GDG datasets, and allocate the first generation.

- Edit the PROC values for the HLQ of the CEP installation libraries (CAC), and DISKU and DISKVL parameters
- Edit the dataset names as required

The ITSO standard for non-SMS generation datasets is for the DSCB model to be on the same volume as the user catalog that contains the catalog alias for the CAC HLQ. Hence, our JCL was modified further to control placement of the models and the datasets as shown in Example 1-4.
Example 1-4  Allocate correlation service restart GDG dataset

//CACGDGA JOB 'ACNT#',REGION=4M,NOTIFY=&SYSUID,MSGLEVEL=(1,1)
//********************************************************************
//*                                                                  *
//* LICENSED MATERIALS - PROPERTY OF IBM                             *
//* 5697-I82                                                          *
//* 5655-M35                                                         *
//* 5655-M38                                                         *
//* 5655-N56                                                         *
//*                                                                  *
//* (C) COPYRIGHT CROSSACCESS 1993, 2003.                            *
//* (C) COPYRIGHT IBM CORP. 2003, 2005.  ALL RIGHTS RESERVED.       *
//* US GOVERNMENT USERS RESTRICTED RIGHTS USE,                      *
//* DUPLICATION OR DISCLOSURE RESTRICTED BY GSA ADP SCHEDULE         *
//* CONTRACT WITH IBM CORP.                                         *
//* DISCLAIMER OF WARRANTIES:                                       *
//*                                                                  *
//* PERMISSION IS GRANTED TO COPY AND MODIFY THIS SAMPLE CODE, AND   *
//* TO DISTRIBUTE MODIFIED VERSIONS PROVIDED THAT BOTH THE COPYRIGHT *
//* NOTICE, - AND THIS PERMISSION NOTICE AND WARRANTY DISCLAIMER    *
//* APPEAR IN ALL COPIES AND MODIFIED VERSIONS.                     *
//*                                                                  *
//* THIS SAMPLE CODE IS LICENSED TO YOU AS-IS.                       *
//* IBM AND ITS SUPPLIERS AND LICENSORS DISCLAIM ALL WARRANTIES,     *
//* EITHER EXPRESS OR IMPLIED, IN SUCH SAMPLE CODE, INCLUDING THE     *
//* WARRANTY OF NON-INFRINGEMENT AND THE IMPLIED WARRANTIES OF       *
//* MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT  *
//* WILL IBM OR ITS LICENSORS OR SUPPLIERS BE LIABLE FOR ANY DAMAGES *
//* ARISING OUT OF THE USE OF OR INABILITY TO USE THE SAMPLE CODE,   *
//* DISTRIBUTION OF THE SAMPLE CODE, OR COMBINATION OF THE SAMPLE    *
//* CODE WITH ANY OTHER CODE. IN NO EVENT SHALL IBM OR ITS           *
//* LICENSORS AND SUPPLIERS BE LIABLE FOR ANY LOST REVENUE, LOST     *
//* PROFITS OR DATA, OR FOR DIRECT, INDIRECT, SPECIAL, CONSEQUENTIAL, *
//* INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED AND REGARDLESS    *
//* OF THE THEORY OF LIABILITY, EVEN IF IBM OR ITS LICENSORS OR      *
//* SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.  *
//*                                                                  *
//********************************************************************
//*                                                                  *
//*   NAME:  CACGDGA                                                 *
//*                                                                  *
//*   JCL TO CREATE GDG DATASETS FOR RECOVERY AND ALLOCATES          *
//*   THE FIRST GENERATION FOR USE BY THE CORRELATION SERVER         *
//*                                                                  *
//*   BE SURE TO CHANGE THE SYMBOLIC VARIABLES AND THE              *
//*   LOWERCASE hlq TO MATCH THE FULLY QUALIFIED DATASET NAMES       *
//*   BEFORE SUBMITTING THIS JOB.                                    *
//*                                                                  *
//* ITSO NOTE:                                                       *
// FOR NON-SMS GENERATION DATASETS                                 *
// THE DSCB MODEL HAS TO BE ON THE SAME VOLUME AS THE USER          *
// CATALOG THAT CONTAINS THE CATALOG ALIAS FOR THE CAC              *
// HIGH LEVEL QUALIFIER.                                            *
//                                                                  *
//* THE GDGS THEMSELVES CAN BE ON ANY VOLUME                        *
//*                                                                  *
//********************************************************************
//CACGDGA PROC CAC='CAC',
// DISKU=SYSDA,
STEP 3h: Configure the JCL and control datasets to run WIICEP for IMS

The Correlation Service and the Publication Service are controlled using Service Information Entries (SIEs) that define all the configuration parameters that drive their operation. A configuration dataset containing the service information must be configured, and then referenced in a DD card by the Job JCL used to initiate the Correlation Service and Publication Service. The same configuration dataset also includes the definition of one or more publications on the “logical” tables identified in the metadata catalog.

WIICEP for IMS provides a sample configuration dataset CAC.V8R2M0.SCACCONF(CACCSCF) which can be edited to suit. Example 1-5 shows an example of a configuration dataset where the correlation service and publication service are collocated on the same z/OS image — it includes the SIE for each service, and the definition of a single publication.
Note: If the correlation service and publication service are not collocated, then a separate configuration dataset must be defined for each service.

In PTF UK09686 released on December 5th 2005:

- The correlation service configuration dataset would contain the correlation service SIE, some of the configuration parameters defined in Table 1-5 on page 28, and the definition of the publications.
- The publication service configuration dataset would contain the publication service SIE and some of the configuration parameters defined in Table 1-5 on page 28.

When the correlation service and publication service are not collocated, the publication service accesses the publication definitions in the correlation service configuration dataset using the TCP/IP transport mechanism.

Example 1-5  SIEs of the correlation service and publication service & a publication definition

```
* CORRELATION SERVICE SIE
SERVICE INFO ENTRY = CACECA2 XM1/XQM/CSQ1/16 2 1 1 16 4 10MS 30S \  
   TCP/WS5C3.ITSO.IBM.COM/5555,CSA=1K,CSARLSE=0,INT=1,COLDSTART

* PUBLICATION SERVICE SIE
SERVICE INFO ENTRY = CACPUB PUB1 2 1 1 100 4 5M 5M \    
    MQI/MQ8G/CAC_RESTARTQ

* XML OVER MQ PUBLISH/SUBSCRIBE
PUB ALIAS=STOKSTAT,
   MSGTYPE=TRANS,
   QUEUE=MQI/MQ8G/IMSK_SENDQ1,
   TABLE=DI21PART.STOKSTAT,
   BEFORE_VALUES=YES

*  
```

Attention: The order of SIEs for the correlation service and the publication service matters. The entry for the correlation service must come before the entry for the publication service. This order is particularly important on shutdown, when services are stopped in LIFO (reverse) order. The publication service must stop first so that it can send the proper quiesce command to the correlation service. If the publication service does not stop first, the correlation service might go into recovery mode (more later) on an otherwise normal shutdown. For this reason, if the publication service is configured to start before its corresponding correlation service, the publication service will fail on startup when it fails to detect that the correlation service exists.

The correlation service SIE has 10 parameters as does the SIE of the publication service. Table 1-3 and Table 1-4 describe the SIE entries we chose for the correlation service and publication service respectively.

The publication described in Example 1-5 is named STOKSTAT with the following attributes:

- MSGTYPE=TRANS specifies that a message is published for each committed transaction that affects the source table. The message contains all of the changes made to the source table by the transaction.
- QUEUE=MQI/MQ8G/IMSK_SENDQ1 specifies the name of the queue where the messages are to be sent.
- TABLE=DI21PART.STOKSTAT specifies the name of the source table for this publication.
BEFORE_VALUES=YES specifies that when a row is updated, the previous values and the current values for all columns are included in the message. This parameter is effective for UPDATE operations only.

<table>
<thead>
<tr>
<th>Positional Parameter</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Task Name - This token specifies that this SIE is for the correlation service. Leave as is.</td>
<td>CACECA2</td>
</tr>
<tr>
<td>2</td>
<td>Service Name - The service name defines the protocol and queue name for receiving raw data changes from active and recovery CCAs.</td>
<td>XM1/XQM/CSQ1/16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>where</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▶ XM1 - specified cross memory services. Leave as is.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▶ XQM/CSQ1 - specifies the cross memory dataspace and queue name, and can be modified to suit any site standards</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▶ 16 - specifies the size (in MB) of the change capture dataspace queue.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>You must define a unique data space or queue name for each correlation service that will be running at any one time.</td>
</tr>
<tr>
<td>3</td>
<td>Service Start Class (leave at 2)</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>Minimum Tasks (leave at 1) A value of 0 is acceptable if you want to manually start the service with an operator command, but changing the value is not recommended.</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>Maximum Tasks (leave at 1)</td>
<td>1</td>
</tr>
<tr>
<td>Positional Parameter</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------</td>
<td>-------</td>
</tr>
<tr>
<td>6</td>
<td>Maximum Connections Per task. This value should be the maximum number of active and recovery agents that the correlation service will service, plus four. The additional connections are used by the publication service and reporting utility. (16 is the recommended default, allowing for change capture agents, publication services, reporting utility etc…)</td>
<td>16</td>
</tr>
<tr>
<td>7</td>
<td>Trace Output Level (leave at 4, or raise to 1 for problem determination)</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>Response timeout - the period that the correlation service will listen for a response before timing out. Keep this value low so that the transport queue is checked frequently for messages arriving from change-capture agents.</td>
<td>10MS where MS is milliseconds</td>
</tr>
<tr>
<td>9</td>
<td>Idle Timeout - this specifies the polling frequency for recovery confirmation messages.</td>
<td>30S where S is seconds</td>
</tr>
</tbody>
</table>
Table 1-4 Publication service SIE parameters

<table>
<thead>
<tr>
<th>Positional Parameter</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Server specific information - multi-valued parameter</td>
<td>TCP/WTSC53.ITSO.IBM.COM/5555,CXA=1K,CSARLSE=3,INT=1,COLDSTART where TCP/WTSC53.ITSO.IBM.COM/5555 is the IP Address/port that defines the queue used by the correlation service for communication with recovery agents and the publication service. Generally, this token defines a TCP/IP connection string to which the publication service connects for receiving change messages. CSA=1K - specifies the number of kilobytes that each server is to allocate for CSA space. In most cases, 1K should be enough to manage change capture on at least 50 tables. Allocates CSA space in bytes for control information. CSARLSE=3 - specifies the number of seconds to wait before attempting to release CSA storage at shutdown. A value of 0 leaves CSA allocated for reuse when the correlation service is restarted. The default value of CSARLSE is 0, which prevents CSA from being released. If CSARLSE is not 0, the server will release CSA only if no other correlation services are still active in the system. INT=1 - specifies the number of changes to process for a committed unit of recovery before checking for incoming change data from active or recovery agents. This parameter prevents large transactions from blocking the incoming raw data queue while sending change messages to the publication service. A value of 0 will not interrupt processing of committed messages to look for incoming raw data messages. 1 is the default and recommended value. COLDSTART - specifies to forget previous event publishing, and start over (this was suitable for our test environment -- WARMSTART is the default and would be appropriate in production environments).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1</th>
<th>Task Name - This token specifies that this SIE is for the publication service. Leave as is.</th>
<th>CACPUB</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Service Name - any string up to 16 characters.</td>
<td>PUB1</td>
</tr>
<tr>
<td>3</td>
<td>Service Start Class (leave at 2)</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>Minimum Tasks (leave at 1)</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>Maximum Tasks (leave at 1)</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>Maximum Connections Per task.</td>
<td>100</td>
</tr>
<tr>
<td>7</td>
<td>Trace Output Level (leave at 4, or raise to 1 for problem determination)</td>
<td>4</td>
</tr>
<tr>
<td>Positional Parameter</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------</td>
<td>-------</td>
</tr>
<tr>
<td>8</td>
<td>Response timeout - the period that the correlation service will listen for a response before timing out.</td>
<td>5M where M is minutes</td>
</tr>
<tr>
<td>9</td>
<td>Idle Timeout - this specifies the polling frequency for recovery confirmation messages.</td>
<td>5M where M is minutes</td>
</tr>
</tbody>
</table>
| 10                   | Server specific information - multi-valued parameter. The tenth parameter specifies the WebSphere MQ message queue to use as the restart queue. The format of the parameter is as follows: `<mqi/queue manager/queue name>` where  
  ▶ queue manager is the name of the local queue manager that manages the message queue.  
  ▶ queue name is the name of the local message queue to use as the restart queue.  
  If your correlation service is running remotely from your publication service, you can follow the name with a comma-delimited communication string to describe how the publication service is to communicate with the correlation service. | MQI/MQZ1/CAC.RESTARTQ where  
  MQI - specifies the communication interface, with which to access the MQ Restart Queue for the Publication Service  
  MQZ1 - specifies the Queue Manager  
  CAC.RESTARTQ - specifies the same of the Publication Service Restart Queue.  
  Note: If the correlation service is running remotely from the publication service, you can follow the name with a comma-delimited communication string to describe how the publication service is to communicate with the correlation service. |
Besides the Service Info Entry and PUB parameters shown in Example 1-5, WIICEP also supports a number of other configuration parameters as shown in Table 1-5.

For further details on all the configuration parameters, refer to *IBM DB2 Information Integrator Operations Guide for Classic Event Publishing Version 8.2, SC18-9157-02.*

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Description</th>
<th>Applies to</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD TEMP SPACE</td>
<td>Optional parameter that defines a temporary data set dynamically allocated by a data server to store the intermediate result set. Temporary data set information is a set of parameters separated by commas. Parameters not specified are set to the defaults. Set this parameter so the resulting file is large enough to hold any intermediate result sets that are generated from a typical query running under a particular data server.</td>
<td>Correlation server</td>
</tr>
<tr>
<td>MESSAGE POOL SIZE</td>
<td>Required parameter that specifies the size of the memory region used for all memory allocation. The number is specified in bytes. The actual workable maximum value should be set to 2MB less than the region size. If the value specified is less than 1MB, 1MB (default) is used. If the amount of storage that can be obtained is less than the value specified, the maximum amount available is obtained.</td>
<td>Publication server</td>
</tr>
<tr>
<td>NL</td>
<td>Required parameter that specifies the language used for text messages produced by WebSphere Information Integrator Classic Event Publisher. US English (default) corresponds to standard English as used in the United States. This parameter is used by the log print utility.</td>
<td>Publication server</td>
</tr>
<tr>
<td>NL CAT</td>
<td>Required parameter that points to the language catalog that contains WebSphere Information Integrator Classic Event Publisher messages in a specified language. It is defined by a DD statement in the start-up procedure. This parameter is used by the log print utility.</td>
<td>Correlation server</td>
</tr>
<tr>
<td>INTERLEAVE INTERVAL</td>
<td>Optional parameter. This value sets the interleaving interval from the query processor. The unit of measurement for this interval is a result set row. When multiple result sets are being processed on the same query processor instance, the interleaving interval controls context switching between users and result sets. For example, if INTERLEAVE INTERVAL is set to 100 (default) then the query processor will context switch between active users on that instance for every 100 rows produced. This parameter is only used in the WebSphere Information Integrator Classic Federation environment.</td>
<td>Correlation server</td>
</tr>
<tr>
<td>MAX TRANSPORT MESSAGE SIZE</td>
<td>Specifies the largest size (in bytes) that a message can be before the publication service writes it to a message queue. Default is 128 KB.</td>
<td>Publication server</td>
</tr>
<tr>
<td>PUB</td>
<td>Required parameter used to specify the parts of a publication as described in Example 1-5 on page 23.</td>
<td>Correlation server</td>
</tr>
<tr>
<td>SAF EXIT</td>
<td>Optional parameter used to specify the SAF system exit that will perform authorization checks for the connector associated with a data server or execute a stored procedure application program. Default is NONE. This parameter is only used in the WebSphere Information Integrator Classic Federation environment.</td>
<td>Correlation server</td>
</tr>
<tr>
<td>SERVICE INFO ENTRY</td>
<td>Required parameter used in server configuration files to inform the region controller task that a service should be activated and how that service should be controlled as described in Example 1-5 on page 23. It is valid only in server and enterprise server configuration files.</td>
<td>Correlation server</td>
</tr>
</tbody>
</table>
STEP 3i: Start WIICEP for IMS

WIICEP for IMS provides sample JCL in CAC.V8R2M0.SCACSAMP(CACCS) to start the Correlation Service and Publication Service.

The steps involved are as follows:

- Edit the CACCS member.
- Start WIICEP for IMS.

**Edit the CACCS member**

Copy the CACCS member to SYS1.PROCLIB and edit the JCL as follows:

- Edit the PROC parameters to supply HLQ of CEP (CAC) and IMS (IMS810K).
- Uncomment and review the DD Cards for IMS datasets.
Add DD Cards for WMQ in the //STELIB DD statement as follows:

```plaintext
// DSN=MQ531.SCQANLE,DISP=SHR
// DSN=MQ531.SCQAUTH,DISP=SHR
```

Edit VHSCONF DD to point to the WIICEP for IMS configuration file.

Edit CACCAT and CACINDEX DD cards to point to the metadata catalog.

Edit CACRCVD and CACRCVX DD cards to point to the GDG restart datasets for the correlation service.

Example 1-6 shows an example of a modified CACCS member.

**Example 1-6  Modified CACCS member**

```plaintext
//CACCS PROC CAC='CAC',
//             CONFIG=CACCSCF,
// IMS='IMS810K',
// DISKU=SYSDA,
// DISKVO=TSTO26,
// SOUT='*'
// RGN=32M
//*
//IEFPROC EXEC PGM=CACCNTL,TIME=1440,REGION=&RGN
//STELIB DD DISP=SHR,DSN=&CAC..V8R2M0.SCACLOAD
// DD DISP=SHR,DSN=&IMS..SDFSRESL
// DD DSN=MQ531.SCQANLE,DISP=SHR
// DD DSN=MQ531.SCQAUTH,DISP=SHR
//VHSCONF DD DISP=SHR,DSN=&CAC..V8R2M0.REDBOOK(&CONFIG)
//CTRANS DD DISP=SHR,DSN=&CAC..V8R2M0.SCACSASC
//*
// META DATA CATALOGS
//CACCAT DD DISP=SHR,DSN=&CAC..V8R2M0.CATALOG
//CACINDEX DD DISP=SHR,DSN=&CAC..V8R2M0.CATINDEX
//*
// RECOVERY DATASETS
//CACRCVD DD DISP=SHR,DSN=&CAC..V8R2M0.CACRCVD(0)
//CACRCVX DD DISP=SHR,DSN=&CAC..V8R2M0.CACRCVX(0)
//*
// IMS DBD LIBRARY
//DBDLIB DD DISP=SHR,DSN=&IMS..DBDLIB
//*
//SYSOUT DD SYSOUT=&SOUT
//SYSTERM DD SYSOUT=&SOUT
//SYSPRINT DD SYSOUT=&SOUT
//SYSDUMP DD DUMMY
//CACLOG DD DISP=(NEW,PASS,KEEP),DSN=&LOG,
// UNIT=&DISKU,SPACE=(TRK,(60,30))
//*
// PRINT OUT THE LOG SUMMARY
//*
//LOGSUM EXEC PGM=CACPRTLG,PARM='SUMMARY=Y',COND=EVEN
//STELIB DD DISP=SHR,DSN=&CAC..V8R2M0.SCACLOAD
//VHSCONF DD DISP=SHR,DSN=&CAC..V8R2M0.REDBOOK(&CONFIG)
//ENGCAT DD DISP=SHR,DSN=&CAC..V8R2M0.SCACMENU
//CTRANS DD DISP=SHR,DSN=&CAC..V8R2M0.SCACSASC
//CACLOG DD DISP=(OLD,PASS),DSN=&LOG
//SYSTERM DD SYSOUT=&SOUT
//SYSPRINT DD SYSOUT=&SOUT
//SYSPRINT DD DUMMY
//*
// PRINT OUT THE LOG
```
Start WIICEP for IMS and verify normal operation

There are very stringent rules for starting up the WIICEP for IMS event publishing environment to ensure normal operations. These are described in more detail in Section 1.4, “WIICEP for IMS processing flow” on page 34.

In this section, we perform a simple cold start procedure for WIICEP for IMS, and start IMS to ensure that the correlation service starts in active mode, and verify normal operation as follows:

1. Start WIICEP for IMS.

   If the correlation service and publication service are configured in the same file (as in our case), you either issue a console command to start the correlation service JCL procedure, or submit a batch job.

   The console command to start the correlation service is as follows:

   ```
   S proclibname
   
   where proclibname is the 1-8 character proclib member name to be started.
   
   When you issue commands from the SDSF product, prefix all operator commands with the forward slash (/) character.
   
   We chose to submit the batch job shown in Example 1-7 to start WIICEP for IMS.
   ```
Verify that the correlation service is running by inspecting the job log, which should contain a number of informational messages that confirm that everything has started correctly as shown in Example 1-8.

Note: The message “CACG113I AGENT 'IMS_IMS810K' SWITCHED TO ACTIVE MODE” is displayed because the correlation service is being cold started after the CCA agent IMS810K was in active communications with the correlation service when the correlation service was shut down.

The message “CACG005I CHANGE CAPTURE SET FOR IMS TABLE DI21PART.STOKSTAT" identifies the table that has been ALTERed for data capture. One common reason for the correlation service to terminate is that it is started without having an ALTER performed. If there is not at least one table that has been ALTERed for data capture, the correlation service terminates.

Note: If the correlation service and publication service are configured in separate files, you can issue a console command to start the correlation service JCL procedure and another console command to start the publication service JCL procedure.

2. Start IMS.

From the SDSF Log, enter the following command to start IMS:

/S IMS810K

Wait for IMS to start, until you see the following WTO message:

STC11485 *672 DFS810A IMS READY
05226/1754571 IMS810K .IMS810K IMSK
Then reply to the IMS Ready message with a normal restart command as follows:

/R 672, NRE

Monitor the messages in the SDSF log for the following messages indicating that the active CCA has been detected and invoked:

CACH001I CHANGE-CAPTURE AGENT 'IMS_IMS810K' INSTALLED FOR SERVICE '(noname)'
CACG109I RECEIVING CHANGES FROM ACTIVE AGENT 'IMS_IMS810K'

3. Make some IMS changes.

Normal update of the IMS databases by applications is allowed to proceed.

4. Check the status of WIICEP for IMS.

There are several sources that can be checked to view WIICEP for IMS status:

- Active status message in the SDSF log indicating that WIICEP for IMS has started receiving changes from the active agent in IMS.

18.04.40 J0811483 CACG109I RECEIVING CHANGES FROM ACTIVE AGENT 'IMS_IMS810K'

- Status of the correlation service in the SDSF log; you can use the following command:

/F CACCSRUN, CMD, XM1/XQM/CSQ1/16, REPORT

where
The WIICEP for IMS job name is CACCSRUN
The correlation service "service name" is XM1/XQM/CSQ1/16

The results of this command may look like Example 1-9, showing one message processed, which was sent onward to the publication server, and confirmed as committed by the publication service.

Note: Often in a test environment, the user will make a single change to the database that does not cause IMS to generate enough IMS log records to cause the IMS log buffer to be flushed and the CCA to be driven. Issuing a simple checkpoint (/CHECKPOINT command) causes IMS to flush the log buffer and drive the exit, and it allows the changes to flow to the publication service.

Example 1-9  Correlation service status

<table>
<thead>
<tr>
<th>Agent</th>
<th>Processed</th>
<th>Sent to Rules</th>
<th>Confirmed</th>
<th>Pending</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMS_IMS810K</td>
<td>000000001</td>
<td>000000001</td>
<td>000000001</td>
<td>000000000</td>
<td>Active</td>
</tr>
</tbody>
</table>

CACG151I END OF REPORT, AGENT TOTAL=1

CACC152I PENDINGQ(0) MSGQ(0) UNCONFIRMED(0)

- Status of the publication service in the SDSF log with the following command:

/F CACCSRUN, CMD, PUB1, REPORT

where
The classic event publisher job name is CACCSRUN
The publication service "service name" is PUB1

The results of this command may look like Example 1-10 which shows one message received, committed and confirmed as processed (confirmed back to the correlation service) by the publication service.
Example 1-10 Publication service status

F CACCrun, CMD, PUB1, REPORT
CAC00200I CMD, PUB1, REPORT
CACJ001I DISTRIBUTION SERVICE ACTIVITY REPORT

<table>
<thead>
<tr>
<th>Message Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change Message(s) Received</td>
<td>1</td>
</tr>
<tr>
<td>Commit Message(s) Received</td>
<td>1</td>
</tr>
<tr>
<td>Commit Message(s) Confirmed</td>
<td>1</td>
</tr>
<tr>
<td>Commit Message(s) Rejected</td>
<td>0</td>
</tr>
</tbody>
</table>

5. Check the contents of the target queue

Given the normal status of the correlation and publication services, XML event messages should appear on the IMSK_RECVQ1 on QMCIF target queue assuming WebSphere MQ channels and listeners are running. The WebSphere MQ Explorer in Figure 1-8 shows the queue depth on IMSK_RECVQ1 as 1.

![WebSphere MQ Explorer](image-url)

Figure 1-8 WebSphere MQ Explorer

The contents of the XML message may be browsed using a freeware tool named RFHUTIL for WebSphere MQ, that allows queues to be read and manipulated. It is a particularly useful testing tool with event publishing since it has an easy option to format message content in XML. The RFHUTIL tool can be downloaded from the following URL:


An example of the use of RFHUTIL is shown in Section A.3, “RFHUTIL tool” on page 237.

1.4 WIICEP for IMS processing flow

This section describes the processing flow in WIICEP for IMS as follows:

- Overview of WIICEP for IMS processing flow
- Detailed description of WIICEP for IMS processing flow
1.4.1 Overview of WIICEP for IMS processing flow

Figure 1-9 provides a high level overview of the processing flow occurring in a WIICEP for IMS environment. Two mutually exclusive processing paths occur in the WIICEP event publishing environment, as follows:

- **Active CCA processing or normal operations:**
  This kind of processing occurs during normal operations when no abnormal conditions have occurred, and changes are intercepted from the IMS log buffers by the active CCA.

  Here is a high level overview of this processing:

  a. Applications updating IMS databases cause log records to be written to the IMS log.
  b. Databases that have change-capture activated have additional “type 99” log records written, and these and other log records such as syncpoint records, are intercepted in the log buffer by the active CCA and sent to the correlation service.
  c. The correlation service then filters the log records (using the metadata catalog) corresponding to those tables that have change-capture activated, and sends only committed records on to the publication service in SQLDAs format. The SQLDAs store before images, after images, or both of the database that was changed.

  d. The publication service then matches publication requests (as defined in the configuration file) with the incoming committed changes, generates XML messages and writes them to a WebSphere MQ queue. It then communicates confirmation or rejection messages to the correlation service when it completes or fails to complete actions in response to the SQLDAs that the correlation service sent.

  Any event that inhibits normal operations causes an error condition and the active CCA to go into recovery mode as described in Section 1.6.2, “Causes of recovery mode” on page 51.

- **Recovery CCA processing or operations:**
  This kind of processing occurs during recovery mode triggered by an error condition, and changes are intercepted from the IMS log file (and not the log buffers) by the recovery CCA. The causes of recovery mode, detection of recovery mode state, and recovering from recovery mode are discussed in detail in Section 1.6, “WIICEP for IMS recovery mode” on page 44.

**Note:** In PTF UK09686 released on December 5th 2005, the correlation service has been changed to send table metadata to the publication service when the publication service connects to the correlation service, and change messages are sent in a different format.
1.4.2 Detailed description of WIICEP for IMS processing flow

This section provides a more detailed description of the processing flow as shown in Figure 1-10.

![Diagram of IMS processing flow](image)

**Figure 1-9  Processing flow overview**

**Figure 1-10  WIICEP for IMS components in detail**
Figure 1-10 on page 36 provides a more detailed look at the various components involved in the processing flow during normal operations, as follows:

- IMS Logger exit DFSFLGX0.
- Optional (but recommended) restart file associated with IMS region.
- CSA that has control information that is shared and used as a bidirectional communication mechanism between the active CCA and the correlation service.

The IMS active change-capture agent accesses the CSA each time the agent is called to identify:

- Whether a correlation service is active
- The name of the XM queue to use
- The current status of the IMS active change-capture agent

The IMS active change-capture agent can also update common storage to report that the agent is in recovery mode.

- Raw data changes dataspace queue that is first-in first-out (FIFO) that is shared by the CCAs and the correlation service.
- Two stores associated with the correlation server as follows:
  - Message store (data plus index) that contains uncommitted raw changes
  - Pending commit store (data plus index) contains information about the committed units-of-recovery (UOR) that have been received from the CCAs and need to be sent to the publication service for processing.

Both these stores are implemented as B-tree data sets and are configured using the LD TEMP SPACE parameter. Both the message store and the pending commit store are temporary data sets. Any information contained in these stores is automatically deleted during initialization of the correlation service.

For performance reasons, IBM recommends that hiperspace is used for these stores as opposed to temporary DASD.

If you use hiperspace, begin with a fairly small initial allocation (perhaps 16 megabytes) and allow the hiperspace to grow slowly up to the maximum configured size (2 gigabytes). Use a small initial allocation because the LD TEMP SPACE parameter is used for both the message store and the pending commit store.

Ideally, there will be a single, or small number of units-of-recovery (UORs) that are waiting to be sent to the publication service for processing in a normal operational environment. Determining the maximum size of the message store is more difficult and depends on the number and size of the UORs that are being generated by your client applications and captured by the CCAs.

While a UOR is in-flight, the changes generated are stored in the message store. Therefore, you must allocate enough space to hold all of these changes. The message store must be configured to be large enough to hold all of the changes for any committed UORs that are being sent, or are waiting to be sent, to the publication service for committed UORs, while new changes are arriving from the CCAs.
Setting a moderate initial allocation size and moderate secondary allocation size, and specifying a large maximum size is recommended since it allows the system to dynamically grow based on load. This approach is viable when your site has sufficient auxiliary storage available to handle the peak load.

- GDG restart data set associated with the correlation server.
- Metadata catalog associated with the correlation server.
- Configuration file associated with the correlation server and publication server.
- WebSphere MQ used exclusively by the publication service.
- Independent restart (WebSphere MQ) queue used by the publication server for restart purposes.

A detailed look at the processing flow under normal operations follows — assuming that the correlation service, publications service and IMS have been started in the correct sequence to ensure active mode operation (and not recovery mode) and WebSphere MQ is up and running as well.

1. Applications update IMS data which causes change capture “type 99” log records to be written for the appropriate IMS segments.

2. IMS passes the IMS Logger Exit a pointer to a buffer containing one or more IMS log records after they have been successfully written to an IMS log file.

   The IMS Logger Exit is also called:
   - During control region initialization processing, after the IMS log files are opened.
   - During control region normal termination processing, after the IMS log files are closed.
   - When an IMS control region terminates abnormally.

   **Note:** If the recovery data set (DD name is CACRCV) exists, and the IMS active CCA detects an error (such as the correlation service not running), it records a restart point in the recovery data set. When the IMS active CCA is again able to communicate with a correlation service, the CCA forwards the restart information to the correlation service, letting the correlation service know that a recovery situation exists.

3. When the IMS active CCA is called by IMS after IMS log records are written to the IMS log and when IMS is terminating (normally or abnormally), the IMS active CCA accesses the CSA to verify the active status of the correlation service and the name of the XM dataspace queue, and sends a single XM data gram to the correlation service. Even if no data capture or synchpoint log records are in the buffer passed by IMS, a data gram is still sent to the correlation service — in this case, the data gram just contains information about the starting and ending log record sequence numbers of the log records contained in the log buffer.

   **Note:** No communication is attempted when the IMS CCA is called during IMS initialization processing, because at this time, a restart point cannot be established.

The data gram contains the following information:

- IMS active change-capture agent identity information
- IMS log record suffix information for the first log record and last log record in the buffer.
- All of the following IMS log records as shown in Table 1-6.
XM data grams represent a unidirectional asynchronous method of communication with the correlation service. The IMS active CCA puts a single XM data gram on a XM dataspace memory queue and exits. XM data grams are posted immediately on the XM queue when the IMS active CCA is called with an IMS log buffer. The IMS active CCA is called synchronously by IMS and must return control as soon as possible to reduce the impact the agent has on IMS operations. The correlation service picks messages up off of the XM queue at the appropriate time and processes them.

### Table 1-6  IMS log records used by WIICEP for IMS

<table>
<thead>
<tr>
<th>Record type</th>
<th>Sub types</th>
<th>Description and purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>All</td>
<td>IMS/VS Accounting Record&lt;br&gt;Used to track the state of IMS batch applications and to detect and handle IMS emergency restart situations for a DB/DC or DBCTL subsystem.</td>
</tr>
<tr>
<td>37</td>
<td>All</td>
<td>QBLK synchpoint Log Record&lt;br&gt;Used to identify the start of a UOR for a BMP application using extended checkpoints, or as the start of Phase II synchpoint processing for a DRA application. These kinds of log records are actually used to identify that a UOR has been committed. WIICEP for IMS uses the same sync-point log records that are used by the DPropNR MQ option.</td>
</tr>
<tr>
<td>38</td>
<td>All</td>
<td>SMB Application Abnormal Termination Log Record&lt;br&gt;Used to identify that a UOR has been rolled back.</td>
</tr>
<tr>
<td>41</td>
<td>All</td>
<td>Batch Checkpoint Log Record&lt;br&gt;Used to identify the end of a UOR for a batch application that issues checkpoints.</td>
</tr>
<tr>
<td>59</td>
<td>37, 38</td>
<td>IMS/VS/FP Sync Point Log Record&lt;br&gt;For Fast Path applications used to determine whether a UOR is being committed or rolled back.</td>
</tr>
<tr>
<td>99</td>
<td>All</td>
<td>Data Capture Log Record&lt;br&gt;Used to capture the changes that occurred to a monitored database. Also used to determine whether an IMS batch application terminated abnormally.</td>
</tr>
</tbody>
</table>

**Note:** By default, the size of the XM dataspace queue allocated is 8 megabytes. As discussed earlier, you can change the size of the queue to be any value between 1 and 256-megabytes on the correlation service’s SIE entry. The size of the queue depends on the number of expected active CCAs and the burst volume of changes from each active CCA. A value of 16 is recommended to help ensure queue space during peak periods. Refer to *IBM WebSphere Information Integrator Getting Started with Classic Event Publishing*, GC18-9186-02 for more information about modifying this setting.

4. The correlation service is designed to operate continuously. For IMS batch jobs, the correlation service must be active before the batch jobs are started and must remain active until the IMS batch jobs complete. Ideally, the correlation service should be active before an IMS DB/DC or DBCTL subsystem is started and remain operational until the DB/DC or DBCTL subsystem is terminated.
The correlation service resides on the same machine as the CCA in a different address space, and performs the following functions:

- Executes as a single threaded processing loop that retrieves raw data from the XM dataspace queue, discards changes that are not flagged for capture or have rolled back, generates SQLDAs for committed UORs that are then sent to the publication service, and then waits for confirmation about the
- Preloads tables flagged for change capture from the metadata catalog at startup.
- Writes queue names to CSA at startup.
- Monitors CSA for CCAs placed in recovery mode at regular intervals.
- Tracks each active CCA’s log record sequence numbers, because it is possible for DFSFLGX0 to be called with duplicated, lost, or out-of-sequence buffers. The correlation service automatically discards duplicated buffers, but if it detects a lost or out-of-sequence condition is encountered it forces a recovery situation by putting the active CCA in recovery mode.
- Tracks the state and health of each IMS control region. The IMS control region facilitates this by sending additional log records to the correlation service that are not related to data capture or tracking the progress of the currently in-flight units-of-work. The main log record of interest for tracking control region status is the “type 06” IMS/VS Accounting Record. Using the information provided in the IMS/VS Accounting Record, as well as information provided by IMS to DFSFGLX0, the correlation service detects and handles the following situations:
  - Start of an IMS control region.
  - Normal termination of a batch job.
  - Abnormal termination of a batch job. If a unit-of-work is in-flight, it is automatically rolled back. In this situation, you might have to run the IMS Batch Backout Utility, but from a WebSphere Information Integrator Classic Event Publisher perspective this is not an error condition.
  - Normal termination of a DB/DC or DBCTL subsystem.
  - Abnormal termination of a DB/DC or DBCTL subsystem. Any in-flight units-of-work that have performed updates are retained, awaiting the control regions restart.
  - Emergency restart of a DB/DC or DBCTL subsystem.
- Retrieves the raw data changes from the XM dataspace queue and verifies the captured change data against the preloaded metadata from the metadata catalog. If the table that was affected is mapped in the metadata catalog and is flagged for data capture, the correlation service stores the data in the message store (B-tree) until it receives a COMMIT or ROLLBACK statement.

Each piece of change information that the correlation service stores in the message store is kept until a COMMIT or ROLLBACK action is performed on the unit of work.

- At COMMIT, the correlation service creates an entry in the pending commit store for this UOR, and also creates one (for inserts and deletes) or two (for updates for the before and after image) SQLDAs for each change to a monitored table. These SQLDAs are sent to the publication service.
- If a ROLLBACK is detected for a unit of work, all data currently stored for that unit of work is deleted.

The data that is sent to the publication service from the correlation service has the following elements:

- Action: This value indicates whether the action performed was an INSERT, UPDATE, or DELETE.
• Table name: This value provides the name of the table that was affected by the action. This table name corresponds to an entry in the DB2 Information Integrator Classic Event Publisher metadata catalog.
• Timestamp: This value provides the time that the action occurred.
• SQLDA1, SQLDA2: These hold the before and/or after images of the data records that were modified.

Transactions are “in-doubt” after they are sent from the correlation service to the publication service, and before the publication service sends a confirmation or rejection message back to the correlation service.

When a transaction is “in-doubt” at the time that the correlation service that sent it terminates, the CCAs that connect to that correlation service are halted, and are set to an “in-doubt” state — this is recorded in the CSA as well as the correlation service restart dataset. When the correlation service is restarted and it detects that the system is “in-doubt”, the correlation service puts any active CCAs into recovery mode.

**Note:** In normal production environments, an active CCA is likely to set recovery mode when it tries to communicate with a correlation service that is down, before the correlation service is restarted. Therefore, the “in-doubt” state is most likely a very transient state in most production environments.

The “in-doubt” state is usually resolved automatically, but may need to be resolved manually depending upon circumstances as follows:

• Automatic “in-doubt” handling:
  When a correlation service is restarted and re-initiates its connection with the publication service that caused the “in-doubt” status, the correlation service sends a message to the publication service with the CCA information and the last transaction confirmation that it received from the publication service. The publication service then attempts to clear the “in-doubt” state by sending any pending confirmations or rejections for past transactions.

• Manual in-doubt handling:
  Recovery CCAs (described in Section 1.6.1, “Recovery mode overview” on page 45) have an override mechanism to allow using an “in-doubt” recovery point for recovery purposes. If you do not want to use this mechanism, you must use the correlation service cold-start procedure. You should consider this option only when the former option will not work.

5. The publication service performs the following tasks:
   a. Receives data in SQLDA format about changes to your source tables from the correlation service.
   b. Matches the data against the publications that you configured in the configuration file to see if the tables specified in the SQLDAs have corresponding publications. If a change does not have a corresponding publication entry, it is discarded.
   c. Converts the SQLDA data into messages in an XML format
   d. Puts the XML messages on the local, persistent WebSphere MQ message queues specified for your publications.
   e. Sends confirmation and rejection messages to the correlation service when it completes or fails to complete actions in response to the SQLDAs that the correlation service sent. These messages establish the new recovery point in the event of a correlation service shutdown or change-capture agent failure.
Information about the last UOR to be sent as a message, and confirmation from WebSphere MQ that messages are accepted, is stored in a local, persistent message queue referred to as the “in-doubt resolution queue”. This queue ensures that if the correlation service is shut down, recovery point information is not lost.

The publication service draws memory buffers from the message pool (determined by the MESSAGE POOL SIZE parameter in the configuration file), and constructs messages within these buffers.

When the publication service sends messages of type TRANS, a large transaction can exceed the size of the buffer in which it is being converted to messages. In such cases, it is useful to allow the publication service to segment the transaction. The publication service constructs two or more messages for the transaction and puts messages on the queue in succession when each becomes a certain size. Use the MAX TRANSPORT MESSAGE SIZE parameter in your configuration file to specify in bytes the largest size that a message can be before the publication service writes it to a message queue. For example, consider the following entry in a configuration file:

```
MAX TRANSPORT MESSAGE SIZE = 262144
```

When the publication service constructs a message for a large transaction in the message pool, whenever the publication service finds that the size of the message reaches 256 KB, the publication service writes the current message to the appropriate message queue and starts building another message to contain the subsequent updates of the transaction.

If this next message becomes 256 KB in size before the end of the transaction is reached, the publication service writes this message to the message queue and begins constructing another message. This process continues until the publication service reaches the end of the transaction. Segments are numbered sequentially by the publication service so that the application that receives them can be sure that they are in sequence. The message attribute “isLast” is set to 1 in the final message so that receiving applications can tell when the transaction is finished.

The maximum value of MAX TRANSPORT MESSAGE SIZE is 10 percent of the parameter MESSAGE POOL SIZE. The minimum value is 64 KB, expressed as 65536. The default value is 128 KB, expressed as 131072.


### 1.5 WIICEP for IMS monitoring tools

WIICEP for IMS provides the following tools to monitor the status and manage some of the resources in the event publishing environment:

- Monitor the correlation service
- Monitor the publication service
- Monitor and manage the CSA

These are described here briefly.

#### 1.5.1 Monitor the correlation service

The following command monitors the correlation service:

```
cmd,name of correlation service,report
```
This command displays a report on the activity of the correlation service and all CCAs that send it change-capture data. Example 1-11 shows an example of the results of this command where the correlation service has processed two transactions for CCA VSAVSAMECA, sent two transactions to the publication service, has received confirmation from the publication service of two transactions, has no pending commits (no in-flight UORs), and the state of the CCA VSAVSAMECA is active.

**Example 1-11  Correlation service report**

```
CAC00200I  CMD,XM1/IMSX/IMSX/200,REPORT
CACG150I  CORRELATION SERVICE ACTIVITY REPORT

*************** Transactions ***************
Agent     Processed  Sent to Rules  Confirmed  Pending  State
----------- ---------- -------------- ---------- ------- -----
VSAVSAMECA 0000002  0000002        0000002  0000000 Active

CACG151I  END OF REPORT, AGENT TOTAL=1
CACG152I  PENDINGQ(0) MSGQ(0) UNCONFIRMED(0)
```

### 1.5.2 Monitor the publication service

The following command monitors the publication service:

```
cmd,name of publication service,report
```

This command publishes a report of the number of change messages received from the correlation service, the number of commit messages received from the correlation service, the number of commits that are confirmed received by the correlation service, the number of commit messages rejected by the publication service.

Example 1-12 shows an example of the results of this command.

**Example 1-12  Publication service report**

```
CACJ001I DISTRIBUTION SERVER ACTIVITY REPORT

-------------------------------
Change Message(s) Received = 13
Commit Message(s) Received = 2
Commit Message(s) Confirmed = 2
Commit Message(s) Rejected = 0
```

### 1.5.3 Monitor and manage the CSA

A CSA reporting and maintenance utility is provided that can be invoked using the SCACSAMP member CACE2RPT. This utility allows you to view and release the contents of the CSA storage that is used for communications between CCAs and the correlation service. The CSA reporting and maintenance utility is controlled with command-line parameters.

The following parameters are supported:

- **REPORT / NOREPORT**

  Identifies whether you want the CSA reporting and maintenance utility to generate an Event Publisher Correlation Service Report describing the CSA usage and state for either an unnamed server, or the correlation service identified by the SERVER command line parameter.

  Example 1-13 shows an example of the output produced for an active unnamed server.
Example 1-13 CSA reporting and maintenance utility report

+---------- Event Publisher Correlation Service Report ----------+ 
| Correlation Service CSA E2CSGBLA, Lth=2588, Lock=x0000      |
| Service Maximum Count=1                                     |
| Allocated Service Blocks=1                                  |
| Agents in Recovery Mode=0                                   |
|                                                              |
| Server AREA XADECA2_WCA047CS, Lth=1024, Free=536 (Active)   |
| Catalog Name 'WCA047.V8R2M00.CATALOG,TCP/9.30.136.99/7056' |
| Connector Filters( VSAM ), ECB(x80000000)                    |
+----------------------------------------------------------------

- RELEASE | NORELASE

Identifies whether the CSA being used by a named or unnamed server should be released (freed) or retained.

Under normal circumstances, you do not want to release the CSA storage. If you encounter error conditions where a correlation service may not start and reports problems related to CSA, you can run the CSA reporting and maintenance utility and specify the RELEASE option to free the currently allocated CSA storage. When you release CSA storage, the correlation service that owns the storage cannot be executing.

- SERVER=Name

Identifies the name of the correlation service to be reported on, or which correlation service owns CSA storage that is being released. If a server is not specified, the CSA reporting and maintenance utility uses the CSA storage associated with the unnamed server.

Example 1-14 shows a sample job to release CSA storage.

Example 1-14 Release CSA job

//CACE2RPT JOB (POK,999),MSGLEVEL=(1,1),MSGCLASS=H, 
//     CLASS=A,NOTIFY=&SYSUID 
//CACE2RPT PROC CAC='CAC'        INSTALLED HIGH LEVEL QUALIFIER 
//* 
//CACE2RPT EXEC PGM=CACECA2U, 
//    PARM='REPORT,RELEASE' TO GET REPORT AND RELEASE CSA 
//*   PARM='REPORT' TO GET A REPORT ONLY 
//STPLIB DD DISP=SHR,DSN=&CAC..V8R2M00.SCACLEAD 
//CTRANS DD DISP=SHR,DSN=&CAC..V8R2M00.SCACSAV 
//SYSPRINT DD SYSOUT=* 
//SYSTERM DD SYSOUT=* 
//    PEND 
//RUNE2RPT EXEC CACE2RPT

1.6 WIICEP for IMS recovery mode

Figure 1-9 on page 36 shows normal event publishing processing involving the IMS active CCA, which retrieves change-capture data directly from the log buffers and publishes it, and recovery event publishing processing involving the IMS recovery CCA, which retrieves change-capture data from the log file and publishes it. These two processing modes are mutually exclusive. The IMS active CCA is implemented as an IMS Logger exit, while the IMS recovery CCA is a batch job.
The IMS active CCA is in one of two modes of operation:

- **Active mode:**
  
  This is the normal mode of operation indicating that the correlation service is active and not reporting any errors, and the IMS active CCA is successfully communicating with the correlation service via XM and CSA.

- **Recovery mode:**
  
  This mode indicates that communications were never successfully established with a correlation service, or the correlation service reported some form of error, or the IMS active CCA detected an error condition.

After an IMS active CCA is in recovery mode, you must run the IMS recovery CCA to find any changes that were made between the time the agent went into recovery mode until one of three points in time:

- The time that the IMS control region is terminated.
- You quiesce the databases being monitored and recover all changes up to the quiesce point.
- You have a window of time where the monitored databases are not actively being updated and you recover all changes up to that point.

An active CCA going into recovery mode requires very special handling to ensure that no ongoing changes to the database are lost. In this section, we provide an overview of recovery mode operations, discuss the various causes of recovery mode, describe the process for getting an active CCA out of recovery mode into active mode using the recovery CCA, discuss measures to minimize the occurrence of recovery mode, and show how to detect the occurrence of recovery mode.

Here is a summary of the main topics covered in the following sections:

- Recovery mode overview
- Causes of recovery mode
- Getting out of recovery mode
- Conditions when recovery is not possible
- Methods to determine if a CCA has updated a monitored database
- Best practices to minimize the occurrence of recovery mode
- Detection of recovery mode

### 1.6.1 Recovery mode overview

Whenever live data capture (changes collected from the log buffer) is prevented, the active CCA is moved into recovery mode.

Recovery mode involves extracting changes from the log file instead, and publishing XML event messages — these event messages have a higher data latency than those generated during active mode operation. However, recovery mode ensures that no changes are lost. The recovery CCA performs recovery mode operation. Recovery is performed differently on the various change-capture agents.

Some of the causes of recovery mode include starting a database without a correlation service, shutting down the correlation service without first stopping the database, rejection of a message by the publication service, and rejection of a message by a WebSphere MQ message queue. Section 1.6.2, “Causes of recovery mode” on page 51 provides a comprehensive list of all the causes of recovery mode, while Section 1.6.6, “Best practices to minimize the occurrence of recovery mode” on page 69 discusses best practices to minimize the occurrence of recovery mode.
Attention: Recovery from these failures may not be possible if they are driven by change data that will produce the same error if recovery is attempted. For example, if the data captured and forwarded to the publication service caused the publication service to reject the message, then the publication service will reject that message every time that the message is resent.

The correlation service is responsible for detecting failure and returning messages stating that an active CCA entered recovery mode.

The recovery CCA runs in five different modes as shown in Table 1-7. Each of these modes may be used in restoring active mode for a CCA as described in Section 1.6.3, “Getting out of recovery mode” on page 54.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>Check agent</td>
<td>Check the recovery data set to see if the identified agent is in recovery mode. If so, identify an initial recovery point. The correlation service is also accessed to see if the agent name is currently in recovery mode. If so a more exact initial recovery starting point is also identified.</td>
</tr>
<tr>
<td>S</td>
<td>Set agent in recovery mode</td>
<td>Given the name of a recovery data set and a IMS CCA name, place that agent in recovery mode. The IMS CCA name must match the name of the agent recorded in the recovery data set.</td>
</tr>
<tr>
<td>L</td>
<td>Log file recovery</td>
<td>Given the name of an IMS log file, an IMS CCA name and the region type attempt to move the recovery point forward in time.</td>
</tr>
<tr>
<td>I</td>
<td>Incremental log file recovery</td>
<td>Perform Log File Recovery processing and halt the Log File Recovery process when the last IMS log file has been processed for this change-capture agent.</td>
</tr>
<tr>
<td>A</td>
<td>Place agent in active mode</td>
<td>Remove the IMS CCA name from recovery mode and place it back into active mode.</td>
</tr>
</tbody>
</table>
Figure 1-11 provides an overview of the components that the recovery CCA interacts with.

![Recovery CCA overview](image)

When the recovery CCA is started and log file recovery is specified, it performs the following actions:

1. Reads one or more control cards from SYSIN that identify the action to be performed. When multiple control cards are provided, all of them must have the same mode. The control cards may be supplied instream, or as a text input file.

   The format of a control card is shown in Table 1-8, while a sample is shown in Example 1-15.

   Example 1-16 shows a sample recovery CCA job with instream control cards specifying the check agent function for the IMS810K online region and its associated CAC.RECOVDS.IMS810K recovery dataset, and the CACBACHG batch job and its associated CAC.RECOVDS.CACBACHG recovery dataset, with its corresponding output in Example 1-17.

   Example 1-18 shows a sample recovery CCA job with instream control cards specifying the log file recovery function for the IMS810K online region and the CACBACHG batch region. The appropriate log files (IMS810K.OLP03 and IMS810K.OLP04 for the online region, and NALUR.LOG for the batch region) are specified, with its corresponding output in Example 1-19.


<table>
<thead>
<tr>
<th>Table 1-8 Recovery CCA control card format</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parameter name</strong></td>
</tr>
<tr>
<td>Mode</td>
</tr>
<tr>
<td>IMS CCA name</td>
</tr>
</tbody>
</table>
Example 1-15  Sample recovery CCA control card

C IMS810K  4 CAC.RECOVDS.IMS810K

Example 1-16  Sample recovery CCA job with instream control cards with check agent

//CACRAREC JOB (3WCA000),'CHRIS',MSGCLASS=A,
// REGION=OM,CLASS=A,NOTIFY=$SYSUID
//IEFPROC EXEC PGM=CACEC1IR,PARM='THROTTLE=1',
// TIME=1440,REGION=32M
//STEPLIB DD DISP=SHR,DSN=CAC.V8R2M0.SCACLOAD
//SYSOUT DD SYSOUT=*  //SYSPRINT DD SYSOUT=*  //CTRANS DD DISP=SHR,DSN=CAC.V8R2M0.SCACSASC
//SYSIN DD *
C IMS810K  4 CAC.RECOVDS.IMS810K
C CACBACHG 4 CAC.RECOVDS.CACBACHG

Example 1-17  Recovery CCA check agent output

+CACHO61I RECOVERY MODE: CHECK AGENT STATUS
+CACHO48I RECOVERY DATASET CAC.RECOVDS.IMS810K OPENED
+CACHO49I AGENT NAME IS 'IMS810K'
+CACHO39I DB2 RESTART TIME 20050906 131128.23
+CACHO40I IMS RESTART TIME 05.249 13:11:28.2
+CACHO37I RESTART SYSTEM CLOCK BD92D7C978646026
+CACHO38I RESTART LOG SEQ, # 00000000-000027FA
+CACHO30I AGENT 'IMS810K' IS NOT IN RECOVERY MODE
+CACHO67I LOG TRACKING FILE DOES NOT EXIST
+CACHO48I RECOVERY DATASET CAC.RECOVDS.CACBACHG OPENED
+CACHO49I AGENT NAME IS 'CACBACHG'
+CACHO39I DB2 RESTART TIME 20050906 131817.98
+CACHO40I IMS RESTART TIME 05.249 13:18:17.9
+CACHO37I RESTART SYSTEM CLOCK BD92DE5EB03C5D64
+CACHO38I RESTART LOG SEQ, # 00000000-00000001
+CACHO30I AGENT 'CACBACHG' IS NOT IN RECOVERY MODE
+CACHO67I LOG TRACKING FILE DOES NOT EXIST
2. The correlation service maintains restart information in the recovery GDG data set and in the CSA. The IMS recovery CCA obtains the information from CSA. In the case where the correlation service does not know that the CCA is in recovery mode and there is restart information in the CCA's recovery dataset, you must use the “set” mode (set agent in recovery mode) of the recovery CCA to notify the correlation service that the agent is in recovery mode and transfer the restart information into the GDG/CSA. The names of the various recovery datasets must be identified in the recovery CCA control cards.

Note: The IMS log file that contains the restart point and all log files created thereafter that contain changes to the monitored databases, up until the point that IMS is shut down, or a quiesce point is reached, must be input into the recovery process. If the CCA associated with an IMS control region is in recovery mode and that IMS job or subsystem is run again before the recovery process is completed, then any IMS log files created in subsequent executions must also be input into the recovery process.

A CCA that is in recovery mode has a specific record within an IMS log file where recovery needs to start. Initially, this is the first IMS log record when the active change-capture agent was called for the first IMS Logger Exit write operation.

- For an IMS batch application, the restart point identifies the first IMS log record in the log file.
- For a DC/DC or DBCTL subsystem this is generally the first IMS log record in the currently active online log data set when the subsystem is normal or emergency restarted.

When a UOR is committed by the correlation service, the CCA's restart point is the IMS log record associated with the first change received from the oldest living UOR when the committed UOR started. As UORs are committed, the restart point keeps getting pushed farther into the future. See the discussion below that more fully explains the concept of the oldest living UOR. Although multiple UORs might be in-flight when a CCA goes into recovery, the restart point is recorded for the UOR that is the oldest. Figure 1-12 illustrates how restart points progress forward in time for a DB/DC or DBCTL subsystem that has four UORs being tracked.
UOR 1 has a restart point: when the UOR was started. Because UOR 1 and 2 overlap in time, UOR 2’s restart point is the time UOR 1 started. Likewise, UOR 2 and 3 overlap in time and therefore UOR 3’s restart point is when UOR 2 started. UOR 4 does not overlap so its restart point is when UOR 4 started.

The following information is associated with a restart point:

- DB2 timestamp when the agent went into recovery.
  The DB2 timestamp for a restart point is based on the system clock value for the oldest living UOR. This is either the IMS log record of the first “type 99” log record, if a UOR was in-flight when the system went into recovery, or the system clock value of the first log record recorded by the CCA. The time has millisecond resolution.

- Internal timestamp when the agent went into recovery.
  The internal timestamp consists of a DB2 timestamp with a resolution of a tenth of a second, based on the oldest living UOR system clock value.

- The IMS log record suffix of the first IMS log record recorded by the CCA or the first “type 99” log record received for the UOR that is in recovery.

**Important:** The IMS log suffix information is the real key to a restart point. The IMS log record suffix consists of an 8-byte system clock value that identifies when the IMS log record was created. Also included in the suffix is an 8-byte double word IMS log record sequence number generated by IMS. The combination of the system clock value and the log sequence number identify a unique IMS log record that is used to establish the restart point.

- The IMS log record suffix of the first log record recorded by the change-capture agent or the first type 99 log record for the oldest UOR in existence when the UOR that is in recovery was created.

Figure 1-12  Recovery CCA restart points
3. It starts reading changes from the IMS log file from the restart point and writes the changes out to the correlation service as XM datagrams. However, unlike the active CCA which posts a single XM data gram for a single IMS buffer at a time, the recovery CCA posts an XM data gram when a 32 kilobyte buffer is filled up with IMS log records of interest.

If IMS log files for multiple agents are being recovered, an XM data gram is posted when:
- A 32-kilobyte buffer is filled up with IMS log records of interest.
- A buffer is truncated due to system clock overlaps with another agent.

4. The recovery CCA job runs until it reaches the end of file on all the input logs identified to the utility — in the case of incremental recovery, until it reaches end of file on the set of log files for a particular CCA. You have to keep modifying the recovery CCA control card input identifying new log files until IMS is shut down or you reach a quiesce point. The Recovery CCA job issues WTO messages to inform you of its progress.

When the recovery of data catches up with the changes occurring on the data source, with some databases such as IMS, you need to stop and start the database to move the CCA back into active mode with no changes lost. However, it is often not practical to stop the monitored database to complete the recovery process. For these situations, WIICEP provides methods to exit recovery mode and resume the active capture of changes from the database system. Section 1.6.3, “Getting out of recovery mode” on page 54 describes the recommended approach to get one or more IMS active CCAs out of recovery mode into active mode.

### 1.6.2 Causes of recovery mode

An active CCA goes into recovery mode when the environment can no longer sustain normal operations where changes occurring in the data source can no longer be captured live from the log buffers and sent to the correlation service and publication service for generating events as XML messages and populating a target WebSphere MQ queue.

The following list describes the most frequent causes of recovery mode in an WIICEP for IMS environment.

1. Incorrect startup and shutdown sequence of IMS, correlation service, and publication service:

   The various components (IMS, correlation service, publication service, WebSphere MQ manager, WebSphere MQ queues) of a WIICEP for IMS event publishing environment need to be operating and functional for normal operations. There is a starting dependency as well as a shutdown dependency between these components which if not satisfied will cause the IMS active CCA to go to recovery mode.
Assuming that WebSphere MQ manager, and queues are up and running, the correct startup and shutdown sequence of the correlation service, publication service, and IMS to avoid recovery mode is as follows:

- **Startup sequence:**
  i. Correlation service
  ii. WebSphere MQ
  iii. Publication service
  iv. IMS

If any of these components are started out of turn, then the IMS active CCA will go into recovery mode and issue an appropriate WTO or WTOR message. For example, if IMS is started without the correlation service, a WTOR message as follows is written:

```
CACH002A EVENT PUBLISHER SERVICE ' (noname)' NOT FOUND BY CHANGE-CAPTURE AGENT 'IMS_IMS810K'. REPLY 'R' 'A'
```

- **Shutdown sequence:**

  The correct shutdown sequence is the inverse of the startup sequence as follows:
  i. IMS
  ii. Publication service
  iii. Correlation service

**Note:** The lack of proper WebSphere MQ channel definitions, or them being down, should not cause a recovery situation. If a remote Q setup is being used, as long as the local queue is defined, the publication service should be able to write to it, and eventually the changes should show up at the remote destination.

2. Publishing new tables or modifying the configuration file (CACCSCF member in the SCACCONF data set):

   The addition of new publication tables or modifications to existing publication tables requires the metadata catalog to be updated. Since the correlation service preloads the metadata catalog, for these changes to take effect, the correlation service must be shut down and restarted.

   The same applies to the configuration file (CACCSCF member in the SCACCONF data set) which includes the SIEs, publications and configuration parameters.

   If considerable care is not taken to observe the shutdown and startup sequence for the correlation service, publication service and IMS, then the IMS active CCA can go into recovery mode.

3. IMS Online Change of DBDs:

   As mentioned in “STEP 3c: Modify IMS DBDs to support event publishing” on page 14, data capture log records are generated at the database or segment level and require augmentation of the DBD definitions via the EXIT= keyword parameter on the DBD or individual SEGM statements. The augmented DBD must then have DBDGEN run, followed by the ACBGEN utility to update all PSBs that reference the augmented DBD. The updated DBD and PSB members then need to be put into the production ACB libraries.

   If you make these changes with the IMS Online Change facility, the correlation service (which has a DD card for the DBDLIB and caches the timestamp associated with the DBD at startup) detects a timestamp difference between the cached timestamp of the DBD and the updated DBD, throws an error and marks the active CCA as being in recovery mode.
4. XM queue overrun:

If the amount of storage allocated for the XM queue is too small, the XM data grams sent by the CCA agents will fail causing the correlation service to shut down and mark all the IMS active CCAs associated with this correlation service as being in recovery mode.

In this situation, the correlation service needs to be stopped and the correlation service SIE definition needs to be changed to increase the size of the XM queue. In particular, if an XM queue overrun occurs from the activity of an active CCA, the size of the XM queue needs to be increased because the IMS recovery change-capture agent sends larger XM data grams than the active CCA does and at a faster rate, when the restart point has been reached in the IMS log files. Experience has shown that you can use the THROTTLE parameter to slow down the IMS recovery CCA somewhat. However, in this type of recovery situation, a larger XM queue is required.

One of the primary reasons for an XM queue overrun is if you have a very large UOR generated by a batch job or BMP, and have the INT parameter set to 0. In this case, the correlation service will not check the XM queue until all messages have been sent to the publication service. This is one of the reasons why INT=1 is the recommended setting.

5. Correlation service unavailability:

The unavailability of the correlation service due to any failure or shutdown command will cause the IMS active CCA to go into recovery mode if it is still running and cannot send communications with the correlation service.

6. “Bad” data being captured:

This may or may not be a mapping error, or there is actually “junk” in the database. These kinds of errors are reported by the correlation service when a committed change is converted into its SQL representation.

- In the mapping error case, you may be able to correct the problem by changing the SQL data type of the column — this kind of error is generally only reported on zoned decimal data that has been morphed into DECIMAL data.
- In the “junk” data case, you can drop the column from the mapping or use the selection exit to filter (discard) the changes that contain “bad” data — the selection exit is only passed the before image so you can not correct the data.

This kind of error is also very common when a segment is redefined — in such cases you need to use the selection exit to “switch” based on the record type code (or some other mechanism) in the segment.

7. Rejection of a message by a publication service:

If the data captured and forwarded to the publication service by the correlation service caused the publication service to reject the message, then the IMS active CCA goes into recovery mode. The correlation service continues operation on the other active CCAs. Such a failure may not be recoverable if it is driven by change data that will produce the same error if recovery is attempted since the publication service will reject that message every time that the message is resent.

**Note:** One method that can be used to avoid having the correlation service go into recovery when IMS Online Change of DBDs is performed is to specify a generic version identifier on the VERSION parameter on the DBD statement. If you use a generic version identifier (such as “WEPIMS”), then IMS Online Change of DBDs can be performed without the correlation service going into recovery, assuming that the data capture option settings for the segments being captured are not changed.
8. Rejection of a message by a WebSphere MQ queue:

If WebSphere MQ rejects a publication service call to insert an XML message into a target queue, the publication service sends a rejection message to the correlation service, which puts the relevant active CCA into recovery mode.

Recovery will also occur at the end of the transaction if the WebSphere MQ commit operation reports an error.

9. Lost or out of sequence log buffers:

As mentioned earlier, an XM data gram is sent to the correlation service even when it contains no data capture or synchpoint log records of interest. In such cases, the data gram just contains information about the starting and ending log record sequence numbers of the log records contained in the buffer. The correlation service tracks each active CCA's log record sequence numbers, because it is possible for DFSFLGX0 to be called with duplicated, lost, or out-of-sequence buffers. The correlation service automatically discards duplicated buffers, but when encounters a lost or out-of-sequence condition, the correlation service marks that particular active CCA as being in recovery mode.

10. "Type 99" log record is marked “in error”:

This can occur when cascade delete information is being captured and IMS does not have enough virtual storage to construct the entire IMS logical log record. This represents a non-recoverable situation.

Section 1.6.7, “Detection of recovery mode” on page 69 describes the approach for detecting whether a CCA has gone into recovery mode, while Section 1.6.3, “Getting out of recovery mode” on page 54 describes our recommended approach for getting one or more CCAs out of recovery mode.

1.6.3 Getting out of recovery mode

You should first verify that recovery of one or more CCAs in recovery mode is required. Recovery of an IMS CCA is not required if the IMS applications for the CCAs that failed did not update any monitored databases. Methods to determine whether a CCA updated a monitored database are discussed in Section 1.6.5, “Methods to determine if a CCA has updated a monitored database” on page 68.

In some cases, recovery of CCAs in recovery mode may not be possible as discussed in “Conditions when recovery is not possible” on page 68.

To successfully recover changes to IMS data that WIICEP for IMS has lost, the following infrastructure needs to be in place:

- The names of the IMS active Change Capture Agents (CCAs) that were active at the time of failure, or could become active before active mode, are established.

  The most important part of getting IMS back in synchronization with WIICEP for IMS is the identification of the names of the IMS Change Capture Agents that are (or potentially are) in recovery mode.

- An approximate earliest restart date and time when one or more of the IMS active CCAs failed.

- The names of the IMS log files that are associated with the IMS Active CCAs address space at, and after the point of failure up to the present time, when no IMS Control Regions are operational.
Figure 1-13 describes the main steps in the recovery process of getting all related IMS active CCAs out of recovery mode back into active mode. Each of these steps is discussed in detail in the following sections.

**STEP 1: Identify status of all related IMS CCAs**

To understand the concept of all related IMS CCAs, assume the following scenario.

An organization has a runtime environment with a single IMS control region associated with three full function IMS databases D1, D2, and D3. Only the D1 database has been enabled for change data capture for event publishing. There are four batch jobs, BD1A, BD1B, BD1C and BD1D, which update the D1 database, and these batch jobs execute either when the IMS control region is down, or when the D1 database is deallocated from the IMS control region via the `/DBR D1` command. In this scenario, there are five CCAs — the IMS control region and the four batch jobs.

**Important:** All the IMS CCAs that write captured data changes to the same correlation service are considered to be related IMS CCAs.

The CACE1OPT member in the SCACSAMP library specifies the particular correlation service that a CCA connects to. Any changes made to this member requires it to be reassembled and relinked.

Figure 1-14 shows one possible timeline of execution of some of the related IMS CCAs as follows.
1. The IMS control region has databases D1, D2 and D3 allocated to it from time “t0” through “t1” with applications updating one or more of these databases.

2. At time “t1”, database D1 is deallocated to allow processing by one or more batch jobs, while the IMS control region continues to operate updating databases D2 and D3.

3. Batch jobs BD1A, BD1B and BD1D submitted in sequence between time intervals “t2” and “t3”, “t4” and “t5”, and “t6” and “t7” respectively.

4. At time “t8” after all the batch jobs have completed successfully, database D1 is reallocated to the IMS control region for processing.

In this scenario, even though batch job BD1C is one of the related IMS CCAs, it is not part of this particular timeline’s processing.

**Note:** In another possible timeline, it is possible for some other mix of batch jobs to be executed after database D1 is deallocated from the IMS control region.

Under normal operating conditions, any changes from these various IMS CCAs are published in the sequence in which they occur — all changes from the IMS control region up to “t1’, followed by changes made by batch jobs BD1A, BD1B and BD1D respectively. Maintaining this time sequence of events is critical in most event publishing environments.

**Figure 1-14  Related IMS CCAs scenario**

It is very important to understand why the status of all the related IMS CCAs must be determined in a timeline such as shown in Figure 1-14:

- When a single IMS active CCA goes into recovery mode for any one of the reasons described in Section 1.6.2, “Causes of recovery mode” on page 51, conditions may or may not have been set that force related IMS active CCAs to go into recovery mode. If a failure of the correlation service occurs, then all related IMS CCAs attempting to communicate with that correlation service will also be put into recovery mode. However, if a rejection of a message by the publication service causes an IMS active CCA to go into recovery mode, then related IMS CCAs will most probably not go into recovery mode.

- If the IMS control region IMS810K were to go into recovery mode sometime between time interval “t0” and “t1”, but the IMS active CCA corresponding to BD1A, BD1B and BD1D did not go into recovery mode, then there will be messages published that are out of sequence. This is because changes that may have occurred in the IMS control region after recovery mode is set are no longer published, but subsequent changes occurring due to IMS active CCAs BD1A, BD1B and BD1D continue to get published since they are not in recovery mode. This may be considered an unacceptable occurrence in many application environments.
We recommend that you create a special IMS recovery CCA job called the “IMS active agent status job” that is used to identify the status of each of the related IMS active CCAs that exist at your site. The control file for the IMS active agent status job is a more-or-less a static file that you create once and only need to update when you add a new related IMS active CCA at your site or remove an existing related IMS active CCA — this control file is called the “IMS Status control file” and contains a control card for each of your related IMS active CCAs. The order of these control cards does not matter. Example 1-20 shows the “IMS Status control file” for our scenario, where CAC.RECOVDS.xxx is the name of the recovery data set associated with the particular CCA.

A recovery data set is a sequential 80-byte fixed record length file. Only one record is ever written to this file, so it does not have to be blocked and you can allocate the minimum of one track. The recovery data set assists in situations when recovery is required but the correlation service is not aware of it — this occurs when you run IMS without running a correlation service, which requires recovery to be run, but the correlation service is not aware of this when it is started up. To handle these situations, you can update your IMS control region JCL and include a reference to a recovery data set — the default recovery data set DD name is CACRCV. When a recovery data set exists, and the IMS active CCA detects an error such as the correlation service not running, it records a restart point in the recovery data set.

When an IMS CCA is set to active mode as described in “STEP 4: Place all related IMS CCAs in active mode” on page 66, the restart information recorded in the recovery data set is deleted.

Tip: Since recovery data sets are required when running the IMS recovery CCA as described in “STEP 3: Initiate recovery CCA with one or more log files” on page 58, we strongly recommend that you update your IMS JCL to include a reference to a recovery data set for each IMS active CCA running at your site even though its optional.

If you do not add a recovery data set to one of your IMS jobs and run the job without a correlation service being active, you need to manually create a recovery data set whose name should be included in the control cards of the recovery CCA to perform recovery.

**Example 1-20  IMS Status control file**

<table>
<thead>
<tr>
<th>C</th>
<th></th>
<th>CAC.RECOVDS.xxx</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMS810K</td>
<td>4</td>
<td>CAC.RECOVDS.IMS810K</td>
</tr>
<tr>
<td>BD1A</td>
<td>4</td>
<td>CAC.RECOVDS.CACBD1A</td>
</tr>
<tr>
<td>BD1B</td>
<td>4</td>
<td>CAC.RECOVDS.CACBD1B</td>
</tr>
<tr>
<td>BD1C</td>
<td>4</td>
<td>CAC.RECOVDS.CACBD1C</td>
</tr>
<tr>
<td>BD1D</td>
<td>4</td>
<td>CAC.RECOVDS.CACBD1D</td>
</tr>
</tbody>
</table>

When you execute the “IMS active agent status job”, a series of WTO messages are generated for each CCA identified in the “IMS Status control file” similar to that shown in Example 1-17 on page 48. These WTO messages identify whether the agent is in recovery mode, and if so, an initial recovery restart point. The restart point identifies:

- The exact restart point when the active CCA failed
- Whether the active CCA is known by the correlation service.
- The exact position in the IMS log files where the restart operation needs to be initiated.
As mentioned earlier, identification of the CCAs that are in recovery mode and identification of recovery restart points is crucial to successful recovery (without loss of changes) of the WIICEP for IMS environment.

**Attention:** The correlation service must be running in order to check agent status.

**STEP 2: Place all related IMS CCAs in recovery mode**

Sometimes, the correlation service is not aware of a particular IMS active CCA being in recovery mode even though the CCA’s recovery data set has that information — this happens when the correlation service is shut down while IMS is still running. Recovery mode has to be set for such IMS CCAs using the set agent in recovery mode function using control cards shown in Example 1-21. The output of this job is shown in Example 1-22.

**Attention:** The correlation service must be made aware of the recovery mode state of an IMS CCA in order for the recovery CCA to function — any attempt to recovery an IMS CCA that the correlation service thinks is **not** in recovery mode will fail with a message indicating that recovery is not required.

**Example 1-21  Set agent in recovery mode control cards**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>BD1A</td>
</tr>
<tr>
<td>S</td>
<td>BD1B</td>
</tr>
<tr>
<td>S</td>
<td>BD1C</td>
</tr>
<tr>
<td>S</td>
<td>BD1D</td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>CAC.RECOVDS.CACBD1A</td>
</tr>
<tr>
<td>S</td>
<td>CAC.RECOVDS.CACBD1B</td>
</tr>
<tr>
<td>S</td>
<td>CAC.RECOVDS.CACBD1C</td>
</tr>
<tr>
<td>S</td>
<td>CAC.RECOVDS.CACBD1D</td>
</tr>
</tbody>
</table>

**Example 1-22  Set agent in recovery mode partial output**

```
+CACH048I RECOVERY DATASET CAC.RECOVDS.CACBD1A
+CACH052I AGENT 'CACBD1A' IS NOW IN RECOVERY MODE
+CACH039I DB2 RESTART TIME 20050906 13181798
+CACH040I IMS RESTART TIME 05.249 13:18:17.9
+CACH037I RESTART SYSTEM CLOCK BD92DE5EBD35C5D64
+CACH038I RESTART LOG SEQ. # 00000000-00000001
+CACH067I LOG TRACKING FILE DOES NOT EXIST
+................
+CACH062I RECOVERY PROCESSING COMPLETED SUCCESSFULLY
```

**STEP 3: Initiate recovery CCA with one or more log files**

After you have ensured that all the related IMS active CCAs are in recovery mode, you need to identify the IMS log files that need to be fed into the recovery process for the CCAs that need to be recovered. In our scenario, this means IMS CCAs IMS810K, BD1A, BD1B and BD1D — BD1C does **not** need to be recovered because it did not run in the particular time interval “t0” through “t8”.

The recovery process involves the following two steps:

- Identify all the log files required for each IMS CCA
- Initiate the recovery of all related IMS CCAs in recovery mode

**Identify all the log files required for each IMS CCA**

There are three ways of determining what log files must be supplied to the recovery CCA to bring the WIICEP for IMS back into sync.
1. Manually track the IMS jobs and IMS log files that were created, and supply this information to the IMS recovery CCA.

2. Run a DBRC LIST.LOG ALL (or variant) report if the IMS jobs are registered with DBRC, and review the DBRC output to identify the IMS log files associated with the IMS control region for the agents in recovery.

**Tip:** We strongly recommend that you register all IMS subsystems being monitored with DBRC, and that you create a recovery data set for each IMS CCA.

As described in “STEP 1: Identify status of all related IMS CCAs” on page 55, for any IMS active CCA in recovery mode, the check agent status job reports the DB2 and IMS DBRC date and time stamp of when the IMS CCA failure occurred.

Based on the state of the IMS active CCA when failure with the correlation service occurred, different restart information is available as follows.

- If the IMS subsystem was activated without a correlation service, then log recovery starts at the beginning of the first log file created by that IMS subsystem when it was activated.

- For IMS active CCAs that failed while connected and active with a correlation service, the starting log sequence number and store clock time of the oldest existing in-flight unit-of-work.

In either case you can use DBRC to determine the names of the IMS log files associated with the IMS subsystem that needs to be recovered. You can obtain information about the IMS log files associated with an IMS subsystem using the DBRC LIST.LOG ALL command — sample output is shown in Example 1-23.

**Example 1-23   Sample DBRC LIST.LOG ALL command output**

```
0 LIST.LOG ALL
105.249 13:53:58.4 LISTING OF RECON
0-------------------------------------------------------------------------------
PRILOG                                       RECORD SIZE=     304
START = 05.207 09:15:08.0    SSID=IMSK     VERSION=8.1
STOP  = 05.207 09:21:53.1    #DSN=1
GSGNAME=**NULL**
FIRST RECORD ID= 0000000000000001   PRILOG TOKEN= 0
EARLIEST CHECKPOINT = 05.207 09:15:11.4
DSN=IMS810K.RLDSP.IMSK.D05207.T0915080.V00    UNIT=3390
START = 05.207 09:15:08.0    FIRST DS LSN= 0000000000000001
STOP  = 05.207 09:21:53.1    LAST  DS LSN= 0000000000000096
FILE SEQ=0001    #VOLUMES=0001
VOLSER=TOTMIN STOPTIME = 05.207 09:21:53.1
CKPTCT=2    CHKPT ID = 05.207 09:21:52.9
LOCK SEQUENCE#= 000000000000

LOGALL
```
START = 05.207 09:15:08.0  
EARLIEST ALLOC TIME = 00.000 00:00:00.0
DBDS ALLOC=0

105.249 13:53:58.4  LISTING OF RECON  PAGE 0003

0-------------------------------------------------------------------------------
PRIISLD
RECORD SIZE= 304
START = 05.207 09:15:08.0  *  SSID=IMSK  VERSION=8.1
STOP = 05.207 09:21:53.1  #DSN=1
GSGNAME=**NULL**
FIRST RECORD ID= 0000000000000001  PRILOG TOKEN= 0

105.249 13:53:58.4  LISTING OF RECON  PAGE 0004

0-------------------------------------------------------------------------------
PRIISLD
RECORD SIZE= 304
START = 05.207 09:23:46.5  *  SSID=IMSK  VERSION=8.1
STOP = 05.207 18:11:06.5  #DSN=1
GSGNAME=**NULL**
FIRST RECORD ID= 0000000000000001  PRILOG TOKEN= 0

105.249 13:53:58.4  LISTING OF RECON  PAGE 0005

0-------------------------------------------------------------------------------
PRIISLD
RECORD SIZE= 304
START = 05.207 09:23:46.5  *  SSID=IMSK  VERSION=8.1
STOP = 05.207 18:11:06.5  #DSN=1
GSGNAME=**NULL**
FIRST RECORD ID= 0000000000000001  PRILOG TOKEN= 0

.................
Log files have to be determined for batch jobs as well as DB/DC or DBCTL subsystems, as follows:

- **Batch jobs:**
  
  If you have an IMS batch job in recovery that is being tracked by DBRC, you can issue the `LIST.LOG ALL SSID(Batch-Job-Name)` command to obtain a list of IMS log files associated with the batch job. The DBRC output lists all of the IMS log files for the IMS batch job that DBRC knows about.

  When the IMS recovery CCA reports in a check agent\(^3\) job, an exact restart point for the CCA in recovery mode (non-zero system store clock and non-zero log sequence numbers), then the IMS log files that need to be input into the recovery process are selected by identifying the IMS log file that was active at time of failure. This log file has a creation date that is on or before the IMS restart time displayed, and has a starting log record sequence number that is less than or equal to the restart log record sequence number.

  After you identify the initial IMS log file you must input all other IMS log files created after the initial IMS log file into the recovery process.

- **DB/DC or DBCTL systems**

  Identifying IMS log files for a DB/DC or DBCTL subsystem in recovery mode is more difficult than with batch jobs. Generally, you need to supply one or more of the archived online log data sets into the recovery process. In DBRC terms, these are identified as either PRISLD (primary system log) data sets or SECSLD (secondary system log) data sets.

  It is also possible to use the non-archived online log data sets as input under the following conditions:
  
  - The IMS DB/DC or DBCTL subsystem is shut down.
  - The online logs contain the initial restart recovery point reported by the IMS recovery CCA.

  Assuming that you are going to use archived logs, then the `LIST.LOG ALL DBRC` command can be issued.

  If you have multiple DB/DC or DBCTL subsystems, you can further restrict the DBRC output by specifying the SSID parameter identifying the 4-character subsystem ID to be selected.

  When dual logging is in effect, either the primary or secondary system logs can be used as input. After you find the correct section of the report, you will see a list of log files created in ascending creation date or time sequence as shown in Example 4-17 on page 206.

  Use the same rules as discussed earlier with batch jobs to determine which IMS log files need to be input into the recovery process.

3. Have WIICEP for IMS track the IMS log files associated with an agent. When this is done the IMS recovery CCA automatically identifies the IMS log files you need when you check an agents status. This is known as log file tracking.

   Even using log file tracking, it is possible to miss a log file if the control file is not updated properly. Log file tracking allows WIICEP for IMS to track the IMS log files associated with an IMS active CCA. When log file tracking is implemented, all IMS log files associated with a CCA are tracked regardless of the state of the CCA.

\(^3\) The check agent mode is one of the functions of the recovery CCA.
Log file tracking is accomplished by creating an 80 byte sequential log file tracking data set. This data set can contain multiple records. You can control how many log entries are maintained for each agent being tracked. When accessing IMS log file tracking information, you do not explicitly tell the IMS recovery CCA the name of the IMS log file tracking data set that is associated with the CCA. The name of the file that the IMS recovery CCA attempts to open is: Recovery-File-Name.LOGS — therefore, the IMS log file tracking data set must conform to the afore naming convention.

The IMS log tracking utility manages the content of an IMS log file tracking data set. The IMS log file tracking data set contains a record for each primary and secondary IMS log file created by IMS for a given CCA.

The content of each record contains the following information:

- Type of IMS log file — Primary Log File or Secondary Log File
- First Log Record Suffix which consists of an 8-byte system clock value and a 8-byte double word IMS log sequence number — which is the restart point.
- Log File Name is the fully-qualified data set name of the IMS log file.

Given the information contained in an IMS log file tracking data set, the IMS recovery CCA can identify the IMS log files that need to be recovered for an IMS active CCA that is in recovery mode. Based on a restart point, the IMS recovery CCA can determine whether the IMS log file was created before or after the agent went into recovery mode. Any file created after the restart point must be recovered. Also, the last log file with a system clock value that starts before the restart point is necessary. It is assumed that this IMS log file contains the IMS log record that the restart point is referring to.

When reporting about IMS log files required in the recovery process, the IMS recovery change-capture agent identifies primary IMS system log files, or archived IMS system log files in preference to secondary IMS system or archived log files. This behavior is modified when there is no primary IMS system or archived log file information and only secondary IMS log file information is available.

**Note:** In extreme recovery situations, you will need to update the content of an IMS log file tracking data set either by running the IMS log tracking utility manually or by physically modifying the data set’s content. For best results, make sure IMS log files are listed in the sequence in which they were created. Also make sure that secondary IMS log files are listed immediately after the corresponding primary log file entry. If you do not follow these guidelines, the IMS recovery CCA will terminate and report errors when duplicate IMS log files are supplied, based on the output generated by the IMS recovery change-capture agent when requesting agent status information.

Implementing IMS log file tracking differs slightly between DB/DC, DBCTL, and IMS batch environments, but in all of these environments IMS log files are defined to the tracking system using the IMS log tracking utility. The goal of tracking IMS log files for a particular IMS active CCA is to retain information about all IMS log files that physically exist:

- The IMS log tracking utility is a job-step that you must add to an IMS DB/DC or DBCTL subsystem’s log archive JCL to register the archived IMS log file in the IMS log file tracking data set.
- For IMS batch jobs, additional job-steps must be added after each job-step that updates IMS data that is monitored by an IMS active CCA.

Member name CACIMSLT in SCACSAMP contains sample JCL used to execute the IMS log tracking utility. The IMS log tracking utility is command-line driven, and uses fixed DD names to identify the primary (referenced by the CACLOG1 DD statement) and secondary (referenced by the CACLOG2 DD statement) log files to be registered and the name of the IMS log file tracking data set (referenced by the CACTRACK DD statement) to be updated.
Initiate the recovery of all related IMS CCAs in recovery mode

After you have identified all the IMS log files required, you must input these files into the recovery process for one or more IMS active CCAs that are in recovery mode.

Attention: It is not necessary to provide all the required IMS log files in a single execution of the recovery CCA. Multiple recovery CCA executions may be performed with additional IMS log files. In fact, we recommend that a single recovery CCA provide only two or three files per CCA in order to minimize resource consumption requirements such as the XM data space queue where may result in XM queue overruns.

It is possible to get XM queue overruns if too much data is fed in at one time, if the XM queue is too small and the INT value is not set to 1. If you only have a single online CCA in recovery, then feeding the recovery CCA is chunks is okay. If you have multiple CCAs in recovery you need to provide all of the log files that span the times when these CCAs ran. In a data sharing environment, you need to feed in all of the logs.

If you have multiple IMS CCAs in recovery mode, it is important that you supply IMS log files for all of these CCAs each time log recovery is performed. Failure to do so can cause changes to be propagated to the publication service in the wrong chronological order. Additionally, if you are supplying multiple IMS log files for a CCA that is in recovery, you need to supply all IMS log files for the time period being recovered.

When the IMS recovery CCA is recovering IMS log files, the recovery process is fairly sophisticated. The IMS recovery CCA accepts IMS log files that were created before the restart point and automatically discards log records created before the restart point. If multiple IMS log files are supplied for an agent, these files are automatically sorted in system clock time sequence so that they are processed in the correct sequence. During the sorting process, the IMS recovery CCA automatically checks for specification of an IMS log file more than once in the control file and for IMS log files that have overlapping system times which indicates that an IMS log file for a different IMS active CCA was supplied. Additional checks are performed for IMS batch applications to insure that IMS log files are not supplied from a different job. However, the IMS recovery CCA cannot detect when an IMS log file that needs to be supplied for a batch job was not. For online regions, the IMS recovery CCA can detect when an online log is missing, provided the IMS subsystem has not been cold started.

Attention: If you start an IMS recovery CCA without all of the necessary log files, the correlation service is likely to detect this condition, but by that time, the restart point might be pushed into the future and the content of the missing log file will no longer be recoverable.

Use the following guidelines for identifying IMS log files to be recovered:

- IMS log files can be supplied that were created before the current restart point. Recovery CCA reads these files and discards the log records that were created before the current restart point.

Note: Many organizations may be unwilling to add to an IMS DB/DC or DBCTL subsystem's log archive JCL, or a job step for batch jobs for log file tracking. In such cases one of the first two approaches may be more appropriate in such situations.

For further details on determining the IMS log files to be input to recovery, refer to IBM DB2 Information Integrator Operations Guide for Classic Event Publishing Version 8.2, SC18-9157-02.

Attention: It is not necessary to provide all the required IMS log files in a single execution of the recovery CCA. Multiple recovery CCA executions may be performed with additional IMS log files. In fact, we recommend that a single recovery CCA provide only two or three files per CCA in order to minimize resource consumption requirements such as the XM data space queue where may result in XM queue overruns.

It is possible to get XM queue overruns if too much data is fed in at one time, if the XM queue is too small and the INT value is not set to 1. If you only have a single online CCA in recovery, then feeding the recovery CCA is chunks is okay. If you have multiple CCAs in recovery you need to provide all of the log files that span the times when these CCAs ran. In a data sharing environment, you need to feed in all of the logs.

If you have multiple IMS CCAs in recovery mode, it is important that you supply IMS log files for all of these CCAs each time log recovery is performed. Failure to do so can cause changes to be propagated to the publication service in the wrong chronological order. Additionally, if you are supplying multiple IMS log files for a CCA that is in recovery, you need to supply all IMS log files for the time period being recovered.

When the IMS recovery CCA is recovering IMS log files, the recovery process is fairly sophisticated. The IMS recovery CCA accepts IMS log files that were created before the restart point and automatically discards log records created before the restart point. If multiple IMS log files are supplied for an agent, these files are automatically sorted in system clock time sequence so that they are processed in the correct sequence. During the sorting process, the IMS recovery CCA automatically checks for specification of an IMS log file more than once in the control file and for IMS log files that have overlapping system times which indicates that an IMS log file for a different IMS active CCA was supplied. Additional checks are performed for IMS batch applications to insure that IMS log files are not supplied from a different job. However, the IMS recovery CCA cannot detect when an IMS log file that needs to be supplied for a batch job was not. For online regions, the IMS recovery CCA can detect when an online log is missing, provided the IMS subsystem has not been cold started.

Attention: If you start an IMS recovery CCA without all of the necessary log files, the correlation service is likely to detect this condition, but by that time, the restart point might be pushed into the future and the content of the missing log file will no longer be recoverable.

Use the following guidelines for identifying IMS log files to be recovered:

- IMS log files can be supplied that were created before the current restart point. Recovery CCA reads these files and discards the log records that were created before the current restart point.
If you supply more than one IMS log file for a CCA in recovery, you can identify the input IMS log files in any sequence. The IMS recovery CCA automatically re-sequences these files into the correct processing sequence based on the system clock values contained in each IMS log record.

If you are recovering multiple CCAs, they can be identified in any sequence as can the names of the input IMS log files. The IMS recovery CCA automatically match merges multiple CCA IMS log files in store clock sequence, and presents these IMS log records to the correlation service in the time sequence that the log records were created in.

Recovery may be performed as full log recovery (mode "L") or incremental log file recovery (mode “I”).

Incremental log file recovery:

Incremental recovery is required when you are recovering multiple IMS DB/DC or DBCTL subsystems, or when one or more of the subsystems are still running. In this situation you have IMS log files that contain log records whose system time stamps overlap, and you have different end points in these logs.

For this situation you can only recover the log files contents up to the end of the IMS log files for the older of the two subsystems. Incremental Log File Recovery can be specified for any IMS log file for the subsystem that is controlling the recovery process. You can also identify that incremental recovery is to be performed on multiple active CCA that are in recovery.

When incremental recovery is specified for any CCA in recovery, once all IMS log files for that CCA have been processed, the IMS recovery CCA stops processing IMS log files for the other CCAs and (normally) terminates processing. This allows you to move the recovery restart points for the CCAs in recovery forward in time.

When more IMS log files are available, you can input these files into the recovery process, possibly identifying a different CCA as being the controlling agent, and continue to push the restart recovery point further into the future. You continue performing incremental recoveries until all IMS control regions can be shut down — it is then possible to complete the recovery process by inputting all of the remaining IMS log files up to the last IMS log file created by each IMS control region that is in recovery mode.


Full log file recovery:

It differs from incremental log file recovery as follows. The IMS recovery CCA processes all input log files until end of file is reached on all logs, at which point it normally terminates. The correlation service normally identifies UOR(s) that are in-flight when a CCA terminates, and places the CCA into recovery mode. This behavior is different when changes come from the IMS recovery CCA. In this situation, for an online region, it is normal for UOR(s) to be in-flight at the end of a log file. No error is reported in this situation.

Log file recovery requires creating a customized control file that contains control cards for the CCAs in recovery. A control card must be supplied for each agent and IMS log file that is to be recovered as shown in Example 1-24, where log recovery is specified for the IMS control region IMS810K associated with log files with the fully qualified names of IMS810K.OLP04 and IMS810K.OLP04, and a batch CCA CACBACHG associated with a log file NALUR.LOG. The corresponding output is shown in Example 1-25.
Attention: If IMS log files are being written to a generation data set (GDG), you must specify the fully qualified data set name and not use relative generation numbers. This ensures that log files will not be fed in out of sequence due to the creation of a new generation. Also, in general, when using GDGs, you need to perform the recovery process up to the latest (current) generation.

Example 1-24  Sample log file recovery control cards

L IMS810K  4 IMS810K.OLP03
L IMS810K  4 IMS810K.OLP04
L CACBACHG 4 NALUR.LOG

Example 1-25  Sample log file recovery output

+CACH061I RECOVERY MODE: IMS LOG FILE RECOVERY
+CACH055I STARTING LOG FILE SEQUENCE CHECKING
+CACH058I LOG FILE SEQUENCE CHECKING COMPLETED
+CACH044I AGENT 'IMS810K ' LOG OPENED: IMS810K.OLP03
+CACH044I AGENT 'CACBACHG' LOG OPENED: NALUR.LOG
+CACH045I AGENT 'IMS810K ' LOG CLOSED: IMS810K.OLP03
+CACH044I AGENT 'IMS810K ' LOG OPENED: IMS810K.OLP04
+CACH045I AGENT 'IMS810K ' LOG CLOSED: IMS810K.OLP04
+CACH045I AGENT 'CACBACHG' LOG CLOSED: NALUR.LOG
+CACH072I BUFFERS SENT 4 RECORDS 56 THROTTLES 4
+CACH062I RECOVERY PROCESSING COMPLETED SUCCESSFULLY

The first WTO message is CACH061I, which identifies that the IMS recovery CCA is performing IMS log file recovery. The next message, CACH055I, is issued after the input file has been read, the IMS recovery CCA has verified that the CCA is in recovery mode, and communications with the correlation service have been established. It indicates that the IMS recovery CCA is in the process of verifying (to the best of its ability) that the correct set of IMS log files have been provided. During this process, the IMS log files are opened and their contents read when either multiple IMS log files have been identified for a CCA in recovery, or when you have requested incremental log file recovery.

After log file sequence checking is completed, then message CACH058I is issued identifying that sequence checking has been completed successfully and that IMS log file recovery is ready to commence.

If an invalid IMS log file is supplied, one of the following WTO messages is issued and the IMS recovery change-capture agent immediately terminates with a non-zero return code:

- CACH056E AGENT Agent-Name DUPLICATE LOG: IMS-Log-File-Name
  This message is self explanatory. The IMS log file identified in the message has the same starting or ending system time stamp values as another IMS log file associated with the CCA.

- CACH057E AGENT Agent-Name INVALID LOG: IMS-Log-File-Name
  Message CACH057E is more ambiguous. This message is issued when an overlap in system time stamps has been discovered with the identified IMS log file and another IMS log file associated with the CCA. The general implication of this message is that you specified the name of IMS log file that is associated with some other IMS control region.

- CACH073E AGENT 'Agent-Name ' INVALID BATCH LOG: -Log-File-Name
  The IMS log file was created by a different batch job.
CACH074E AGENT 'Agent-Name' INVALID ONLINE LOG: -Log-File-Name
The IMS log file was created by a different DB/DC or DBCTL subsystem.

CACH075E AGENT 'Agent-Name' MISSING ONLINE LOG BEFORE: -Log-File-Name
The IMS recovery CCA inspected “type 43” log records (log data set control records) and detected that an online log created before the one identified by Log-File-Name was not supplied.

Message CACH044I is issued each time an IMS log file is opened for recovery processing. After the entire contents of an IMS log file are read and processed, message CACH045I is issued. If you supply an IMS log file that contains log records that were created before the recovery restart point, the number of IMS log records sent to the correlation service is zero. The IMS recovery CCA automatically filters these types of record outs. The last message issued is CACH062I that identifies that the IMS log files were recovered successfully — CACH062E is issued if errors encountered. In Example 1-25, all IMS log files were processed successfully and no errors were reported.

Note: When you are performing incremental recovery, after the IMS recovery CCA completes processing, always re-run the IMS active agent status job to see how much farther into the future the recovery restart points have been pushed. In most cases you will have to supply the last processed IMS log file for each agent in recovery back into the IMS log file recovery process when you perform the next incremental recovery run, or the final recovery run.

It is important for the user should wait for awhile before checking the status again. What has happened is that the IMS recovery CCA has sent the correlation service a number of changes to be processed. The IMS active agent status check job reports on the current restart point. If there are still log records to be processed and you run the IMS recovery CCA again, before it starts sending new log records, it places the CCA into an “error” state that forces the correlation service to discard all existing changes that are cached in the message store and that may arrive via the XM queue. The idea of waiting awhile is to let the correlation service process complete its processing and then run the check job. Generally, you should issue the correlation service REPORT command until the pending queue has been drained down to zero, and you then run the check job again and provide in another set of logs.

The correlation service records, as the recovery restart point, the log record of the oldest living unit-of-work when a unit-of-work is started. Generally, for a DB/DC or DBCTL subsystem, there are always active units-of-work so that the restart recovery point almost never matches the physical end of file for an IMS log file. An exception to this rule is when you supply an IMS log file that was the last log file produced for a DB/DC or DBCTL subsystem before the subsystem was shut down.

STEP 4: Place all related IMS CCAs in active mode
The previous step has to be repeated until all the log files have been processed.

After you complete the recovery process for all of your IMS active CCAs that are in recovery mode, you use the IMS recovery CCA to place those CCAs in recovery mode back into active mode. CCAs can be successfully placed back into active mode once the following conditions are met:

- The IMS control region when the active agent was running is shut down.
- All log files created by the IMS control region in recovery have successfully gone through the IMS log file recovery process.
Important: Placing an IMS CCA in active mode is a “force” action — recovery CCA will set a CCA to active regardless of whether or not recovery was successfully completed. You should therefore perform this action only when it is safe to do so as discussed in “When it is safe to place a CCA back in active mode” on page 68.

Setting agents back into active mode requires creating a customized control file that only contains control cards for the CCAs that need to be activated as shown in Example 1-26 where CAC.RECOVDS.IMS810K and CAC.RECOVDS.CACBACHG are the recovery data sets associated with the IMS810K and CACBACHG CCAs respectively. The corresponding output is shown in Example 1-27.

Example 1-26  Sample set agent in active mode control cards

A IMS810K 4 CAC.RECOVDS.IMS810K
A CACBACHG 4 CAC.RECOVDS.CACBACHG

Example 1-27  Sample set agent in active mode output

+CACH061I RECOVERY MODE: ACTIVATE AGENT
+CACH054I PREPARING TO ACTIVATE AGENT 'IMS810K'
+CACH047E RECOVERY DATASET OPEN ERROR FOR CAC.RECOVDS.IMS810K
+ ERRNO: 19
+CACH054I PREPARING TO ACTIVATE AGENT 'CACBACHG'
+CACH031I AGENT 'CACBACHG' SWITCHING TO ACTIVE MODE
+CACH062V RECOVERY PROCESSING ENDED WITH WARNINGS

The first WTO message CACH061I identifies that the IMS recovery CCA is preparing to place one or more agents back into active mode. For each CCA being activated, message CACH054I is issued indicating that the IMS recovery CCA is about to attempt to re-activate the CCA. If the CCA is successfully returned to active mode, then message CACH031I is issued. If the CCA is reported as not being in recovery or there are problems communicating with the correlation service, message CACH031I is not issued, and you will see other error messages that identify what the problem is.

In Example 1-27, even though message “CACH047E RECOVERY DATASET OPEN ERROR FOR CAC.RECOVDS.IMS810K ERRNO: 19” was issued, the activation was successful. This is a normal error message that is issued when you activate a CCA for a DB/DC or DBCTL subsystem and that subsystem is operational.

Attention: When the message CACH047E is issued, it indicates that the IMS recovery CCA was not able to delete the contents of the restart dataset. If there is a record in that dataset and the IMS DB/DC or DBCTL subsystem is recycled without a correlation service being active, the IMS active CCA will not update the contents of the restart dataset — therefore, the restart point is way in the past. If a CACH047E message is issued, then extreme caution needs to be used. Ideally, a shutdown of the IMS subsystem should be scheduled and while the subsystem is down the system administrator needs to manually edit the recovery dataset and delete any record that may exist in there.

As part of activating a CCA, the contents of the recovery data set are deleted. After all IMS CCAs in recovery are placed into active mode, WIICEP for IMS recovery is complete and you can resume normal IMS and WIICEP for IMS operations.
When it is safe to place a CCA back in active mode

Consider the following rules to determine when an IMS active CCA can be removed from recovery mode and placed back into active mode:

- The IMS control region for the CCA in recovery mode is not active, or the databases being monitored have been stopped so that changes cannot occur, or you have reached a “logical” quiesce point, and you know that changes to the monitored databases will not occur.
- The IMS log file recovery process must be completed for each IMS log file (or archived log files, for DB/DC or DBCTL subsystems) created by the IMS control region from the failure point, up to and including the last IMS log file created by the IMS control region where the CCA failed.

If these conditions apply to an IMS active CCA in recovery mode, and you have a situation where multiple IMS active CCAs are in recovery mode, leave all CCAs in recovery mode until all CCAs meet the above conditions. Doing so lessens the likelihood of propagating changes in the wrong chronological order.

1.6.4 Conditions when recovery is not possible

Recovery may not be possible for CCAs in recovery mode under the following conditions:

- There is a media failure for an IMS log file that is needed in the IMS log file recovery process and dual logging was not in effect, or both log files have media problems.
- The IMS log or archived log files required for IMS log file recovery processing no longer exist. This condition should occur only when you defer the recovery process for an extended period of time.
- Using archived log files in the recovery process and the IMS Log Archive Utility has supplied an SLDS control statement that removed log records of interest to WIICEP for IMS from the archived log file.
- An IMS log file contains a “type 99” data capture log record that is marked “in error.” This situation should occur only when cascade delete information is being recorded.

Under normal circumstances, the first condition should never occur, nor the second condition, assuming that operations personnel are monitoring WIICEP for IMS’ operations.

WTO messages are issued by the IMS active CCAs, the correlation service, or both when a problem is encountered. When a recovery situation is identified, begin the recovery process as soon as possible. This ensures that the IMS log files required still exist and helps reduce the time agent remains in recovery mode.

1.6.5 Methods to determine if a CCA has updated a monitored database

There are several methods that you can use to determine whether a CCA updated a monitored database. The recommended approach is to use member name CACIMSLA in SCACSMAMP to determine whether the IMS log files associated with the IMS CCA that is in recovery updated any monitored databases.

CACIMSLA executes the IMS File Select and Formatting Print Utility (DFSERA10). The input control cards search the IMS log files for “type 99” data capture log records. If a data capture log record exists in the IMS log files, its content is printed and execution of the utility is terminated.
If you update CACIMSLA to reference the IMS log files for the IMS active CCA that is in recovery mode, and a data capture log record is found, then recovery is required for this CCA.

If no data capture log records exists, then recovery is not required.

- If the correlation service knows the IMS active CCA is in recovery mode, then all you need to do is “activate” the IMS CCA that is in recovery mode using the IMS recovery CCA as described in “STEP 4: Place all related IMS CCAs in active mode” on page 66.

- If the correlation service does not report the IMS CCA in recovery mode, then run the IMS recovery CCA to “set” the agent in recovery mode as described in “STEP 2: Place all related IMS CCAs in recovery mode” on page 58. Then run the IMS recovery CCA again to “activate” the IMS agent as described in “STEP 4: Place all related IMS CCAs in active mode” on page 66.

By performing these operations, the IMS CCA will not go into recovery mode because the contents of the restart data set are cleared during “activate” the next time that IMS control region is executed.

1.6.6 Best practices to minimize the occurrence of recovery mode

The most common causes of recovery mode for which one can take precautions to minimize are as follows:

- Ensure that startup/shutdown sequence of the various components of the event publishing environment is followed rigorously.

- Ensure that updates to the metadata catalog and configuration file are planned well ahead of schedule so that they can be installed during periods of scheduled IMS region downtime.

- Ensure that the XM data space queue is adequately configured to prevent XM queue overruns.

- Ensure that WebSphere MQ manager is configured to run continuously without interruption during WIICEP for IMS operation.

- Avoid IMS Online Change of DBDs if possible.

1.6.7 Detection of recovery mode

An IMS active CCA may go into recovery mode for a number of reasons, and such events are written as WTO messages. A recovery mode condition needs to be detected as soon as possible to avoid publishing of event messages in the correct chronological sequence when multiple related IMS CCAs are involved.

The IMS active CCA issues WTO messages informing you when an IMS CCA is in recovery mode or has experienced some other error that has caused it to go into recovery mode.

Likewise, the correlation service issues normal processing and error WTO messages. These messages inform you of any of the following states:

- Changes are being received from an IMS active CCA.

- The CCA has shut down, that is, the IMS control region where the IMS active CCA was running, has terminated.

- The IMS active CCA is in recovery mode.
Tip: We recommend using an automated operator facility (AOF) to identify the various WTO messages that can be issued, monitor for these, and allow the AOF to notify you when one or more IMS active CCAs go into recovery mode.

Table 1-9 lists some of the WTO messages that should be monitored as potential indicators of the occurrence of recovery mode in one or more IMS active CCAs.

<table>
<thead>
<tr>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CACG001I CORRELATION SERVICE IS COLD STARTING</td>
<td>Self explanatory</td>
</tr>
<tr>
<td>CACG001I CORRELATION SERVICE IS WARM STARTING</td>
<td>Self explanatory</td>
</tr>
<tr>
<td>CACG005I CHANGE CAPTURE SET FOR IMS TABLE ‘T1’</td>
<td>Identifies all the IMS “logical” tables for which change capture is enabled</td>
</tr>
<tr>
<td>CACG113I AGENT ‘xxxx’ SWITCHED TO ACTIVE MODE</td>
<td>Identifies the specific CCA that is switched from recovery mode to active mode</td>
</tr>
<tr>
<td>CACH021E UNKNOWN LOGGER EXIT FUNCTION</td>
<td>IMS active CCA was called with an unknown IMS Logger Exit function code. The exit has totally deactivated itself and will not attempt to inform the correlation service that a problem was encountered. IMS must be recycled to reactive the agent.</td>
</tr>
<tr>
<td>CACH022E IMS CHANGE CAPTURE AGENT ABENDED</td>
<td>IMS active CCA has abnormally terminated and its ESTAT routine has received control. The exit has totally deactivated itself and will not attempt to inform the correlation service that a problem was encountered. IMS must be recycled to reactive the agent.</td>
</tr>
<tr>
<td>CACH023E IMS CHANGE CAPTURE AGENT STORAGE ALLOCATION FAILED</td>
<td>IMS active CCA could not allocate required storage during initialization processing. The exit has totally deactivated itself and will not attempt to inform the correlation service that a problem was encountered. IMS must be recycled to reactive the agent.</td>
</tr>
<tr>
<td>CACH024E XM DATAGRAM FAILED</td>
<td>IMS active CCA reported an error writing an XM data gram. The IMS active CCA attempts to notify the correlation service that a recovery situation has occurred.</td>
</tr>
<tr>
<td>CACG153E MEMORY CORRUPTION DETECTED</td>
<td>The correlation service has detected an internal memory corruption and is shutting down. The correlation service must be recycled and then normal recovery operations can begin.</td>
</tr>
<tr>
<td>CACG316E IMS UOR IN-FLIGHT</td>
<td>Correlation service has been shut down and in-flight UORs were being tracked at time of shutdown. This message is issued for each agent that was actively tracking in-flight UORs. Each of these agents are placed in recovery mode. Upon correlation service restart normal recovery operations can begin for each of these agents.</td>
</tr>
</tbody>
</table>
The alternative is to be on the lookout for these WTO messages manually.

A better approach is to create a special version of the IMS recovery CCA that is used to track the status of all of the IMS active CCAs at your site. This is referred to as the “IMS Active Agent Status Job”, and the control file is referred to as the “IMS Status Control File”. The “IMS Active Agent Status Job” should be run periodically, and it identifies:

- Whether any of the IMS active CCAs are in recovery
- The restart point for any agents in recovery
- The IMS log files that are presently available for recovery, for agents that are in recovery, using IMS log file tracking.

If there are any IMS active CCAs in recovery mode, a CACH071I WTO message is issued identifying the number of CCAs in recovery mode.
WebSphere Information Integrator Event Publisher architecture

In this chapter we provide an overview of WebSphere Information Integrator Event Publisher (WIIEP), and its architecture and processing flow.

The topics covered are:
- WebSphere Information Integrator Event Publisher (WIIEP) architecture
- WIIEP processing flow
- WIIEP administration
2.1 Introduction

WebSphere Information Integrator makes it easy to link data events with business processes. It complements and extends client investments in service oriented architecture, enterprise application integration, and extract-transform-load infrastructure by eliminating the hand coding typically required to detect data changes and removing the extra overhead from the transaction itself.

Figure 2-1 shows the WebSphere Information Integrator’s event publishing process where changes in databases are captured by reading the recovery log, formats the changes into XML messages, and publishes them to WebSphere MQ. Thus the events are available to a user application, or first brokered by the WebSphere Business Integration Message Broker (formerly known as MQSI – MQ Series Integrator) and then passed on to other applications, or the data could be brokered by the MQ Listener function of DB2 on LUW or z/OS, and then passed on to your user written stored procedure.

Event publishing can be used to provide information to central information brokers and Web applications or to trigger actions or processes that are based on updates, inserts, or deletions to source data. Event publishing may be used to:

- Integrate applications, for example, a change to the marital status captured in a home grown application needs to be reflected in an ERP application.
- Initiate business processes, for example, the addition of a new customer record could initiate welcome E-mail, credit verification, accounting update, and input into the CRM system.
- Monitor events, for example, a change in an inventory value could be used to drive a product restocking workflow.
- Feed data warehouses, for example, pushing only changed data to an extract-transform-load (ETL) product that populates a data mart or data warehouse.
These are some benefits of event publishing:

- XML messages are created asynchronously from the originating application, thereby reducing the performance impact on that application.

- The originating application is similarly shielded from any loss of availability of the message queue or service. For example, in an order processing source system where a new order triggers an event that feeds an XML message to a billing transaction and shipping system, new orders can continue to be created while the connectivity to the in-house billing system or the external shipping system is lost, even if queues are temporarily overfilled. When the connectivity is regained and message queues are again available, the orders can be processed and sent on.

- Disparate systems can be linked together. For example, changes in customer data stored in an IMS system can be published as XML messages that can be processed by a DB2 database on a distributed platform.

Event publishing capability is delivered in multiple WebSphere II editions of WebSphere Information Integrator Version 8.2 for DB2 Universal Database (UDB) for Linux, UNIX, or Windows as follows:

- Event Publisher edition
- Replication edition
- Standard edition
- Advanced edition
- Advanced Unlimited edition
- Developer edition

**Note:** On the z/OS platform, the event publishing capability is delivered as part of WebSphere Information Integrator Replication for z/OS (5655-L88), or purchased separately as WebSphere Information Integrator Event Publisher for DB2 UDB for z/OS (5655-M36).

The following sections describe:

- WebSphere Information Integration Event Publisher architecture
- WebSphere Information Integration Event Publisher processing flow
- WebSphere Information Integration Event Publisher administration

### 2.2 WIIEP architecture

WebSphere Information Integrator Event Publisher captures database changes in DB2 UDB by reading the recovery log, formats the changes into XML messages, and publishes them to WebSphere MQ. Any application or service that integrates with WebSphere MQ directly or supports Java Message Service (JMS) can asynchronously receive the data changes as they occur as shown in Figure 2-2.
WIIEP formats data into either transaction messages or row messages depending upon application need. It also filters database changes such that only the data that the receiving application is interested in are published. Publication actions are specified with easy to use wizards.

Figure 2-2 shows applications updating source tables that have their changes written to a log by DB2. These changes are then captured by a program called the Q Capture program. The Q Capture program uses a set of control tables, called Q Capture control tables, to store the information that the Q Capture program requires to perform its tasks (such as information about what its sources are and what to publish from its sources) and to store monitoring information. You can run multiple Q Capture programs on the same DB2 UDB server. Each Q Capture program uses its own set of control tables. The schema associated with a set of Q Capture control tables identifies the Q Capture program that uses those control tables. This schema is called a Q Capture schema.

The following objects exist between servers:

- **Publishing queue maps**
  
  A publishing queue map identifies the WebSphere MQ queue that a Q Capture program uses to transport data, and communicate with the user application that is receiving the XML messages. Each publishing queue map identifies one send queue, which is the WebSphere MQ queue where the Q Capture program sends source data and informational messages.

---

1 This is the same Q Capture program that is used in Q replication. In fact, it is possible for one Q Capture program to be involved in Q replication and event publishing at the same time.
In event publishing, the Q Capture program publishes the source data in XML format to the send queue. Unlike the case with Q replication\(^2\), the Q Apply program does not retrieve those messages. You can configure WebSphere MQ or an application of your choice to handle the XML messages that are on the send queue.

A single publishing queue map may be used to transport data for one or more XML publications. Also, a single Q Capture program can work with multiple publishing queue maps. In general, you should create XML publications that all share a common publishing queue map for tables that are involved in transactions with one or more applications. Grouping similar XML publications with the same publishing queue map assures the transactional consistency of the data that is sent to the send queue.

In addition to identifying the WebSphere MQ queue that the Q Capture program sends changes to, a publishing queue map also contains publishing attributes that involve the send queue. For each publishing queue map, you specify the following attributes that involve the send queue:

- The WebSphere MQ name of the send queue.

  **Note:** The same send queue cannot be used for both event publishing and Q replication, since a queue-level attribute specifies whether it transports XML messages for event publishing, or compact messages for Q replication.

- The type of message content that the send You can choose whether the XML messages that are sent by the Q Capture program contain only a single row change (also called row operation message) from the source table, or an entire transaction (also called a transaction message). The Q Capture program will use the message format that you choose for all XML publications that specify this publishing queue map.

  **Note:** Regardless of the row or transaction operation chosen, LOB data types are sent separately as individual physical messages that are associated with either the transaction message or row operation message.

- The maximum size of a message that the Q Capture program can put on this send queue. This limit is independent of the WebSphere MQ Series maximum message length, but this limit must be less than or equal to the WebSphere MQ Series maximum message length.

- How the Q Capture program responds if an error occurs at the WebSphere MQ queue. The choices are to deactivate the XML publications that use the queue, or stop the Q Capture program.

- How often the Q Capture program sends heartbeat messages to a send queue to tell the user application that the Q Capture program is still running when there are no changes to publish. Also called a heartbeat interval, the heartbeat message is sent on the first commit interval after the heartbeat interval expires. The heartbeat interval is different from the WebSphere MQ parameter HBINT that can be defined for a WebSphere MQ channel.

  **Note:** At least one publishing queue map must be defined to transport data from the Q Capture program on each source server.

XML publications:

For each source table that you want to publish changes from, you must create at least one XML publication. The XML publication specifies the table, as well as the rows and columns that you want to publish.

You can have multiple XML publications that specify the same source table. For example, if you wanted to publish changes from the EMPLOYEE table to Application A and Application B, and changes from the DEPARTMENT table to Application A, you would create three XML publications, with two publishing queue maps:

- XML publication EMPLOYEE0001 from the EMPLOYEE table to Application A using publishing queue map A.
- XML publication DEPARTMENT0001 from the DEPARTMENT table to Application A using publishing queue map A.
- XML publication EMPLOYEE0002 from the EMPLOYEE table to Application B using publishing queue map B.

Application A reads the messages that the Q Capture program sends to the send queue in publishing queue map A, and Application B reads the messages that the Q Capture program sends to the send queue in publishing queue map B.

The XML publication specifies how changes from a single source are published in XML format to a WebSphere MQ queue — these XML messages can then be retrieved and consumed by an application. The following attributes are specified for an XML publication:

- Source server and source table from which you want to publish changes.
- Columns and rows you want published from the source table.
- Predicate for publishing a subset of rows. For example, WHERE :LOCATION = 'EAST' AND :SALES > 100000
- Publication queue map to use.
- Whether values in unchanged on-key columns should be published together with updates to other non-key columns.
  
  By default, if there are updates to one or more of the columns that you selected, the updates are published but the values in the unchanged columns are not.

  For example, from table T1 you want to publish columns A1 (the primary key), A2, and A3. If a transaction updates a value in column A2 and then commits, the Q Capture program will publish a message that contains only the new value in column A2, as well as the value in the key column A1.

  You can choose to have the messages contain updated and unchanged values in non-key columns.

  If column A2 is updated, the Q Capture program will publish a message that contains the new value in column A2 and the unchanged value in column A3, in addition to the value in the key column A1.

  You might want to choose this option if it is easier to write your application to always expect a value for each column. You also might want to choose this option if you are writing an application to audit changes to

  **Note:** An entire LOB is published or replicated even if a small portion of the LOB is changed.

- Whether to include new and old values for the data in updated non-key columns, or only the new values.
• By default, when there are updates to one or more of the columns that you selected, the message containing those updates will provide only the new values.

For example, from table T2 you want to publish columns B1 (the primary key) and B2. If a transaction updates column B2 from 5 to 6, the message that contains that update will provide only the value 6.

• You can choose to have the messages provide the old values together with the new ones. In this case, the message that contains the update to column B2 would provide both the value 5 and the value 6. Choose to send old values if the application that receives the published changes requires both old and new values.

For example, your application might be aggregating information and finding the difference between the old and new values. Also, if you are publishing significant price changes to a Web application and that application requires the old prices, sending the old prices along with the new prices saves the application from having to look up the old prices elsewhere.

– Whether the XML publication, when it is first created, is activated automatically when you start or re-initialize the Q Capture program (this is the default behavior). You can also choose to create the XML publication in an inactive state, which requires you to activate the XML publication for the Q Capture program to begin capturing changes.

**Important:** XML publications are separate objects from Q subscriptions. XML publications do not publish data to the Q Apply program, but to an application of your choice. XML publications are for publishing data, and Q subscriptions are for replicating data. If you want to replicate changes from a source table and want the Q Apply program to apply those source changes to a target table or pass them to a stored procedure for data manipulation, you create a Q subscription, not an XML publication.

Table 2-1 highlights some of the key feature differences between Q replication and event publishing functionality in WebSphere Information Integrator.

*Table 2-1  Comparing the main features of event publishing and Q replication*

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Q replication</th>
<th>Event publishing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uses</td>
<td>Multiple, including failover, capacity relief, supporting geographically distributed applications, data availability for planned or rolling upgrades or outages.</td>
<td>Multiple, including feeding central information brokers and Web applications, and triggering actions based on updates, inserts, or deletes to source tables.</td>
</tr>
<tr>
<td>How data is replicated or published</td>
<td>Committed transactional data is captured and put on WebSphere MQ message queues by a capturing program. An applying program reads the information from message queues and applies the transactions to target tables.</td>
<td>Committed transactional data is captured and put on WebSphere MQ message queues by a capturing program.</td>
</tr>
<tr>
<td>Sources</td>
<td>Tables in DB2 UDB.</td>
<td>Tables in DB2 UDB</td>
</tr>
<tr>
<td>Targets</td>
<td>Tables in DB2 UDB. Heterogeneous targets include Oracle and Sybase with fixpack 9 (4/2005)</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>
2.3 WIIEP processing flow

Installation, planning, and configuration of an WIIEP environment is beyond the scope of this redbook. For details on these tasks, refer to the Program Directory for DB2 Information Integrator Event Publisher for DB2 UDB for z/OS, and DB2 Information Integrator Replication and Event Publishing Guide and Reference Version 8.2, SC18-7568.

In this section, we focus on the flow of changes occurring in DB2 tables through to the user application consuming the events in the form of XML messages.

The process of capturing changes at the source, transporting them over WebSphere MQ to the target, and processing the events occurs in three distinct phases as shown in Figure 2-3.

1. The first phase involves capturing changes from the log, formatting them as XML messages, and writing them to the WebSphere MQ send queue.
2. The second phase involves WebSphere MQ transporting the XML messages from the source to the target queue. This is optional if the consuming application resides on the same server as WebSphere MQ, as shown in Figure 2-2 on page 76.
3. The third phase involves processing the XML messages by the consuming application.
A brief review of Q Capture and WebSphere MQ follows.

### 2.3.1 Q Capture

The Q Capture program is a program that reads the DB2 recovery log for changes that occur in source tables, turns the changes into messages, and sends the messages over WebSphere MQ queues where the messages are processed by a user application in the case of event publishing, or by a Q Apply program in the case of Q replication.

As soon as an XML publication is activated, Q Capture begins collecting changes for the source table identified in the XML publication, and writes them out to a WebSphere MQ queue at the transaction boundary. The Q Capture program reads the DB2 log sequentially for changes to the source tables. If it reads a change to one of your source tables, the Q Capture program adds the change to the corresponding database transaction that it is retaining in memory. Transactions in memory are therefore potentially subsets of the corresponding transactions in the log; they contain only changes to your source tables.

When the Q Capture program reads the COMMIT statement for a transaction, it converts the transaction into one or more messages, depending on whether you want to send row-level messages or transaction-level messages and depending on how many send queues are being used by the source tables that are involved in the transaction. The Q Capture program then puts the message on the corresponding send queues:

- If a row change involves columns with large object (LOB) data, the Q Capture program copies the LOB data directly from the source table to the send queue.
- If you define a search condition, the Q Capture program uses it to evaluate each row in memory. Rows that meet the search condition are assembled into messages when the transaction that they belong to is committed at the source database.

For example, assume that you create three XML publications: PUB1 (source table T1), PUB2 (source table T2), and PUB3 (source table T3) where:

- PUB1 and PUB2 use the same publishing queue map (which uses SENDQ1), and you want to use these XML publications to publish messages that contain committed row-level changes.
- PUB3 uses a different publishing queue map (which uses SENDQ2), and you want to use this XML publication to publish messages that contain committed transaction-level changes.

The Q Capture program reads a COMMIT for a database transaction that involves changes to the source tables T1, T2 and T3 in all three XML publications. The Q Capture program converts parts of the transaction that involve:

- Source tables in PUB1 and PUB2 to row-level messages that it writes to SENDQ1.
- Source table in PUB3 to a transaction-level message that it writes to SENDQ2.

**Note:** You can run more than one Q Capture program on the same source server. Although one Q Capture program can capture changes made to many sources and send those changes to many user applications, in some situations you might benefit from running more than one Q Capture program. For example, multiple Q Capture programs, which could also be helpful in large sysplexes, can improve performance and achieve higher throughput. The trade-off is additional CPU overhead associated with multiple log readers. Using multiple Q Capture programs also requires more DB2 UDB connections.
A Q Capture program translates changes from a source table into XML format (row operation message or transaction message depending upon the option chosen) for event publishing. After the Q Capture program puts messages on one or more WebSphere MQ send queues, it issues a commit call to the WebSphere MQ queue manager instructing it to make the messages on the send queues available to the user applications. You can configure how often the Q Capture program commits messages. All of the DB2 transactions grouped within each commit are considered to be a single WebSphere MQ transaction. Typically, each WebSphere MQ transaction contains several DB2 transactions. You can adjust the time between commits by changing the value of the COMMIT_INTERVAL parameter of Q Capture. A shorter commit interval can lower end-to-end latency.

The Q Capture program uses a local WebSphere MQ queue to store restart information. The restart queue contains a single message that tells the Q Capture program where to start reading in the DB2 recovery log when the Q Capture program restarts. Each time that the Q Capture program reaches its commit interval, it checks to see whether or not it needs to update its restart information. If so, the Q Capture program replaces the message on the restart queue with a new message that contains relevant restart information including, among other things, the earliest point in the log at which it needs to start processing log records upon restart. When the cold start option is used, the Q Capture program replaces the restart message with a message that indicates for the program to start processing log records at the current point in the log.

The Q Capture program can be controlled by a number of parameters including the amount of memory that can be used to build transactions, the commit interval, and the monitor interval which specifies how often the Q Capture program writes to the trace tables. For details on configuring these parameters, refer to DB2 Information Integrator Replication and Event Publishing Guide and Reference Version 8.2, SC18-7568.

### 2.3.2 WebSphere MQ

WebSphere MQ ensures that every message\(^3\) generated by the Q Capture program is transported to the target without loss and in the correct order. Since the WebSphere MQ queues used by event publishing and replication must be defined as being persistent, the messages in the queues can survive crashes.

Depending on the type of replication or publishing to be performed, various WebSphere MQ objects are required. The following is a summary of the WebSphere MQ objects required by the Q Capture and Q Apply (not used by event publishing) programs with a brief description of their usage:

- **Queue manager** is a program that manages queues for Q Capture programs, Q Apply programs, and user applications. One queue manager is required on each system.
- **Send queue** is a queue that directs data messages from a Q Capture program to a Q Apply program or user application. In remote configurations, this is defined as a remote queue on the source system corresponding to the receive queue on the target system. Each send queue should be used by only one Q Capture program.
- **Receive queue** is a queue that receives data and informational messages from a Q Capture program to a Q Apply program. This is a local queue on the target system.
- **Administration queue** is a queue that receives control messages from a Q Apply program or a user application to the Q Capture program. This is a local queue on the system where the Q Capture instance runs. There is a remote queue definition on the system where the Q Apply program or a user application runs, corresponding to the administration queue where the Q Capture instance runs.

---

\(^3\) Q replication messages are persistent.
▶ **Restart queue** is a queue that holds a single message that tells the Q Capture program where to start reading in the DB2 recovery log after a restart. This is a local queue on the source system. Each Q Capture program must have its own restart queue.

▶ **Spill queue** is a model queue that you define on the target system to hold transaction messages from a Q Capture program, while a target table is being loaded. The Q Apply program creates these dynamic queues during the loading process based on the model queue definition, and then deletes them. The spill queue must have a specific name “IBMQREP:SPILL.MODELQ”.

**Note:** In Fixpak 9, which is due in April 2005, the spill queue name may be chosen by the user.

The WebSphere MQ objects must be defined to Q replication or event publishing.

**Note:** While the definitions of the WebSphere MQ objects need to be synchronized in WebSphere MQ and Q replication or event publishing, the sequence in which these are defined is immaterial. However, the IBM manuals recommend defining the objects in WebSphere MQ first, before defining them in Q replication or event publishing.

The WebSphere MQ objects are defined in Q replication or event publishing when the Q Capture and Q Apply (not used in event publishing) tables are defined, and when the publishing queue maps and replication queue maps (not used in event publishing) and are defined.

▶ For the Q Capture control tables, you must provide the name of a queue manager on the system where the Q Capture program runs, and the name of a local administration queue and local restart queue.

▶ For the Q Apply control tables, you must provide the name of a queue manager on the system where the Q Apply program runs.

▶ For the replication queue maps, you must provide the name of a send queue on the system where the Q Capture program runs, and a receive queue and administration queue on the system where the Q Apply program runs.

▶ For the publishing queue maps, you must provide the name of a send queue on the system where the Q Capture program runs.

**Attention:** In Fixpak 8, support is provided for a WebSphere MQ client which allows WebSphere MQ objects used by Q replication to be on a different server from the ones where Q Capture and Q Apply run. However, we recommend a local queue manager for performance reasons.

**Note:** The WebSphere MQ transmission queues and channels do not have to be defined for the Q replication or event publishing programs. They only need to be defined within the source and target queue managers.

WebSphere MQ objects allow multiple settings to control their behavior. The queue manager, for instance, allows you to limit the maximum size (MAXMSGL parameter) of the messages on the queue, while the queues allow you to specify the maximum number of messages (MAXDEPTH parameter) in the queue and whether there are to be persistent or shared. The channels allow you to set the disconnect interval which specifies the duration the channel is to remain open when there are no transactions to replicate. For details on these parameters, refer to *WebSphere MQ, System Administration Guide*, SC34-6079-01.
2.4 WIIEP monitoring tools

Two tools are available to administer and manage the WIIEP event publishing environment. These are the Replication Center and the Replication Alert Monitor.

2.4.1 Replication Center

The Replication Center creates and manipulates replication and event publishing objects by generating and then running operational commands and SQL scripts.

For example, when you start a Q Capture program by using the Replication Center, the Replication Center generates and then runs an operational command. As another example, when you delete a Q subscription or XML publication by using the Replication Center, the Replication Center generates and then runs an SQL script.

The Replication Center can generate operational commands to perform many tasks for the Q Capture program, Q Apply program, and Replication Alert Monitor. For example, the commands can start or stop the program, prune control tables, change parameters, or check program status. The Replication Center can also generate SQL scripts to create, modify, and delete objects. For example, the scripts can create, change, delete, or activate Q subscriptions or XML publications. The scripts can also create, change, or drop control tables, and they can create and delete alert conditions.

A detailed discussion of the Replication Center is beyond the scope of this redbook. For further details, refer to DB2 Information Integrator Replication and Event Publishing Guide and Reference Version 8.2, SC18-7568.

2.4.2 Replication Alert Monitor

The Replication Alert Monitor is a program that checks the status of the replication and event publishing environments. When the Replication Alert Monitor is running, it automatically checks the status of replication and event publishing and notifies designated persons about certain conditions that occur in the replication environment. The Replication Alert Monitor can send a notification when a threshold has been exceeded in your replication and event publishing environment. For example, when the QCAPTURE_MEMORY alert condition alert condition is specified, the Replication Alert Monitor will send a notification anytime the Q Capture program uses more memory than its threshold allows.

You can check the status of the replication and event publishing environments by using the following methods:

- View Replication Center windows that report statistics about the Q Capture, and Q Apply programs.
- Run select statements on the control tables to view statistics about the operation of these programs.

The Replication Alert Monitor automatically monitors the replication and event publishing environments across all of your operating systems. It checks replication on your servers and automatically alerts you to conditions that require attention.

A detailed discussion of the Replication Alert Monitor is beyond the scope of this redbook. For further details, refer to DB2 Information Integrator Replication and Event Publishing Guide and Reference Version 8.2, SC18-7568.
Trustworthy Insurance Customer Information Facility (CIF) and Event Alert System scenario

In this chapter, we describe a step-by-step approach to implementing a customer information facility (CIF) application and an Event Alert System application using WebSphere Information Integrator Classic Event Publisher and WebSphere Information Integrator Event Publisher for a hypothetical insurance company.

The topics covered are:

- Business requirement
- Environment configuration
- Step-by-step set up
3.1 Business requirement

Our fictitious company Trustworthy Insurance is a fast growing insurance company that is expanding through acquisitions of other insurance companies in different geographic markets and domains such as auto insurance and home insurance. The resulting IT environment comprises heterogeneous products from IBM.

To improve customer service and promote cross selling and up selling among its customers in the different markets and quickly detect potential fraudulent activity in the legacy systems, Trustworthy Insurance needed to rapidly implement the following features:

▶ A customer information facility (CIF) scenario that supported its entire customer base without having to engage in a massive migration effort of its legacy insurance applications, or consolidation of its IT environment.

▶ An Event Alert System that focused on patterns of activity that indicate potential customer attrition, fraud, or cross selling opportunities.

Trustworthy Insurance’s business requirements may be summarized as follows:

1. Rapidly develop a cost effective:
   - Customer information facility application that supports its insurance product portfolio of auto, and home and contents insurance.
   - Event alert system for the auto insurance, and home insurance legacy applications.

2. Avoid a costly and time consuming migration of the legacy applications.

3. Defer a consolidation of the existing IT environment of the acquired companies.

The CIF and Event Alert System requirements are described briefly in the following sections.

3.1.1 Customer Information Facility (CIF) requirements

Trustworthy Insurance’s auto insurance application uses the hierarchical IBM IMS DBMS on a z/OS platform, while the home insurance application uses a relational IBM DB2 UDB for z/OS DBMS on another z/OS platform. The individual database structures are shown in Figure 3-1.

![Figure 3-1 Auto insurance (hierarchic IMS) and Home (relational) DB2) structures](image)

Both databases contain information about customers, policies, premiums, and claims, but they differ in the following respects:

▶ The IMS segments do not correspond directly to DB2 tables.
The keys for the various entities customer, policy, premiums etc. have different data types and precision, and the key values are from unrelated domains. For example, a particular customer in the IMS databases may have an alphanumeric key value of “A00056”, while a customer in the DB2 database has a numeric key value of 25645.

There are differences in the number of columns, column names, column data types, and column precision.

**Note:** Such differences would be the norm with applications developed in different organizations using different DBMS software.

The CIF requirements are as follows:

- Create an operational data store (ODS) containing a holistic view of a customer’s insurance information — assuming that some customers have both auto insurance and home insurance policies, while others may have only one or the other insurance policies.
- Data model for the CIF to be different from either the auto insurance model or the home insurance model, but closer to that of the relational home insurance model.
- Form the basis for the development of new customer relationship management systems such as marketing, telemarketing, call center and customer service.

### 3.1.2 Event Alert System requirements

In the current competitive global market, it is essential for an organization to offer superior services, cut costs, retain customers and exploit new opportunities. An Event Alert System that rapidly detects business transactions of interest in the auto insurance (IMS) and home insurance (DB2) and alerts relevant personnel to take appropriate action is considered crucial. Examples of business transactions of interest include unusual claims (submission very soon after a policy purchase), policy cancellation, overdue premium payments, and policies with unusually large policy amounts.

At the same time, such a system needs to have minimal impact on the operational legacy systems.

### 3.2 Environment configuration

Our fictitious company Trustworthy Insurance’s IT operations are hypothetically spread over two geographically distributed sites — in practice, in our controlled environment, both the auto insurance application and the home insurance application were located in the same z/OS image.

We chose to implement WebSphere Information Integrator Classic Event Publisher and WebSphere Information Integrator Event Publisher for DB2 UDB on z/OS for our solution for the following reasons:

- A common event publishing infrastructure could be used to address both the CIF requirement as well as the event alert system requirement.
- Event publishing infrastructure of these products had minimal impact on the legacy applications because they both captured database changes from the logs.
- The products’ use of WebSphere MQ for the transport layer provided a very desirable low latency solution.
Both products generated events as XML messages using the same structure and format — thus making it easy for the message consuming application not to have to understand the differences between IMS database generated events and DB2 database generated events.

Figure 3-2 shows the configuration used in the Trustworthy Insurance’s CIF and event alert system applications, which include the following features:

- A single z/OS server located in Poughkeepsie, New York running both the auto insurance IMS database application, and the home insurance DB2 database application in Jose.
- A Windows 2000 DB2 UDB V8.2 server located in San Jose, California that hosts Trustworthy Insurance’s CIF database and event alert system consuming applications. The Windows platform is used because of its lower TCO.

Figure 3-3 shows the technical components of Trustworthy Insurance’s CIF and event alert system, as follows:

- **WebSphere Information Integrator Classic Event Publisher for IMS**. It is used to:
  - Capture changes to the auto insurance IMS database.
  - Format the changes into row operation XML messages (for the Event Alert System) and transaction messages (for the CIF).
  - The XML messages are in a standard xsd\(^1\).
  - Publish the changes as XML messages into WebSphere MQ Send Queues, as defined by the publications.

- **WebSphere Information Integrator Event Publisher for DB2 UDB on z/OS**. It is used to:
  - Capture changes to the home insurance DB2 UDB for z/OS database.
  - Format the changes into row operation XML messages (for the Event Alert System) and transaction messages (for the CIF).
  - The XML messages are in a standard XSD.
  - Publish the changes as XML messages into WebSphere MQ Send Queues, as defined by the publications.

---
\(^1\) An XSD (XML Schema Definition) specifies how to formally describe the elements in an XML document. This description can be used to verify that each item of content in a document adheres to the description of the element in which the content is to be placed.
WebSphere MQ V5.3:
It is deployed on both the source z/OS systems and the target Windows 2000 system. It takes XML row operation messages and transaction messages published on the Send Queues on z/OS, and sends them to Receive Queues on the Windows 2000 platform.

DB2 Universal Database V8.2 for Linux, UNIX & Windows on the Windows 2000 machine.
It is the database chosen to act as the ODS for the CIF.

Note: DB2 UDB for Windows is used in the examples in this redbook. However, all the examples used here can be ported to the Linux and UNIX platforms.

Various programs that “listen” for event messages being published via WebSphere MQ, and execute pre-defined actions as a result of that event such as:

- CIF Updater programs:
  These are custom-written java programs that process XML messages received from the z/OS system, and apply the changes to keep the CIF ODS in sync with the source IMS and DB2 UDB for z/OS databases.
  - The XML messages are interpreted using standard tools and classes that are available with WIICEP. The tools most extensively used in our scenario are the XML parser (to parse the event publishing xml message) and the event publishing toolkit (to execute program logic against the changed data message).
  - We modified the sample programs to include procedural logic to determine the common surrogate customer number used to identify customers in the ODS using a mapping table for the source database customer number.
  - The updates to the ODS are more complex than supported by the standard database replication mapping facility.

- Event notification programs:
  These are programs that receive the published XML messages as “events”, and take appropriate action depending upon the event. Several kinds of event notification programs are used in our scenario as follows:
  - Java event notification programs that use the same techniques as used in the CIF Updater programs.
  - DB2 UDB Message Queue Listener stored procedures that use the DB2 XML Extender to help process the XML event message.
  - DB2 UDB Message Queue Listener stored procedures, that use the WebSphere Information Integrator XML Wrapper to help process the XML event message.

Figure 3-3 also shows both WebSphere Information Integrator Classic Event Publisher for IMS, and WebSphere Information Integrator Event Publisher for DB2 UDB on z/OS publishing XML messages to two sets of queues — one set that is processed by the CIF Updater programs, and another set processed by the event notification programs.
### 3.3 Step-by-step set up

In this section, we document the step-by-step set up of the Trustworthy Insurance’s CIF and Event Alert System solution. Figure 3-4 lists the main steps involved in setting up the environment. Each of these steps is described in detail in the following subsections.

**Note:** Steps 6A and 6B may be executed in parallel or sequentially in any sequence, but we recommend that STEP 7 be done after steps 6A and 6B are completed successfully.
3.3.1 STEP 1: Design topology of the CIF and Event Alert System solution

Figure 3-5 shows the topology used in Trustworthy Insurance’s solution where:

- Auto insurance application runs on the z/OS platform with the IMS V8 database using the IMSK subsystem. This system has WebSphere Information Integrator Classic Event Publisher for IMS installed.

- Home insurance application runs on the same z/OS platform as the auto insurance application, with the DB2 V8 database using the DB2G subsystem. This system has WebSphere Information Integrator Event Publisher for DB2 UDB for z/OS installed.

**Note:** In the real world scenario involving mergers and acquisitions, these two applications would almost certainly reside on different z/OS platforms. For our contrived scenario, we chose to have both the IMS and DB2 databases on the same z/OS image.

- WebSphere MQ V5.3 on the z/OS platform, which is shared by both WIICEP for IMS and WIIEP for DB2 UDB for z/OS. The queue manager they share is MQ8G.

- WebSphere MQ V5.3 on the Windows platform which uses the queue manager QMCIF.

- Three queues each are associated with WIICEP for IMS and WIIEP for DB2 UDB for z/OS respectively where:
  - One queue that has transaction messages written to it for processing by the CIF Updater programs.
  - Two queues that have transaction messages and row operation messages written to them respectively for processing by the event notification programs.
CIF Updater programs process the transaction messages originating from the IMS auto insurance application and DB2 home insurance application and update DB2 tables using two-phase commit protocol.

Event notification programs process transaction and row operation messages originating from the IMS auto insurance application and DB2 home insurance application and have the potential to send E-mails, invoke another application, update a database or write to a file. If a database is written to, then two-phase protocol is invoked between WebSphere MQ and DB2.

Note: The CUSTLINK table in the DB2 database maintains a cross reference between the key of a customer in the CIF ODS and their corresponding keys in the IMS auto insurance database, and DB2 home insurance database. This table is not updated or maintained by the CIF Updater programs.

Table 3-1 through Table 3-8 show the information required to configure the environment shown in Figure 3-5.

**Table 3-1  System information**

<table>
<thead>
<tr>
<th>Items</th>
<th>z/OS platform</th>
<th>Windows 2000 platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>System name</td>
<td>SC53TS</td>
<td>TONGA</td>
</tr>
<tr>
<td>IP Hostname</td>
<td>wtsc53.itso.ibm.com</td>
<td>tonga.itso.ibm.com</td>
</tr>
<tr>
<td>IP Address</td>
<td>9.12.6.77</td>
<td>9.1.39.168</td>
</tr>
<tr>
<td>User/password</td>
<td>NEALE/xxxx</td>
<td>db2admin/xxxx</td>
</tr>
</tbody>
</table>

Figure 3-5  Trustworthy Insurance’s CIF & Event Alert System topology

Table 3-1 through Table 3-8 show the information required to configure the environment shown in Figure 3-5.
### Table 3-2  IMS DB/DC information

<table>
<thead>
<tr>
<th>Items</th>
<th>Item value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>IMS DB/DC V8.1</td>
</tr>
<tr>
<td>IMS Subsystem name</td>
<td>IMSK</td>
</tr>
<tr>
<td>IMS target distribution libraries</td>
<td>IMS810K.*</td>
</tr>
<tr>
<td>IMS PROCLIB member</td>
<td>SYS1.PROCLIB(IMS810K)</td>
</tr>
<tr>
<td>IMS RESLIB</td>
<td>IMS810K.SDSFRESL</td>
</tr>
<tr>
<td>Source database</td>
<td>AUTODB</td>
</tr>
<tr>
<td>DBD source data set for AUTODB</td>
<td>NALUR.DATA(AUTODB)</td>
</tr>
<tr>
<td>PSB source data set for AUTODB</td>
<td>NALUR.DATA(AUTOPSB1)</td>
</tr>
<tr>
<td>COBOL copybooks to define segment layouts</td>
<td>NALUR.DATA(CUSTOMER)</td>
</tr>
<tr>
<td></td>
<td>NALUR.DATA(ADDRESS)</td>
</tr>
<tr>
<td></td>
<td>NALUR.DATA(POLICY)</td>
</tr>
<tr>
<td></td>
<td>NALUR.DATA(CLAIM)</td>
</tr>
<tr>
<td></td>
<td>NALUR.DATA(PREMIUM)</td>
</tr>
<tr>
<td>DBDGEN JCL</td>
<td>NALUR.JCL(AUTODBD)</td>
</tr>
<tr>
<td>PSBGEN JCL</td>
<td>NALUR.JCL(AUTOPSB1)</td>
</tr>
<tr>
<td>ACBGEN JCL</td>
<td>NALUR.JCL(ACBGEN)</td>
</tr>
<tr>
<td>Copy staging libraries to active libraries JCL</td>
<td>IMS810K.INSTALIB(IV3E316J)</td>
</tr>
</tbody>
</table>

### Table 3-3  WebSphere Information Integrator Classic Event Publisher for IMS information

<table>
<thead>
<tr>
<th>Items</th>
<th>Item value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installation libraries</td>
<td>CAC.V8R2M0.*</td>
</tr>
<tr>
<td>Correlation Service GDG data sets</td>
<td>MODEL DSCB FOR CACRCVD=CAC.V8R2M0.CACRCVD</td>
</tr>
<tr>
<td></td>
<td>MODEL DSCB FOR CACRCVX=CAC.V8R2M0.CACRCVX</td>
</tr>
<tr>
<td>Metadata catalog</td>
<td>CAC.V8R2M0.CATALOG</td>
</tr>
<tr>
<td></td>
<td>CAC.V8R2M0.CATINDX</td>
</tr>
<tr>
<td>z/OS Cross Memory Resources</td>
<td></td>
</tr>
<tr>
<td>Cross Memory Service Name</td>
<td>XM1</td>
</tr>
<tr>
<td>Cross Memory Dataspaces and Queue Name</td>
<td>XQM/CSQ</td>
</tr>
<tr>
<td>“Logical” tables</td>
<td>➤ IMSKAUTO.CUSTOMER</td>
</tr>
<tr>
<td></td>
<td>➤ IMSKAUTO.ADDRESS</td>
</tr>
<tr>
<td></td>
<td>➤ IMSKAUTO.POLICY</td>
</tr>
<tr>
<td></td>
<td>➤ IMSKAUTO.CLAIMS</td>
</tr>
<tr>
<td></td>
<td>➤ IMSKAUTO.PREMIUMS</td>
</tr>
<tr>
<td></td>
<td>➤ IMSKAUTO.QUICKCLM</td>
</tr>
<tr>
<td>Items</td>
<td>Item value</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| Publications                      | ► PUBCUSTN (transaction message, both old and new data values)  
|                                   | – Source table is IMSKAUTO.CUSTOMER  
|                                   | ► PUBCUSTA (transaction message, both old and new data values)  
|                                   | – Source table is IMSKAUTO.ADDRESS  
|                                   | ► PUBPOLY (transaction message, both old and new data values)  
|                                   | – Source table is IMSKAUTO.POLICY  
|                                   | ► PUBPREM (transaction message, both old and new data values)  
|                                   | – Source table is IMSKAUTO.PREMIUMS  
|                                   | ► PUBCLAIM (transaction message, both old and new data values)  
|                                   | – Source table is IMSKAUTO.CLAIMS  
|                                   | ► QUIKCLM (row operation message, only new data values)  
|                                   | – Source table is IMSKAUTO.QUIKCLM  |

Table 3-4  DB2 UDB for z/OS information

<table>
<thead>
<tr>
<th>Items</th>
<th>Item value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB2 version</td>
<td>DB2 for z/OS V7.1</td>
</tr>
<tr>
<td>DB2 subsystem name</td>
<td>DB2</td>
</tr>
</tbody>
</table>

Table 3-5  WebSphere Information Integrator Event Publisher - DB2 UDB for z/OS information

<table>
<thead>
<tr>
<th>Items</th>
<th>Item value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installation libraries</td>
<td>ASN.V8R2M0.*</td>
</tr>
<tr>
<td>Q Capture schema</td>
<td>ASN</td>
</tr>
</tbody>
</table>
| Publication Queue Maps            | ► EVPUB1 (transaction message)  
|                                   | ► EVPUB2 (transaction message)  
|                                   | ► EVPUB3 (row operation message) |
| Publications                      | ► CUSTOMER (both changed and unchanged columns, both old and new data values)  
|                                   | ► POLICY (both changed and unchanged columns, both old and new data values)  
|                                   | ► CLAIM (both changed and unchanged columns, both old and new data values)  
|                                   | ► PREMIUM (both changed and unchanged columns, both old and new data values) |

Table 3-6  WebSphere MQ information

<table>
<thead>
<tr>
<th>Items</th>
<th>Item value</th>
</tr>
</thead>
<tbody>
<tr>
<td>WebSphere MQ on the z/OS platform</td>
<td>WebSphere MQ V5.3</td>
</tr>
</tbody>
</table>
### Table 3-7  DB2 UDB CIF ODS information

<table>
<thead>
<tr>
<th>Items</th>
<th>Item value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB2 UDB version</td>
<td>DB2 UDB V8.2 Fixpak 9</td>
</tr>
<tr>
<td>ODS database name</td>
<td>CIF</td>
</tr>
<tr>
<td>DB2 Listener Port</td>
<td>50000</td>
</tr>
<tr>
<td>WebSphere Information Integrator</td>
<td>WebSphere II Advanced Edition Fixpak 9</td>
</tr>
</tbody>
</table>

### Table 3-8  CIF updater programs information

<table>
<thead>
<tr>
<th>Items</th>
<th>Item value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java version</td>
<td>JRE 1.4</td>
</tr>
<tr>
<td>Libpath</td>
<td>/usr/mqm/java/lib</td>
</tr>
</tbody>
</table>
3.3.2 STEP 2: Install prerequisite software

The installation of the prerequisite software is beyond the scope of this redbook. Please refer to the appropriate installation manuals for installing WebSphere Information Integrator Classic Event Publisher for IMS, WebSphere Information Integrator Event Publisher for DB2 UDB for z/OS, WebSphere MQ (on z/OS and Windows 2000), DB2 Connect™ Personal Edition, and DB2 UDB V8.2 for Windows.

<table>
<thead>
<tr>
<th>Items</th>
<th>Item value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classpath</td>
<td>/usr/mqm/java/lib/com.ibm.mq.jar</td>
</tr>
<tr>
<td></td>
<td>/usr/mqm/java/lib/com.ibm.mqbind.jar</td>
</tr>
<tr>
<td></td>
<td>/usr/mqm/java/lib/com.ibm.mq.jms.jar</td>
</tr>
<tr>
<td></td>
<td>/usr/mqm/java/lib/connector.jar</td>
</tr>
<tr>
<td></td>
<td>/usr/mqm/java/lib/samp/fscontext.jar</td>
</tr>
<tr>
<td></td>
<td>/usr/mqm/java/lib/jms.jar</td>
</tr>
<tr>
<td></td>
<td>/usr/mqm/java/lib/jndi.jar</td>
</tr>
<tr>
<td></td>
<td>/usr/mqm/java/lib/idap.jar</td>
</tr>
<tr>
<td></td>
<td>/usr/mqm/java/lib/postcard.jar</td>
</tr>
<tr>
<td></td>
<td>/usr/mqm/java/lib/providerutil.jar</td>
</tr>
<tr>
<td></td>
<td>/usr/mqm/java/lib/rmm.jar</td>
</tr>
<tr>
<td></td>
<td>/yourpath/xmlPubTk.jar</td>
</tr>
<tr>
<td></td>
<td>/SQLLIB/java/db2java.zip;</td>
</tr>
<tr>
<td></td>
<td>/SQLLIB/java/db2jcc.jar;</td>
</tr>
<tr>
<td></td>
<td>/SQLLIB/java/sqlj.zip;</td>
</tr>
<tr>
<td></td>
<td>/SQLLIB/java/db2jcc_license_cisuz.jar</td>
</tr>
<tr>
<td></td>
<td>/SQLLIB/java/db2jcc_license_cu.jar</td>
</tr>
<tr>
<td></td>
<td>/SQLLIB/bin;</td>
</tr>
<tr>
<td></td>
<td>/SQLLIB/java/Common.jar;</td>
</tr>
<tr>
<td></td>
<td>/SQLLIB/java/db2cmn.jar;</td>
</tr>
<tr>
<td></td>
<td>/SQLLIB/java/db2replapis.jar;</td>
</tr>
<tr>
<td></td>
<td>/SQLLIB/java/db2qreplapis.jar;</td>
</tr>
<tr>
<td></td>
<td>/SQLLIB/java/jt400.jar;</td>
</tr>
</tbody>
</table>

3.3.3 STEP 3: Unload auto insurance and home insurance data

The CIF ODS on DB2 UDB on the Windows platform needs to be initialized with auto insurance data from IMS and home insurance data from DB2 UDB for z/OS.

This should be performed after all updates are suspended by making the databases read only.

3.3.4 STEP 4: Stop WIICEP for IMS and WIIEP for DB2 UDB for z/OS

We recommend that you stop both WIICEP for IMS, and WIIEP for DB2 UDB for z/OS before proceeding with the configuration of the WIICEP for IMS and WIIEP for DB2 UDB for z/OS for the Trustworthy Insurance CIF and Event Alert System solution.

Note: With careful planning, it might be possible to defer this activity to a later point to minimize outage of IMS and DB2 for z/OS. We chose to adopt a simpler approach of shutting down both IMS and DB2 for z/OS at this point prior to the commencement of the configuration process.
3.3.5 STEP 5: Configure WebSphere MQ on the z/OS and Windows platforms

Figure 3-6 shows the channels, transmit, send and receive queues used in our scenario.

![Figure 3-6](image)

A brief description of these objects follows:

- **MQ8G** is the queue manager on the z/OS platform.
- **QMCIF** is the queue manager on the Windows 2000 platform.
- **CAC_RESTARTQ** is a local queue associated with MQ8G which stores restart information for WebSphere Information Integrator Classic Event Publisher for IMS.
- **IMSK_SENDQT**, **IMSK_SENDQ1**, **IMSK_SENDQ2** and **IMSK_SENDQ3** are remote queue definitions associated with MQ8G.
  - **IMSK_SENDQT** is only used for testing the WebSphere MQ transport layer.
  - **IMSK_SENDQ1** is used to publish transaction messages to the CIF Updater programs.
  - **IMSK_SENDQ2** is used to publish transaction messages to the Event Notification programs.
  - **IMSK_SENDQ3** is used to publish row operation messages to the Event Notification programs.

**Note:** **IMSK_SENDQ2** was not used due to time constraints.

- **IMSK_SENDQ3** is used to publish row operation messages to the Event Notification programs.
- **ASN.RESTARTQ** is a local queue associated with MQ8G that is used to store restart information for Q Capture in WebSphere Information Integrator Event Publisher for DB2 UDB for z/OS.
- **ASN.ADMINQ** is a local queue associated with MQ8G that is used by the Q Capture program to process messages sent from a Q Apply program.
DB2G_SENDQT, DB2G_SENDQ1, DB2G_SENDQ2 and DB2G_SENDQ3 are remote queue definitions associated with MQ8G.
- DB2G_SENDQT is only used for testing the WebSphere MQ transport layer.
- DB2G_SENDQ1 is used to publish transaction messages to the CIF Updater programs.
- DB2G_SENDQ2 is used to publish transaction messages to the Event Notification programs.
- DB2G_SENDQ3 is used to publish row operation messages to the Event Notification programs.

**Note:** DB2G_SENDQ2 and DB2G_SENDQ3 were not used due to time constraints.

QMCIF_XMITQ is a shared local transmit queue associated with MQ8G, where all messages sent to queue manager QMCIF from MQ8G are persisted.

IMSK_RECVQT, IMSK_RECVQ1, IMSK_RECVQ2 and IMSK_RECVQ3 are remote queue definitions associated with QMCIF.
- IMSK_RECVQT is only used for testing the WebSphere MQ transport layer.
- IMSK_RECVQ1 is used to consume transaction messages for the CIF Updater programs.
- IMSK_RECVQ2 is used to consume transaction messages for the Event Notification programs.
- IMSK_RECVQ3 is used to consume row operation messages for the Event Notification programs.

**Note:** IMSK_RECVQ2 was not used due to time constraints.

DB2G_RECVQ1, DB2G_RECVQ2 and DB2G_RECVQ3 are remote queue definitions associated with QMCIF.
- DB2G_RECVQT is only used for testing the WebSphere MQ transport layer.
- DB2G_RECVQ1 is used to consume transaction messages for the CIF Updater programs.
- DB2G_RECVQ2 is used to consume transaction messages for the Event Notification programs.
- DB2G_RECVQ3 is used to consume row operation messages for the Event Notification programs.

**Note:** DB2G_RECVQ2 and DB2G_RECVQ3 were not used due to time constraints.

MQ8G_XMITQ is a shared local transmit queue associated with QMCIF, where all messages sent to queue manager MQ8G from QMCIF are persisted.

MQ8G_TO_QMCIF and QMCIF_TO_MQ8G are the send and receive channels defined on z/OS and Windows 2000 respectively.

ASN.ADMINQ is a remote queue definition associated with QMCIF that corresponds to ASN.ADMINQ associated with MQ8G that is used by the Q Apply program to communicate with the Q Capture program on the z/OS platform.
The following subsections describe the following topics:

- Creation of the WebSphere MQ objects on z/OS
- Creation of the WebSphere MQ objects on Windows 2000
- Verify successful WebSphere MQ transport layer configuration

**WebSphere MQ objects on z/OS**

The WebSphere MQ objects in z/OS described in Figure 3-6 on page 97 may be created using several different ways. This redbook describes the commands we used to create the objects.

**Note:** The creation of an MQ Manager and Listener is not covered here. For full details, refer to *WebSphere MQ System Setup Guide*, SC34-6052-00.

The creation of the channels and queues was split into two jobs as follows:

- CAC.V8R2M8.REDBOOK(MQBASE) is used to define the base infrastructure objects such as local queues, channels, transmit queue as shown in Example 3-1.
- CAC.V8R2M8.REDBOOK(MQSENDQ) is used to define the send queues as shown in Example 3-2.

**Example 3-1  Job to create local queues, channel and transmit queue**

```plaintext
//MQBASE JOB (POK,999),MSGLEVEL=(1,1),MSGCLASS=H,
  //    CLASS=A,NOTIFY=&SYSUID
  //*******************************************************************************
  //* JOB TO CREATE BASE OBJECTS FOR MQ TRANSPORT CHANNELS 
  //* FROM QUEUE MANAGER MQ8G ON Z/OS IMAGE SC53
  //* TO QUEUE MANAGER QMCIF ON WINDOWS 2000 AT TONGA.ALMADEN.IBM.COM 
  //* 
  //* CAC_RESTARTQ IS THE RESTART QUEUE FOR CLASSIC EVENT PUBLISHER 
  //* QMCIF_XMITQ IS THE TRANSMIT QUEUE TO TONGA 
  //* QM8G_TO_QMCIF IS THE SEND QUEUE TO TONGA 
  //* QMCIF_TO_QM8G IS THE RECEIVE QUEUE FROM TONGA (CURRENTLY UNUSED) 
  //********************************************************************************

/*JOBPARM SYSAFF=SC53
//STEP1 EXEC PGM=CSQUTIL,PARM='MQ8G'
//STELIB DD DSN=MQ531.SCSQANLE,DISP=SHR
//        DD DSN=MQ531.SCSQAUTH,DISP=SHR
//OUTPUT  DD SYSOUT=* 
//SYSPRINT DD SYSOUT=* 
//SYSIN DD *
COMMAND DDNAME(INPUT) MAKEDEF(OUTPUT)
/*
//INPUT DD *

DEFINE QLOCAL(CAC_RESTARTQ) +
REPLACE +
DESCR('LOCAL ADMIN QUEUE') +
PUT(ENABLED) +
GET(ENABLED) +
INDXTYPE(GROUPID) +
DEFSOPT(SHARED) +
DEFPSIST(YES)

DEFINE QLOCAL(QMCIF_XMITQ) +
REPLACE +
DESCR('TRANSMISSION QUEUE TO QMCIF') +
USAGE(XMITQ) +
```
Example 3-2  Job to create send queues

```sql
//MOSENDQ JOB (POK,999),MSGLEVEL=(1,1),MSGCLASS=H,
//    CLASS=A,NOTIFY=&SYSUID
//******************************************************************************
// JOB TO CREATE SEND QUEUES FOR EVENT PUBLISHING
// FROM QUEUE MANAGER MQ8G ON Z/OS IMAGE SC53
// TO QUEUE MANAGER QMCIF ON WINDOWS 2000 AT TONGA.ALMADEN.IBM.COM
//******************************************************************************
//*/
/*JOBPARM SYSAFF=SC53
//STEP1 EXEC PGM=CSQUTIL,PARM='MQ8G'
//STEPLIB DD DSN=MQ531.SCSQANLE,DISP=SHR
// DD DSN=MQ531.SCSQAUTH,DISP=SHR
//OUTPUT DD SYSOUT=* 
//SYSPRINT DD SYSOUT=* 
//SYSIN DD *
COMMAND DDNAME(INPUT) MAKEDEF(OUTPUT) */
/*
//INPUT DD *
DEFINE QREMOTE(IMSK_SENDQT) -
REPLACE -
DESCR('REMOTE DEFN OF IMSK_RECVQT ON QMCIF') -
PUT(ENABLED) -
XMITQ(QMCIF_XMITQ) -
RNAME(IMSK_RECVQT) -
RQMNAME(QMCIF) -
DEFPSIST(YES)

DEFINE QREMOTE(IMSK_SENDQ1) -
REPLACE -
DESCR('REMOTE DEFN OF IMSK_RECVQ1 ON QMCIF') -
PUT(ENABLED) -
```

XMITQ(QMCIF_XMITQ) -
RNAME(IMSK_RECVQ1) -
RQMNAME(QMCIF) -
DEFPSIST(YES)

DEFINE QREMOTE(IMSK_SENDQ2) -
REPLACE -
DESCR('REMOTE DEFN OF IMSK_RECVQ2 ON QMCIF') -
PUT(ENABLED) -
XMITQ(QMCIF_XMITQ) -
RNAME(IMSK_RECVQ2) -
RQMNAME(QMCIF) -
DEFPSIST(YES)

DEFINE QREMOTE(IMSK_SENDQ3) -
REPLACE -
DESCR('REMOTE DEFN OF IMSK_RECVQ3 ON QMCIF') -
PUT(ENABLED) -
XMITQ(QMCIF_XMITQ) -
RNAME(IMSK_RECVQ3) -
RQMNAME(QMCIF) -
DEFPSIST(YES)

DEFINE QREMOTE(DB2G_SENDQT) -
REPLACE -
DESCR('REMOTE DEFN OF DB2G_RECVQT ON QMCIF') -
PUT(ENABLED) -
XMITQ(QMCIF_XMITQ) -
RNAME(DB2G_RECVQT) -
RQMNAME(QMCIF) -
DEFPSIST(YES)

DEFINE QREMOTE(DB2G_SENDQ1) -
REPLACE -
DESCR('REMOTE DEFN OF DB2G_RECVQ1 ON QMCIF') -
PUT(ENABLED) -
XMITQ(QMCIF_XMITQ) -
RNAME(DB2G_RECVQ1) -
RQMNAME(QMCIF) -
DEFPSIST(YES)

DEFINE QREMOTE(DB2G_SENDQ2) -
REPLACE -
DESCR('REMOTE DEFN OF DB2G_RECVQ2 ON QMCIF') -
PUT(ENABLED) -
XMITQ(QMCIF_XMITQ) -
RNAME(DB2G_RECVQ2) -
RQMNAME(QMCIF) -
DEFPSIST(YES)

DEFINE QREMOTE(DB2G_SENDQ3) -
REPLACE -
DESCR('REMOTE DEFN OF DB2G_RECVQ3 ON QMCIF') -
PUT(ENABLED) -
XMITQ(QMCIF_XMITQ) -
RNAME(DB2G_RECVQ3) -
RQMNAME(QMCIF) -
DEFPSIST(YES)

/*
WebSphere MQ objects on Windows 2000

The WebSphere MQ objects in Windows 2000 described in Figure 3-6 on page 97 may be created using several different ways.

**Note:** The creation of an MQ Manager and Listener is not covered here. For full details, refer to *WebSphere MQ System Administration Guide*, SC34-6068-02.

We created these channels and queues by submitting a WebSphere MQ command from a DOS command prompt from a directory where QMCIF.txt exists as follows:

```
runmqsc QMCIF<QMCIF.txt
```

Where QMCIF.txt contains the commands for creating the objects as shown in Example 3-3. This command submits the script through the WebSphere MQ command processor resulting in the output shown in Example 3-4.

**Example 3-3  Script file for creating WebSphere MQ objects**

```
DEFINE QLOCAL(IMSK_RECVQT) +
REPLACE +
DESCR('LOCAL RECVQ QUEUE') +
PUT(ENABLED) +
GET(ENABLED) +
DEFSOPT(SHARED) +
DEFSYST(YES)

DEFINE QLOCAL(IMSK_RECVQ1) +
REPLACE +
DESCR('LOCAL RECVQ QUEUE') +
PUT(ENABLED) +
GET(ENABLED) +
DEFSOPT(SHARED) +
DEFSYST(YES)

DEFINE QLOCAL(IMSK_RECVQ2) +
REPLACE +
DESCR('LOCAL RECVQ QUEUE') +
PUT(ENABLED) +
GET(ENABLED) +
DEFSOPT(SHARED) +
DEFSYST(YES)

DEFINE QLOCAL(IMSK_RECVQ3) +
REPLACE +
DESCR('LOCAL RECVQ QUEUE') +
PUT(ENABLED) +
GET(ENABLED) +
DEFSOPT(SHARED) +
DEFSYST(YES)

DEFINE QLOCAL(DB2G_RECVQT) +
REPLACE +
DESCR('LOCAL RECVQ QUEUE') +
PUT(ENABLED) +
GET(ENABLED) +
DEFSOPT(SHARED) +
DEFSYST(YES)

DEFINE QLOCAL(DB2G_RECVQ1) +
```
Example 3-4   QMCIF.txt script output

13 MQSC commands read
No commands have asyntax error
All valid MQSC commands were processed
WebSphere MQ Explorer can be used to view the WebSphere MQ queues and channels just created as shown in Figure 3-7 and Figure 3-8 respectively.

![WebSphere MQ queues created](image1.png)

**Figure 3-7  WebSphere MQ queues created**

![WebSphere MQ channels](image2.png)

**Figure 3-8  WebSphere MQ channels**

**Note:** Channels need to be started on both the z/OS and Windows 2000 platforms since they are inactive after creation.

WebSphere MQ need to be started on the z/OS and Windows 2000 platform, as follows:

- **z/OS:**
  - The queue manager and channels can be started using the following commands from SDSF:
    - `/=MQ8G START QMGR`
    - `/=MQ8G START CHINIT`
  - The status of the queue manager and channels can be viewed using WebSphere MQ administration panels.
Windows 2000:
The queue manager and channels can be started using the following commands from a
DOS command prompt:

```
strmqm QMCIF
runmqsc QMCIF=startchannelqmcif.txt
```

where the contents of the `startchannelqmcif.txt` file is as follows:

```
start channel (QMCIF_TO_MQ8G)
```

The status of the channels can be checked by command — but the easier method is to
use WebSphere MQ Explorer as shown in Figure 3-9.

![WebSphere MQ channels status](image)

**Figure 3-9  WebSphere MQ channels status**

**Verify successful WebSphere MQ transport layer configuration**

After successfully defining all the WebSphere MQ objects and starting the channels, we
strongly recommend verifying the operation of the queues and channels. WebSphere MQ
provides simple “PUT” and “GET” programs to test the queues and channels on all platforms.

- On Windows, these programs are usually called `amqsput` and `amqsget`, and are included
  as source and executable code in the WebSphere MQ libraries.
- On z/OS, these programs are called `CSQ4BVK1` and `CSQ4BVJ1`.

To test the send channels from z/OS to Windows 2000, perform the following actions:
1. Invoke the `amqsput` program on z/OS to place messages on all the send queues
2. Invoke the `amqsget` program on Windows to retrieve them.

The `amqsput` program can be invoked by JCL as shown in Example 3-5 to put three
25-character messages on each of the send queues on z/OS. Once executed, the messages
are sent to the corresponding receive queues on QMCIF where they can be read.

**Example 3-5  Invoking amqsput on z/OS**

```
//CSQ4BVJR JOB 'ACCN#',REGION=4M,NOTIFY=&SYSUID,MSGLEVEL=(1,1)
//******************************************************************
//*                                                                *
//* @START_COPYRIGHT@                                              *
//* Statement: Licensed Materials - Property of IBM                 *
//*                                                             *
//* 5655-F10                                                   *
//* (C) Copyright IBM Corporation. 1993, 2002                  *
//*                                                             *
//* Status: Version 5 Release 3                                 *
//* @END_COPYRIGHT@                                              *
//******************************************************************
//*               IBM WebSphere MQ for z/OS                        *
```
/*
/* THIS JOB RUNS THE PUT/GET SAMPLE PROGRAMS, CSQ4BVK1/CSQ4BVJ1
/*
/* PROGRAM CSQ4BVK1 ISSUES MQPUT ON A QUEUE.
/* - FIRST PARM (++QMGR++) QUEUE MANAGER NAME
/* - SECOND PARM (++QUEUE++) QUEUE NAME
/* - THIRD PARM (++MSGS++) THE NUMBER OF MESSAGES TO PUT-(9999)
/* - FOURTH PARM (++PAD++) THE PADDING CHARACTER
/* - FIFTH PARM (++LEN++) THE LENGTH OF EACH MESSAGE-(9999)
/* - SIXTH PARM (++PERS++) (P)ERSISTENT/(N)ON PERSISTENT MESSAGES
/* MESSAGES ARE PRINTED TO DD SYSPRINT
/*
/* **************************************************************************
/* CUSTOMIZE THIS JCL HERE FOR YOUR INSTALLATION
/* YOU MUST DO GLOBAL CHANGES ON THESE PARAMETERS USING YOUR EDITOR
/*
/* Replace
/* ++THLQUAL++ - The high level qualifier of the
/* WebSphere MQ target library data sets.
/* ++LANGLETTER++ - The language suffix letter:
/* - C - Simplified Chinese
/* - E - US English (Mixed case)
/* - K - Japanese
/* - U - US English (Uppercase)
/*
/* If you create your own CSQ4BVK1/CSQ4BVJ1 load modules, replace the
/* DD statement
//JOBLIB   DD   DSN=MQ531.SCSQLOAD,DISP=SHR
//         DD   DSN=MQ531.SCSQANLE,DISP=SHR
//         DD   DSN=MQ531.SCSQAUTH,DISP=SHR
//*
/* PUTMSGS  EXEC PGM=CSQ4BVK1,REGION=1024K,
/* PARM=('++QMGR++,++QUEUE++,++MSGS++,++PAD++,++LEN++,++PERS++')
/* PUTMSGS  EXEC PGM=CSQ4BVK1,REGION=1024K,
/* PARM=('MQ8G,IMSK_SENDQT,3,W,25,N')
/* PUTMSGS  EXEC PGM=CSQ4BVK1,REGION=1024K,
/* PARM=('MQ8G,IMSK_SENDQ1,3,W,25,N')
/* PUTMSGS  EXEC PGM=CSQ4BVK1,REGION=1024K,
/* PARM=('MQ8G,IMSK_SENDQ2,3,W,25,N')
/* PUTMSGS  EXEC PGM=CSQ4BVK1,REGION=1024K,
/* PARM=('MQ8G,IMSK_SENDQ3,3,W,25,N')
/* PUTMSGS  EXEC PGM=CSQ4BVK1,REGION=1024K,
/* PARM=('MQ8G,DB2G_SENDQT,3,W,25,N')
/* PUTMSGS  EXEC PGM=CSQ4BVK1,REGION=1024K,
/* PARM=('MQ8G,DB2G_SENDQ1,3,W,25,N')
/* PUTMSGS  EXEC PGM=CSQ4BVK1,REGION=1024K,
The script shown in Example 3-6 produces output as shown in Example 3-7. Successful reading of these test messages confirms the implementation of the WebSphere MQ transport layer.

Example 3-6 amqsget script

```
amqsget IMSK_RECVQT QMCIF
amqsget IMSK_RECVQ1 QMCIF
amqsget IMSK_RECVQ2 QMCIF
amqsget IMSK_RECVQ3 QMCIF
amqsget DB2G_RECVQT QMCIF
amqsget DB2G_RECVQ1 QMCIF
amqsget DB2G_RECVQ2 QMCIF
amqsget DB2G_RECVQ3 QMCIF
```

Example 3-7 Sample amqsget output

```
C:\EPCODE\>amqsget IMSK_RECVQT QMCIF
Sample AMQSGETO start
message <WWWWWWWWWWWWWWWWWWWWWWWWW>
message <WWWWWWWWWWWWWWWWWWWWWWWWW>
message <WWWWWWWWWWWWWWWWWWWWWWWWW>
no more messages
Sample AMQSGETO end

C:\EPCODE\>amqsget IMSK_RECVQ1 QMCIF
Sample AMQSGETO start
message <WWWWWWWWWWWWWWWWWWWWWWWWW>
message <WWWWWWWWWWWWWWWWWWWWWWWWW>
message <WWWWWWWWWWWWWWWWWWWWWWWWW>
message <WWWWWWWWWWWWWWWWWWWWWWWWW>
.....
```

3.3.6 STEP 6A: Configure WIICEP for IMS

This section provides a step-by-step guide to configuring WebSphere Information Integrator Classic Event Publisher for IMS for the CIF and Event Alert System solution. The main steps are shown in Figure 3-10, and described in more detail in the following subsections.

Attention: Steps 6A1 and 6A2 may be skipped if you can hand-code the USE Grammar for creating the metadata, but we do not recommend it.
STEP 6A1: Download and install the Data Mapper client on Windows

This process is described in detail in “STEP 3a: Download and install the Data Mapper client on Windows” on page 12.

STEP 6A2: Use Data Mapper to generate USE Grammar for AUTODB

As described in “STEP 3b: Use Data Mapper to generate USE Grammar” on page 14, the Data Mapper tool assists in generating “logical” table definitions for the metadata catalog — the “logical” table definitions are referred to as USE Grammar because they define how the underlying hierarchical database will be used as a “logical” relational table for change capture.

In this section, we generate the USE Grammar for the AUTODB IMS database used by the auto insurance application. The AUTODB DBD is shown in Example 3-8, while the corresponding IMS SEGMENT COBOL copybooks are shown in Example 3-9. The PSB used by the auto insurance application is shown in Example 3-10.

Example 3-8  AUTODB DBD

<table>
<thead>
<tr>
<th>DBD</th>
<th>NAME=AUTODB,ACCESS=(HISAM,VSAM)</th>
<th>00160002</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATASET</td>
<td>DD1=AUTODB,DEVICE=3380,OVFLW=AUTOPOB,</td>
<td>X00170002</td>
</tr>
<tr>
<td></td>
<td>SIZE=(2048,2048),RECORD=(2000,2000)</td>
<td>00180003</td>
</tr>
<tr>
<td>SEGMENT</td>
<td>NAME=CUSTOMER,PARENT=0,BYTES=75,FREQ=250</td>
<td>00190002</td>
</tr>
<tr>
<td>FIELD</td>
<td>NAME=(CUSTKEY,SEQ),TYPE=C,BYTES=10,START=1</td>
<td>00200002</td>
</tr>
<tr>
<td>SEGMENT</td>
<td>NAME=ADDRESS,PARENT=CUSTOMER,BYTES=100,FREQ=3</td>
<td>00210002</td>
</tr>
<tr>
<td>FIELD</td>
<td>NAME=(ADDRTYPE,SEQ),TYPE=C,BYTES=1,START=1</td>
<td>00220002</td>
</tr>
<tr>
<td>SEGMENT</td>
<td>NAME=POLICY,PARENT=CUSTOMER,BYTES=250,FREQ=2</td>
<td>00230002</td>
</tr>
<tr>
<td>FIELD</td>
<td>NAME=(POLICYNM,SEQ),TYPE=C,BYTES=10,START=1</td>
<td>00240002</td>
</tr>
<tr>
<td>SEGMENT</td>
<td>NAME=CLAIMS,PARENT=POLICY,BYTES=250,FREQ=3</td>
<td>00250002</td>
</tr>
<tr>
<td>FIELD</td>
<td>NAME=(CLAIMNM,SEQ),TYPE=C,BYTES=10,START=1</td>
<td>00260002</td>
</tr>
<tr>
<td>SEGMENT</td>
<td>NAME=PREMIUMS,PARENT=POLICY,BYTES=30,FREQ=12</td>
<td>00270003</td>
</tr>
</tbody>
</table>
Example 3-9  AUTODB IMS SEGMENT COBOL copybooks

01 CUSTOMER.
  05 CUSTKEY           PIC X(10).
  05 FIRSTNME         PIC X(20).
  05 LASTNAME          PIC X(25).
  05 BIRTHDTE         PIC X(8).

01 ADDRESS.
  05 ADDRTYPE         PIC X.
  05 ADDRESS1        PIC X(25).
  05 ADDRESS2        PIC X(25).
  05 ADDRESS3        PIC X(25).
  05 ZIPCODE          PIC X(10).

01 POLICY.
  05 POLICYNM        PIC X(10).
  05 POLYTYPE        PIC X(3).
  05 POLCOVER       PIC 9(6).
  05 POLSTDTE       PIC X(8).
  05 POLNOTES       PIC X(200).

01 PREMIUMS.
  05 RISKLINM        PIC X(2).
  05 RPREMIUM       PIC 9(4)V99.

Example 3-10  AUTOPSB1

TITLE 'AUTODB PSB' 00100000
* 00010000
******************************************************************************@SCPYRT** 00030000
* 00031000
* Licensed Materials - Property of IBM 00032000
* 00033000
* Restricted Materials of IBM 00034000
* 00035000
* 5655-C56 00036000
* 00037000
* (C) Copyright IBM Corp. 1989,1998 00070000
* 00100000
******************************************************************************@ECPYRT** 00140000
* 00150000
PCB TYPE=DB,DBDNAME=AUTODB,PROCOPT=AP,KEYLEN=30,SB=COND 00170001
SENSEG NAME=CUSTOMER,PARENT=0 00180000
SENSEG NAME=ADDRESS,PARENT=CUSTOMER 00190000
SENSEG NAME=POLICY,PARENT=CUSTOMER 00200000
SENSEG NAME=CLAIMS,PARENT=POLICY 00210000
SENSEG NAME=PREMIUMS,PARENT=POLICY 00220000

Chapter 3. Trustworthy Insurance Customer Information Facility (CIF) and Event Alert System scenario  109
As described in “STEP 3b: Use Data Mapper to generate USE Grammar” on page 14, creating USE Grammar for the AUTODB IMS database involves the following steps:

1. Create a new repository.
2. Load the AUTODB DBD file.
3. Create Schema and IMS “logical” tables.
4. Generate USE Grammar.

Each of these steps is described in the following subsections.

**Create a new repository**

Start the Data Mapper Tool by clicking Start → Programs → IBM DB2 Information Integrator Classic Tools → DataMapper, and then select File from the action bar, and click New Repository as shown in Figure 3-11. Name the repository “autodb” which creates a file called autodb.mdb to store all the metadata mappings as shown in Figure 3-12.

![Figure 3-11 Create a new repository](image1)

![Figure 3-12 DataMapper autodb repository](image2)
With the autodb.mdb repository open, select **Edit** from the action bar, and select **Create a New Data Catalog** as shown in Figure 3-13 to proceed to Figure 3-14.

![Create a Data Catalog 1/2](image1.png)

**Figure 3-13 Create a Data Catalog 1/2**

![Create Data Catalog](image2.png)

**Figure 3-14 Create a Data Catalog 2/2**

In Figure 3-14, supply the name (AUTODB) of the data catalog, select IMS from the dropdown list for Type, and check the Change Capture box. Click **OK** to complete the definition as shown in Figure 3-15.

![AUTODB data catalog](image3.png)

**Figure 3-15 AUTODB data catalog**
**Load the AUTODB DBD file**

Highlight the AUTODB data catalog in Figure 3-15, click **File** in the action bar, and select “Load a DBD file for Reference” from the dropdown menu as shown in Figure 3-16.

![Figure 3-16 Load DBD file 1/5](image)

The DBD file may be loaded in one of two ways as follows:

- If the DBD file was downloaded previously to the Windows machine using an FTP program, you can browse the local hard drive or a network hard drive to select the DBD file.

- You can FTP the DBD file directly from the z/OS dataset that it is stored in, by clicking the **Remote** button, as shown in Figure 3-17 to proceed to Figure 3-18.

![Figure 3-17 Load DBD file 2/5](image)
Chapter 3. Trustworthy Insurance Customer Information Facility (CIF) and Event Alert System scenario

In Figure 3-18, provide the TCP/IP address of the z/OS system (9.1.6.77), PortID (21), and the TSO UserID (NEALE) and User Password (**), and click Connect to proceed to Figure 3-19.
In Figure 3-19, provide values for the Working Directory (‘NALUR.DATA’), Remote File (‘NALUR.DATA(AUTODB)’) and hit Enter to list the members in the data set as shown. Select member AUTODB, and click Transfer to initiate the file transfer.

When the FTP completes, a new panel is displayed within DataMapper showing the segments in the AUTODB DBD that was successfully downloaded as shown in Figure 3-20.

![Data Mapper](image)

**Figure 3-20  Load DBD file 5/5**

**Create Schema and IMS “logical” tables**

This step defines the “logical” tables that will be used for event publishing.

These “logical” tables are usually based on the physical IMS Segments (plus concatenated key values), because a “logical” table only generates published events for changes to a physical segment. Therefore, a join of a parent and child segment would only publish changes for the child segment. In order to capture changes from both segments, you must define two “logical” tables for event publishing, and the program that consumes the published events must process changes from both “logical” tables. This approach is used in the Trustworthy Insurance CIF scenario to capture changes from both the CUSTOMER and ADDRESS IMS segments, but apply these changes to a single target table that combines customer and address details.

The CIF solution requires five “logical” tables to be defined — each “logical” table is based on an IMS physical segment, and will feed a publication as shown in Figure 3-21.
The Event Alert System (EAS) solution requires one “logical” table to be defined named QUIKCLM which is based on the CLAIM IMS physical segment in the auto insurance IMS database and feeds the row operation message publication as shown in Figure 3-21.

**Figure 3-21  Relationship between IMS physical segments and “logical” tables for CIF solution**

**Figure 3-22  Relationship between IMS physical segments and “logical” tables for EAS solution**
A WIICEP for IMS publication is associated with one physical IMS segment. However, by associating all five publications with the same physical WebSphere MQ queue, and choosing the publication mode of all five publication as transactional, WIICEP for IMS ensures that all changes occurring in a single unit-of-work are written in a single XML message. For example, the sale of a new insurance policy involves inserts to four of the five IMS segments (excluding CLAIMS which may happen at a later point in time), which corresponds to four insert operations (CUSTOMER, ADDRESS, POLICY and PREMIUM) included in a single XML message. Figure 3-21 shows three XML messages — corresponding to a new policy (minus claim information), a new claim and a renewal of an existing policy.

**Important:** Changes to a leaf segment drive changes to that particular “logical” table. However, the “logical” table may contain fields from other parent segments (if parent segments exist). In the case of the root segment, the only fields that will be captured are those from the root segment itself.

For example, if the CLAIMS “logical” table is created from the CLAIMS IMS segment and the concatenated keys and other fields from the POLICY and CUSTOMER IMS segments, then the “logical” table CLAIMS is considered to have changed only when modifications occur to fields in the CLAIMS IMS segment, and not when changes occur to fields in the POLICY or CUSTOMER IMS segments that are included in the “logical” table.

We describe the definition of five “logical” tables CUSTOMER, ADDRESS, POLICY, CLAIMS and PREMIUMS in the following sections.

- Create “logical” table CUSTOMER:
  
  As shown in Figure 3-23, select **Windows → List Tables** from the action bar to proceed to Figure 3-24 which shows an empty list since no tables have as yet been defined for AUTODBD.

![Figure 3-23 List Tables 1/2](image)
Prior to creating any tables, a schema (owner) must be defined for the tables. To determine the list of available schemas (owners), select Windows $\rightarrow$ List Owners from the action bar as shown in select Windows $\rightarrow$ List Tables to proceed to Figure 3-26.

Figure 3-24  List Tables 2/2

Figure 3-25  List Owners 1/2
A default owner ‘SYS’ is provided as shown in Figure 3-26. Since we want to define the “logical” tables under the schema name of IMSKAUTO, we need to create the IMSKAUTO schema first.

Select Edit → Create a New Owner from the action bar in Figure 3-27 to proceed to a dialog box shown in Figure 3-28. Supply the Name (IMSKAUTO) and Remarks (Schema for AUTODB tables in IMSK) in the dialog box and click OK to create the new schema as shown in Figure 3-29.
We can now begin creating the CUSTOMER "logical" table as follows. Select Edit → Create a New Table from the action bar shown in Figure 3-30 to proceed to Figure 3-31.
Supply the appropriate values for the CUSTOMER "logical" table, and click OK which shows the new CUSTOMER table added in Figure 3-32.
After the table has been added, the columns associated with this table must be defined. This is achieved by downloading COBOL copybooks for the IMS segment(s) and selecting the fields of interest as follows:

Select **Edit → Import External File** from the action bar as shown in Figure 3-33 to proceed to Figure 3-34.

*Figure 3-33  Create “logical” table CUSTOMER 4/9*

*Figure 3-34  Create “logical” table CUSTOMER 5/9*
Using a process similar to that used to download the DBD earlier, transfer the COBOL copybook for the CUSTOMER segment by clicking the **Transfer** button to display Figure 3-35.

![Import Copybook](image)

**Figure 3-35** Create “logical” table CUSTOMER 6/9

Click **Import** in Figure 3-35 to view Figure 3-36, which shows all the imported columns.

![Data Mapper](image)

**Figure 3-36** Create “logical” table CUSTOMER 7/9
You can delete columns that are not required in the “logical” table, modify the Column Name as well as the SQL Datatype. In our scenario, we chose to delete the BIRTHDTE column from the “logical” table as shown in Figure 3-37 to complete the definition of the CUSTOMER table as shown in Figure 3-38.

Figure 3-37  Create “logical” table CUSTOMER 8/9

Figure 3-38  Create “logical” table CUSTOMER 9/9
Create “logical” table ADDRESS:

The process to create the ADDRESS “logical” table is slightly different because it is derived from the ADDRESS IMS segment which is a dependent segment of the CUSTOMER IMS segment. The ADDRESS "logical" table requires the concatenated key columns of the parent CUSTOMER IMS segment to be added.

Therefore in Figure 3-39, the Index Root field should have CUSTOMER, and both the CUSTOMER and ADDRESS COBOL copybooks must be downloaded an imported, and relevant fields selected for inclusion in the ADDRESS “logical” table as shown in Figure 3-40 through Figure 3-44.

Figure 3-39   Create “logical” table ADDRESS 1/6

Figure 3-40   Create “logical” table ADDRESS 2/6
Chapter 3. Trustworthy Insurance Customer Information Facility (CIF) and Event Alert System scenario

Figure 3-41  Create “logical” table ADDRESS 3/6

Figure 3-42  Create “logical” table ADDRESS 4/6
Figure 3-43  Create "logical" table ADDRESS 5/6

Figure 3-44  Create "logical" table ADDRESS 6/6
Create “logical” table POLICY:
This process is identical to the creation of the ADDRESS “logical” table. Figure 3-45 through Figure 3-49 show the steps involved.

![Create IMS Table](image)

Figure 3-45 Create “logical” table POLICY 1/5

![Import Copybook](image)

Figure 3-46 Create “logical” table POLICY 2/5
Figure 3-47  Create “logical” table POLICY 3/5

Figure 3-48  Create “logical” table POLICY 4/5
Figure 3-49  Create “logical” table POLICY 5/5

- Create “logical” table CLAIMS:
  This process is identical to the creation of the POLICY “logical” table, except that includes copybooks CUSTOMER, POLICY and CLAIMS. Figure 3-50 through Figure 3-54 show the steps involved.

Figure 3-50  Create “logical” table CLAIMS 1/5
Figure 3-51  Create “logical” table CLAIMS 2/5

Figure 3-52  Create “logical” table CLAIMS 3/5
Figure 3-53  Create "logical" table CLAIMS 4/5

Figure 3-54  Create "logical" table CLAIMS 5/5
Create “logical” table PREMIUMS:

This process is identical to the creation of the CLAIMS “logical” table. Figure 3-55 through Figure 3-59 show the steps involved.

Figure 3-55   Create “logical” table PREMIUMS 1/5

Figure 3-56   Create “logical” table PREMIUMS 2/5
Figure 3-57  Create "logical" table PREMIUMS 3/5

Figure 3-58  Create "logical" table PREMIUMS 4/5
Create “logical” table QUIKCLM:

This process is almost identical to the creation of the CLAIMS “logical” table except that the POLSTDTE field has been included from the POLICY parent segment in the path. Figure 3-60 through Figure 3-63 show the steps involved.
Chapter 3. Trustworthy Insurance Customer Information Facility (CIF) and Event Alert System scenario

Figure 3-61  Create “logical” table QUIKCLM 2/4

Figure 3-62  Create “logical” table QUIKCLM 3/4

**Attention:** The step shown in Figure 3-62 is different from previous examples. We previously selected key columns from only the parent segments. In this case we choose an attribute column (POLSTDTE) from the POLICY segment to include with the columns from the CLAIMS segment as shown in Figure 3-63.
Generate USE Grammar

After all the tables have been defined, the USE Grammar need to be generated as follows: Highlight AUTODB data catalog name in AUTODB repository, and select File → Generate USE Statements from the action bar in Figure 3-64 to generate the file on a Windows hard drive, or saved directly to z/OS platform using the Remote option. Example 3-11 shows the generated USE Grammar for the logical tables generated earlier.

Example 3-11  Generate USE Grammar

```sql
/* IMSK AUTODB for Redbook CIF Scenario */
DROP TABLE IMSKAUTO.ADDRESS;
USE TABLE IMSKAUTO.ADDRESS DBTYPE IMS
AUTODB INDEXROOT CUSTOMER ADDRESS
( /* COBOL Name CUSTKEY */
  CUSTKEY SOURCE DEFINITION ENTRY ADDRESS
  DATAMAP OFFSET 0 LENGTH 10 DATATYPE C
  USE AS CHAR(10),
  /* COBOL Name ADDRTYPE */
  ADDRTYPE SOURCE DEFINITION ENTRY ADDRESS
  DATAMAP OFFSET 0 LENGTH 1 DATATYPE C
  USE AS CHAR(1),
  /* COBOL Name ADDRESS1 */
  ADDRESS1 SOURCE DEFINITION ENTRY ADDRESS
  DATAMAP OFFSET 1 LENGTH 25 DATATYPE C
  USE AS CHAR(25),
  /* COBOL Name ADDRESS2 */
  ADDRESS2 SOURCE DEFINITION ENTRY ADDRESS
```
DATAMAP OFFSET 26 LENGTH 25 DATATYPE C
USE AS CHAR(25),
/* COBOL Name ADDRESS3 */
ADDRESS3 SOURCE DEFINITION ENTRY ADDRESS
DATAMAP OFFSET 51 LENGTH 25 DATATYPE C
USE AS CHAR(25),
/* COBOL Name ZIPCODE */
ZIPCODE SOURCE DEFINITION ENTRY ADDRESS
DATAMAP OFFSET 76 LENGTH 10 DATATYPE C
USE AS CHAR(10)
);
ALTER TABLE IMSKAUTO.ADDRESS DATA CAPTURE CHANGES;

DROP TABLE IMSKAUTO.CLAIMS;
USE TABLE IMSKAUTO.CLAIMS DBTYPE IMS
AUTODB INDEXROOT CUSTOMER CLAIMS
(
  /* COBOL Name CUSTKEY */
  CUSTKEY SOURCE DEFINITION ENTRY CLAIMS
  DATAMAP OFFSET 0 LENGTH 10 DATATYPE C
  USE AS CHAR(10),
  /* COBOL Name POLICYNM */
  POLICYNM SOURCE DEFINITION ENTRY CLAIMS
  DATAMAP OFFSET 0 LENGTH 10 DATATYPE C
  USE AS CHAR(10),
  /* COBOL Name CLAIMNM */
  CLAIMNM SOURCE DEFINITION ENTRY CLAIMS
  DATAMAP OFFSET 0 LENGTH 10 DATATYPE C
  USE AS CHAR(10),
  /* COBOL Name CLAIMDTE */
  CLAIMDTE SOURCE DEFINITION ENTRY CLAIMS
  DATAMAP OFFSET 10 LENGTH 8 DATATYPE C
  USE AS CHAR(8),
  /* COBOL Name CLAIMAMT */
  CLAIMAMT SOURCE DEFINITION ENTRY CLAIMS
  DATAMAP OFFSET 18 LENGTH 8 DATATYPE UC /* Zoned Decimal */
  USE AS DECIMAL(8,2),
  /* COBOL Name CLAIMDET */
  CLAIMDET SOURCE DEFINITION ENTRY CLAIMS
  DATAMAP OFFSET 26 LENGTH 200 DATATYPE C
  USE AS CHAR(200)
)
;
ALTER TABLE IMSKAUTO.CLAIMS DATA CAPTURE CHANGES;

DROP TABLE IMSKAUTO.CUSTOMER;
USE TABLE IMSKAUTO.CUSTOMER DBTYPE IMS
AUTODB INDEXROOT CUSTOMER CUSTOMER
(
  /* COBOL Name CUSTKEY */
  CUSTKEY SOURCE DEFINITION ENTRY CUSTOMER
  DATAMAP OFFSET 0 LENGTH 10 DATATYPE C
  USE AS CHAR(10),
  /* COBOL Name FIRSTNME */
  FIRSTNME SOURCE DEFINITION ENTRY CUSTOMER
  DATAMAP OFFSET 10 LENGTH 20 DATATYPE C
  USE AS CHAR(20),
  /* COBOL Name LASTNAME */
  LASTNAME SOURCE DEFINITION ENTRY CUSTOMER
DATAMAP OFFSET 30 LENGTH 25 DATATYPE C  
USE AS CHAR(25)
);
ALTER TABLE IMSKAUTO.CUSTOMER DATA CAPTURE CHANGES;

DROP TABLE IMSKAUTO.POLICY;
USE TABLE IMSKAUTO.POLICY DBTYPE IMS
AUTODB INDEXROOT CUSTOMER POLICY
(
    /* COBOL Name CUSTKEY */
    CUSTKEY SOURCE DEFINITION ENTRY POLICY  
    DATAMAP OFFSET 0 LENGTH 10 DATATYPE C  
    USE AS CHAR(10),
    /* COBOL Name POLICYNM */
    POLICYNM SOURCE DEFINITION ENTRY POLICY  
    DATAMAP OFFSET 0 LENGTH 10 DATATYPE C  
    USE AS CHAR(10),
    /* COBOL Name POLYTYPE */
    POLYTYPE SOURCE DEFINITION ENTRY POLICY  
    DATAMAP OFFSET 10 LENGTH 3 DATATYPE C  
    USE AS CHAR(3),
    /* COBOL Name POLCOVER */
    POLCOVER SOURCE DEFINITION ENTRY POLICY  
    DATAMAP OFFSET 13 LENGTH 6 DATATYPE UC /* Zoned Decimal */  
    USE AS CHAR(6),
    /* COBOL Name POLSTDTE */
    POLSTDTE SOURCE DEFINITION ENTRY POLICY  
    DATAMAP OFFSET 19 LENGTH 8 DATATYPE C  
    USE AS CHAR(8),
    /* COBOL Name POLNOTES */
    POLNOTES SOURCE DEFINITION ENTRY POLICY  
    DATAMAP OFFSET 27 LENGTH 200 DATATYPE C  
    USE AS CHAR(200)
);
ALTER TABLE IMSKAUTO.POLICY DATA CAPTURE CHANGES;

DROP TABLE IMSKAUTO.PREMIUMS;
USE TABLE IMSKAUTO.PREMIUMS DBTYPE IMS
AUTODB INDEXROOT CUSTOMER PREMIUMS
(
    /* COBOL Name CUSTKEY */
    CUSTKEY SOURCE DEFINITION ENTRY PREMIUMS  
    DATAMAP OFFSET 0 LENGTH 10 DATATYPE C  
    USE AS CHAR(10),
    /* COBOL Name POLICYNM */
    POLICYNM SOURCE DEFINITION ENTRY PREMIUMS  
    DATAMAP OFFSET 0 LENGTH 10 DATATYPE C  
    USE AS CHAR(10),
    /* COBOL Name RISKLINM */
    RISKLINM SOURCE DEFINITION ENTRY PREMIUMS  
    DATAMAP OFFSET 0 LENGTH 2 DATATYPE C  
    USE AS CHAR(2),
    /* COBOL Name RPREMIUM */
    RPREMIUM SOURCE DEFINITION ENTRY PREMIUMS  
    DATAMAP OFFSET 2 LENGTH 6 DATATYPE UC /* Zoned Decimal */  
    USE AS DECIMAL(6,2)
);
ALTER TABLE IMSKAUTO.PREMIUMS DATA CAPTURE CHANGES;
DROP TABLE IMSKAUTO.QUICKCLM;
USE TABLE IMSKAUTO.QUICKCLM DBTYPE IMS
AUTODB INDEXROOT CUSTOMER CLAIMS
(
    /* COBOL Name CUSTKEY */
    CUSTKEY SOURCE DEFINITION ENTRY CLAIMS
    DATAMAP OFFSET 0 LENGTH 10 DATATYPE C
    USE AS CHAR(10),
    /* COBOL Name POLICYNM */
    POLICYNM SOURCE DEFINITION ENTRY CLAIMS
    DATAMAP OFFSET 0 LENGTH 10 DATATYPE C
    USE AS CHAR(10),
    /* COBOL Name POLSTDTE */
    POLSTDTE SOURCE DEFINITION ENTRY CLAIMS
    DATAMAP OFFSET 19 LENGTH 8 DATATYPE C
    USE AS CHAR(8),
    /* COBOL Name CLAIMNM */
    CLAIMNM SOURCE DEFINITION ENTRY CLAIMS
    DATAMAP OFFSET 0 LENGTH 10 DATATYPE C
    USE AS CHAR(10),
    /* COBOL Name CLAIMDTE */
    CLAIMDTE SOURCE DEFINITION ENTRY CLAIMS
    DATAMAP OFFSET 10 LENGTH 8 DATATYPE C
    USE AS CHAR(8),
    /* COBOL Name CLAIMAMT */
    CLAIMAMT SOURCE DEFINITION ENTRY CLAIMS
    DATAMAP OFFSET 18 LENGTH 8 DATATYPE UC /* Zoned Decimal */
    USE AS DECIMAL(8,2),
    /* COBOL Name CLAIMDET */
    CLAIMDET SOURCE DEFINITION ENTRY CLAIMS
    DATAMAP OFFSET 26 LENGTH 200 DATATYPE C
    USE AS CHAR(200)
);
ALTER TABLE IMSKAUTO.QUICKCLM DATA CAPTURE CHANGES;

This generated USE Grammar needs to be transferred to the
CAC.V8R2M0.REDBOOK(AUTOUSE) data set on the z/OS platform, where it will be used as
input for generating the metadata catalog in “STEP 6A6: Allocate and populate metadata
catalog with USE Grammar” on page 143.

STEP 6A3: Modify AUTODB IMS DBD to support event publishing

This step prepares the AUTODB IMS DBD for event publishing by specifying the EXIT
keyword on the DBD, performing DBDGEN, PSBGEN and ACBGEN, and then copying the
staging libraries to the active libraries as follows:

1. Specify the ‘EXIT=’ keyword as described in Section , “STEP 3c: Modify IMS DBDs to
support event publishing” on page 14. Example 3-12 shows the modified AUTODB DBD to
support event publishing for the Trustworthy Insurance CIF and Event Alert System
solution. We chose to specify the EXIT keyword on each of the segments in the database.

Example 3-12  Modified AUTODB to support event publishing

<table>
<thead>
<tr>
<th>DBD</th>
<th>NAME=AUTODB,ACCESS=(HISAM,VSAM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATASET</td>
<td>DD1=AUTODB,DEVICE=3380,OVFLW=AUTO80, SIZE=(2048,2048),RECORD=(2000,2000)</td>
</tr>
<tr>
<td>SEG</td>
<td>NAME=CUSTOMER,PARENT=0,BYTES=75,FREQ=250, EXIT=('*,NOKEY,DATA,NOPATH,(NOCASCADE),LOG)</td>
</tr>
<tr>
<td>FIELD</td>
<td>NAME=(CUSTKEY,SEQ),TYPE=C,BYTES=10,START=1</td>
</tr>
</tbody>
</table>
2. Perform DBDGEN.

The DBDGEN job in ‘NALUR.JCL(AUTODBD)’ as shown in Example 3-13 performs this function. Verify that the return code is zero.

Example 3-13  AUTODB DBDGEN job

```
//AUTODBDJ JOB (999,POK),
// 'NALUR',
// CLASS=A,MSGCLASS=H,MSGLEVEL=(1,1),
// NOTIFY=&SYSUID,
// REGION=128M
//*
//*
// * JCLLIB ORDER=(IMS810K.PROCLIB)
/*JOBPARM L=9999,SYSAFF=*  
//*
/* DBDGEN FOR THE AUTODB HISAM/VSAM DATA BASE 
//*
/*AUTODBD1 EXEC PROC=DBDGEN,MBR=AUTODB,SOUT='**'
/C.SYSIN DD DISP=SHR,
            DSN=NALUR.DATA(AUTODB)
//*
/*
```

3. Perform PSBGEN.

The PSBGEN job in ‘NALUR.JCL(AUTOPSB1)’ as shown in Example 3-14 performs this function. Verify that the return code is zero.

Example 3-14  AUTODB PSBGEN job

```
//AUTOPSB1 JOB (999,POK),
// 'NALUR',
// CLASS=A,MSGCLASS=H,MSGLEVEL=(1,1),
// NOTIFY=RC53,
// REGION=128M
//*
//*
// * JCLLIB ORDER=(IMS810K.PROCLIB)
/*JOBPARM L=9999,SYSAFF=*  
//*
//**************************************************************************
//* IVP IMS 8.1
//*
//* SKELETON: DFSIXSE2
//*
```
4. Perform ACBGEN

The ACBGEN job in ‘NALUR.JCL(AUTOACB)’ as shown in Example 3-15 performs this function. Verify that the return code is zero.

Example 3-15  AUTODB ACBGEN job

```sql
//AUTOACBJ JOB (999,POK),
// 'NALUR',
// CLASS=A,MSGCLASS=H,MSGLEVEL=(1,1),
// NOTIFY=&SYSUID,
// REGION=128M
//
// JCLLIB ORDER=(IMS810K.PROCLIB)
/*JOBPARM L=9999,SYSAFF=* */
//
/* IVP IMS 8.1 */
/*
** SKELETON: DFSIXSE3 */
/*
** FUNCTION: PERFORM ACBGEN FOR THE AUTODB APPLICATION */
/*
********************************************************************* */
/* LICENSED MATERIALS - PROPERTY OF IBM */
/*
** RESTRICTED MATERIALS OF IBM */
/*
** 5655-C56 */
/*
** (C) COPYRIGHT IBM CORP. 1989,2002 */
/*
********************************************************************* */
//ACBGEN  EXEC PROC=ACBGEN,SOUT='*',COMP='POSTCOMP'
```
5. Copy staging libraries to active libraries.

The copy job in ‘IMS810K.INSTALIB(IV3E316J)’ as shown in Example 3-16 performs this function. Verify that the return code is zero.

Example 3-16  Copy staging libraries to active libraries

//IV3E316J JOB (999,POK),
// 'NALUR',
// CLASS=A,MSGCLASS=H,MSGLEVEL=(1,1),
// NOTIFY=&SYSUID,
// REGION=128M
//*
//* JCLLIB ORDER=(IMS810K.PROCLIB)
/*JOBPARM L=9999,SYSAFF=* 
/*
﻿//*********************************************************************
//* IVP IMS 8.1
//*
//* SKELETON: DFSIXSEB
//*
//* FUNCTION: COPY STAGING LIBRARIES TO ACTIVE LIBRARIES
//*********************************************************************
//*
//************************************************************@SCPYRT**
//* LICENSED MATERIALS - PROPERTY OF IBM
//* RESTRICTED MATERIALS OF IBM
//* 5655-C56
//* (C) COPYRIGHT IBM CORP. 1989,2002
//************************************************************@ECPYRT**
//* COPY MODBLKS TO MODBLKSA
//*/ 
//MODBLKS EXEC PROC=OLCUTL,SOUT='*',TYPE=MODBLKS,IN=S,OUT=A 
//* 
//* COPY MATRIX  TO MATRIXA
//*/ 
//MATRIX EXEC PROC=OLCUTL,SOUT='*',TYPE=MATRIX,IN=S,OUT=A
//*
//* COPY ACBLIB  TO ACBLIBA
//*/ 
//ACBLIB EXEC PROC=OLCUTL,SOUT='*',TYPE=ACB,IN=S,OUT=A
//*
//* COPY FORMAT  TO FORMATA
//*/ 
//FORMAT EXEC PROC=OLCUTL,SOUT='*',TYPE=FORMAT,IN=S,OUT=A
//************************************************************@ECPYRT**
**STEP 6A4: Install WIICEP Active CCA IMSK**

“STEP 3d: Install WIICEP for IMS Active CCA” on page 17 describes the two methods available to install an active CCA for the CIF and Event Alert System solution.

For our simple scenario, we chose the simpler method of copying the CAC.V8R2M0.SCACLOAD(DFSFLGX0) module directly into IMS810K.SDFSRESL.

**STEP 6A5: Allocate recovery data set and update IMS System Proc**

“STEP 3e: Allocate recovery data set and update IMS System Proc” on page 18 describes the considerations in allocating the optional recovery data set and updating the IMS System Proc. We chose to define one with the attributes shown in Example 3-17, and added a DD card to the IMS started task JCL in SYS1.PROCLIB(IMS810K) as shown in Example 3-18.

*Example 3-17  Recovery data set allocation information*

<table>
<thead>
<tr>
<th>Command ==&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Set Name . . . : CAC.RECOVDS.IMS810K</td>
</tr>
<tr>
<td>General Data</td>
</tr>
<tr>
<td>Volume serial . . . : TSTO10</td>
</tr>
<tr>
<td>Device type . . . . : 3390</td>
</tr>
<tr>
<td>Organization . . . : PS</td>
</tr>
<tr>
<td>Record format . . . : FBS</td>
</tr>
<tr>
<td>Record length . . . : 80</td>
</tr>
<tr>
<td>Block size . . . . : 5120</td>
</tr>
<tr>
<td>1st extent cylinders: 1</td>
</tr>
<tr>
<td>Secondary cylinders : 1</td>
</tr>
<tr>
<td>Creation date . . . : 2005/07/22</td>
</tr>
<tr>
<td>Referenced date . . : 2005/07/25</td>
</tr>
<tr>
<td>Expiration date . . : <em><strong>None</strong></em></td>
</tr>
</tbody>
</table>

*Example 3-18  DD card to add to SYS1.PROCLIB(IMS810K) IMS started task JCL*

```plaintext
//* ADDED CCARECOVERY CARD NAGRAJ
//CACRCV DD DISP=SHR,DSN= CAC.RECOVDS.IMS810K
```

**STEP 6A6: Allocate and populate metadata catalog with USE Grammar**

“STEP 3f: Allocate and populate metadata catalog with USE Grammar” on page 19 describes the steps involved in allocating the metadata catalog and populating it.

Example 3-19 shows the CAC.V8R2M0.SCACSAMP(CACCATLG) job for allocating the metadata catalog which reflects the appropriate references — these are highlighted in Example 3-19. Verify that the return code is zero.

*Example 3-19  Allocate metadata catalog for the CIF & Event Alert System solution*

```plaintext
//CACCATLG JOB 'ACCNT#',REGION=4M,NOTIFY=&SYSUID,MSGLEVEL=(1,1)
//********************************************************************
//*                                                                  *
//*   CACCATLG - SAMPLE JCL TO ALLOCATE META DATA CATALOGS           *
//*                                                                  *
//********************************************************************
//*ALLOC1   PROC CAC='CAC',         INSTALLED HIGH LEVEL QUALIFIER
//*                  DISKU=SYSDA,        DASD UNIT
//*                  DISKVOL=VOLSER  DASD VOLSER
```
This metadata catalog is then populated with the USE Grammar generated in “Generate USE Grammar” on page 136 using CAC.V8R2M0.SCACSAMP(CACMETAU), which reflects the appropriate references — these are highlighted in Example 3-20.

In this case a return code of 4 is expected because the USE Grammar of Example 3-11 on page 136 includes a DROP statement for the objects about to be created. If this is the first time the job has been run, the USE statements for the tables will return an SQLCODE -204, because the objects do not exist. Ensure that the USE TABLE and ALTER TABLE statements succeed with SQLCODE = 0.

Example 3-20 Populate metadata catalog for the CIF & Event Alert System solution
STEP 6A7: Allocate GDG data sets for the correlation service

“STEP 3g: Allocate GDG data sets to provide restart information for CS” on page 20 describes the allocation of the correlation service restart data set.

Example 3-21 shows the CAC.V8R2M0.REDBOOK(ALLOCGDG) job (which is a modified version of CAC.V8R2M0.SCACSAMP(CACGDGA)) for allocating the GDG data sets for the correlation service which reflects the appropriate references — these are highlighted in Example 3-21. Verify that the return code is zero.

Example 3-21   Allocate GDG data sets for the correlation service

//CACGDGA JOB 'ACNT#',REGION=4M,NOTIFY=&SYSUID,MSGLEVEL=(1,1)
//********************************************************************
//*                                                                  *
//*   NAME:  CACGDGA                                                *
//*                                                                  * 

"*/     JCL TO CREATE GDG DATASETS FOR RECOVERY AND ALLOCATES        *
"*/     THE FIRST GENERATION FOR USE BY THE CORRELATION SERVER       *
"*/     BE SURE TO CHANGE THE SYMBOLIC VARIABLES AND THE            *
"*/     LOWERCASE hlq TO MATCH THE FULLY QUALIFIED DATASET NAMES    *
"*/     BEFORE SUBMITTING THIS JOB.                                *
"*/                                                           *
"*/ ITSO NOTE:                                                   *
"*/ FOR NON-SMS GENERATION DATASETS                              *
"*/ THE DSCB MODEL HAS TO BE ON THE SAME VOLUME AS THE USER        *
"*/ CATALOG THAT CONTAINS THE CATALOG ALIAS FOR THE CAC           *
"*/ HIGH LEVEL QUALIFIER.                                        *
"*/                                                           *
"*/ THE GDGs THEMSELVES CAN BE ON ANY VOLUME                      *
"*/                                                           *
"*/                                                           *
"*/******************************************************************
//CACGDGA PROC CAC='CAC',                                     *
//      DISKU=SYSDA,                                          *
//      DISK1=TOTCAT,                                         *
//      DISK2=TST026                                          *
/*
GDGDEF EXEC PGM=IDCAMS                                      *
//SYSPRINT DD SYSOUT=**                                      *
/*
/* MODEL DSCB FOR CACRCVD                                    *
//MDLDSCBD DD DSN=ACAC..V8R2M0.CACRCVD,                      *
//      UNIT=&DISKU,Vol=SER=&DISK1,                           *
//      SPACE=(TRK,(0)),                                     *
//      DCB=(RECFM=FBS,LRECL=80,BLKSIZE=32760),               *
//      DISP=(NEW,KEEP)                                      *
/*
/* MODEL DSCB FOR CACRCVX                                    *
//MDLDSCBX DD DSN=ACAC..V8R2M0.CACRCVX,                      *
//      UNIT=&DISKU,Vol=SER=&DISK1,                           *
//      SPACE=(TRK,(0)),                                     *
//      DCB=(RECFM=FBS,LRECL=80,BLKSIZE=32760),               *
//      DISP=(NEW,KEEP)                                      *
/*
//GDGALLOC EXEC PGM=IEFBR14,COND=(0,NE,GDGDEF)                *
//CACRCVD DD DSN=ACAC..V8R2M0.CACRCVD(+1),                   *
//      UNIT=&DISKU,Vol=SER=&DISK2,                           *
//      SPACE=(TRK,(1)),                                     *
//      DISP=(NEW,CATLG)                                     *
/*
//CACRCVX DD DSN=ACAC..V8R2M0.CACRCVX(+1),                   *
//      UNIT=&DISKU,Vol=SER=&DISK2,                           *
//      SPACE=(TRK,(1)),                                     *
//      DISP=(NEW,CATLG)                                     *
//PEND                                                          *
//ALOCGDG EXEC CACGDGA                                          *
//GDGDEF.SYSIN DD *                                           *
DEFINE GENERATIONDATAGROUP -                                 *
( NAME(CAC..V8R2M0.CACRCVD) - NOEMPTY - SCRATCH - LIMIT(5))  *
DEFINE GENERATIONDATAGROUP -                                 *
( NAME(CAC..V8R2M0.CACRCVX) - NOEMPTY - SCRATCH -            *
STEP 6A8: Configure the JCL and control data sets to run WIICEP for IMS

“STEP 3h: Configure the JCL and control datasets to run WIICEP for IMS” on page 22 describes the configuration of the JCL and control data sets to execute WIICEP for IMS.

Example 3-22 shows the CAC.V8R2M0.REDBOOK(CACCSCF) configuration data set — which is a modified version of CAC.V8R2M0.SACCONF(CACCSCF) — with its SIE and publication information for the CIF solution.

The five publication definitions PUBCUSTN, PUBCUSTA, PUBPOLY, PUBPREM and PUBCLAIM in Example 3-22 show one corresponding to each “logical” table defined earlier, and specify transaction messages for each one, with all of the five publications directed to the same send queue IMSK_SENDQ1. It also specifies that the before image of the changed data must be included in the message.

Example 3-22  SIEs and publication information for the CIF solution

```plaintext
* CORRELATION SERVICE
SERVICE INFO ENTRY = CACECA2 XM1/XQM/CSQ1/16 2 1 1 16 1 10MS 30S \ 
   TCP/WSC53.ITSO.IBM.COM/5555,CSA=1K,CSARLSE=3,INT=1,COLDSTART *
*
* XML OVER MQ PUBLISH/SUBSCRIBE
SERVICE INFO ENTRY = CACPUB PUB1 2 1 1 100 1 5M 5M \MQI/MQ8G/CAC_RESTARTQ *
*
PUB ALIAS=PUBCUSTN,
   MSGTYPE=TRANS,
   QUEUE=MQI/MQ8G/IMSK_SENDQ1,
   TABLE=IMSKAUTO.CUSTOMER,
   BEFORE_VALUES=YES *
*
PUB ALIAS=PUBCUSTA,
   MSGTYPE=TRANS,
   QUEUE=MQI/MQ8G/IMSK_SENDQ1,
   TABLE=IMSKAUTO.ADDRESS,
   BEFORE_VALUES=YES *
*
PUB ALIAS=PUBPOLY,
   MSGTYPE=TRANS,
   QUEUE=MQI/MQ8G/IMSK_SENDQ1,
   TABLE=IMSKAUTO.POLICY,
   BEFORE_VALUES=YES *
*
PUB ALIAS=PUBPREM,
   MSGTYPE=TRANS,
   QUEUE=MQI/MQ8G/IMSK_SENDQ1,
   TABLE=IMSKAUTO.PREMIUMS,
   BEFORE_VALUES=YES *
*
PUB ALIAS=PUBCLAIM,
   MSGTYPE=TRANS,
   QUEUE=MQI/MQ8G/IMSK_SENDQ1,
   TABLE=IMSKAUTO.CLAIMS,
   BEFORE_VALUES=YES *
```

This concludes the configuration of WIICEP for IMS.

3.3.7 STEP 6B: Configure WIIEP for DB2 UDB for z/OS

This section provides a step-by-step guide to configuring WebSphere Information Integrator Event Publisher for DB2 UDB for z/OS for the CIF and Event Alert System solution. The main steps are shown in Figure 3-65, and described in more detail in the following subsections.

<table>
<thead>
<tr>
<th>STEP 6B1: Install DB2 Admin Client on Windows with DB2 Connect access to z/OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>STEP 6B2: Create Q Capture control tables on DB2 for z/OS</td>
</tr>
<tr>
<td>STEP 6B3: Create the Publication Queue Maps on DB2 for z/OS</td>
</tr>
<tr>
<td>STEP 6B4: Create the Publications on DB2 for z/OS</td>
</tr>
</tbody>
</table>

Figure 3-65  WIIEP for DB2 UDB configuration steps for CIF and Event Alert System solution

STEP 6B1: Install DB2 Admin Client on Windows with DB2 Connect access to z/OS

The Replication Center performs administration functions for SQL replication and Q replication, with Event Publishing being a subset of the functions provided by Q replication. As mentioned earlier, Event Publishing only uses the Q Capture and administration components of Q replication, but does not use the Q Apply component since the XML messages are consumed by user-written applications instead.

For details on Q replication refer to the IBM redbook *WebSphere Information Integration Q Replication: Fast Track Implementation Scenarios*, SG24-6487.
Event Publishing specific administration functions supported by the Replication Center include the following actions:

- Create Q Capture control tables.
- Create Publishing Queue Maps which identify the target message queues for changed data.
- Create Publications which identify the source tables and content to be captured and published.
- Start and stop Q Capture.
- Monitor Q Capture.

Two possible configuration options are available for Replication Center on Windows to access DB2 for z/OS as follows:

- DB2 Connect Personal Edition which includes the Administration Client as shown in Figure 3-66.

![Figure 3-66 Windows DB2 Connect Personal Edition access to DB2 for z/OS](image)

- DB2 UDB Administration Client with z/OS access via the DB2 Connect server as shown in Figure 3-67.

![Figure 3-67 DB2 Administration Client with DB2 for z/OS access via DB2 Connect server](image)

We chose the configuration involving the DB2 Personal Edition shown in Figure 3-66. The DB2 for z/OS database server was configured to listen to DRDA® requests at IP Address 9.12.6.77 on port 38060. The DB2 Connect workstation client was configured using the commands to catalog the DB8G database as shown in Example 3-23.

**Example 3-23 Catalog DB8G database commands**

```
catalog tcpip node DB2NODE remote 9.12.6.77 server 38060
catalog database DB8G as DB8G at node DB2NODE
catalog dcs database DB8G as DB8G
```
STEP 6B2: Create Q Capture control tables on DB2 for z/OS

Replication Center is used to create the Q Capture tables.

Expand the navigation tree in the Q Replication folder, select and right-click Q Capture Servers, and select Create Q Capture Control Tables as shown in Figure 3-68 to proceed to Figure 3-69.

Figure 3-68  Create Q Capture control tables 1/4

Figure 3-69  Create Q Capture control tables 2/4
Specify the Q Capture server details such as Q Capture schema, DB2 subsystem and database as shown in Figure 3-69, and click **Next** to proceed to Figure 3-70.

**Figure 3-70  Create Q Capture control tables 3/4**

Specify the WebSphere MQ object details such as the Queue Manager, Administration queue and Restart queue as shown in Figure 3-70, and click **Finish**. This will generate an SQL script (to create eleven control tables shown using the Control Center in Figure 3-71) which needs to be executed on the z/OS platform.

**Figure 3-71  Create Q Capture control tables 4/4**
STEP 6B3: Create the Publication Queue Maps on DB2 for z/OS
The Replication Center is used to create the Publication Queue Maps as well.

Expand the navigation tree in the Q Replication folder, and navigate to Publication Queue Maps and right-click on it and select Create as shown in Figure 3-72 to proceed to Figure 3-73.

![Figure 3-72 Create the Publication Queue Maps 1/3](image1)

Provide Publication Queue Map details under the General and Properties tab as shown in Figure 3-72, and click OK. This will generate an SQL script which needs to be executed on the z/OS platform. Three Publication Queue Maps EVPUB1, EVPUB2 and EVPUB3 are defined as shown using the Replication Center in Figure 3-74.

![Figure 3-73 Create the Publication Queue Maps 2/3](image2)
STEP 6B4: Create the Publications on DB2 for z/OS

Replication Center is used to create the publications. Navigate to XML Publications from the navigation tree, and right-click on it and select Create as shown in Figure 3-75 to proceed to Figure 3-76.
Provide the Publishing queue map (EVPUB1) details in Figure 3-76 and click **Next** to proceed to Figure 3-77.

![Select Source Tables](image)

**Figure 3-77  Create publications 3/7**

Click **Retrieve** to obtain the list of tables with a Creator of DB2GHOME as shown in Figure 3-77.

![Create XML Publications](image)

**Figure 3-78  Create publications 4/7**
Select all the tables in Figure 3-78 and click **Next** to proceed to Figure 3-79.

![Create XML Publications](image)

Figure 3-79  Create publications 5/7

Specify the detailed content of the messages that Q Capture publishes in Figure 3-79, and click **Next** to list the definition of the publications CUSTOMER0001, POLICY0001, PREMIUM0001 and CLAIM0001 as shown in Figure 3-80.

![Create XML Publications](image)

Figure 3-80  Create publications 6/7
Click **Finish** in Figure 3-80 to generate an SQL script (that includes the definition of the four publications) that needs to be executed on the z/OS platform. Figure 3-81 shows the XML publications in the Replication Center.

This concludes the configuration of WIIEP for DB2 for z/OS.

### 3.3.8 STEP 7: Define and configure CIF ODS and install programs

Figure 3-82 describes the steps to be performed to initialize and configure the CIF and Event Notification environment on the Windows platform.

**STEP 7a:** Define the CIF ODS database

The CIF ODS provides a holistic view of the Trustworthy Insurance’s customers on a Windows 2000 platform.

Figure 3-83 shows the tables in the CIF ODS, where tables CUSTOMER, POLICY, PREMIUMS, and CLAIMS map more or less directly with the home insurance database on DB2 UDB for z/OS. CUSTLINK is a mapping table where the mapping of the customer key in the CIF ODS database, the customer key of the auto insurance database, and the customer key of the home insurance database is kept since the customer key domains (data type and value) of auto insurance and home insurance are different.
Example 3-24 shows the DB2 UDB for Windows DDL to create the CIF ODS tables.

```sql
create table CIF.CUSTLINK (  
CIF_CUSTOMER_NUMBER INTEGER NOT NULL,  
IMSK_CUSTKEY CHAR(10) NOT NULL,  
DB2G_CUSTOMER INTEGER NOT NULL  
) IN USERSPACE1 ;

create table CIF.CUSTOMER (  
CIF_CUSTOMER_NUMBER INTEGER NOT NULL,  
FIRST_NAME CHAR(30),  
LAST_NAME CHAR(30),  
ADDRESS_LINE1 CHAR(30),  
ADDRESS_LINE2 CHAR(30),  
ADDRESS_LINE3 CHAR(30),  
ADDRESS_LINE4 CHAR(30),  
ZIPCODE CHAR(10),  
DELIMSK_IND CHAR(1),  
DELDB2G_IND CHAR(1)  
) IN USERSPACE1 ;

create table CIF.POLICY (  
CIF_CUSTOMER_NUMBER INTEGER NOT NULL,  
SOURCE_SYSTEM CHAR(4) NOT NULL,  
POLICY_NUMBER CHAR(10),  
POLICY_START DATE,  
DELIMSK_IND CHAR(1),  
DELDB2G_IND CHAR(1)  
) IN USERSPACE1 ;

create table CIF.PREMIUM (  
CIF_CUSTOMER_NUMBER INTEGER NOT NULL,  
SOURCE_SYSTEM CHAR(4) NOT NULL,  
POLICY_NUMBER CHAR(10),  
RISK_LINE_NUMBER CHAR(2),  
RISK_LINE_PREMIUM DECIMAL(10,2),  
DELIMSK_IND CHAR(1),  
DELDB2G_IND CHAR(1)  
) IN USERSPACE1 ;
```

Figure 3-83  CIF ODS database

**Example 3-24  CIF ODS database DDL**
create table CIF.CLAIM (  
CIF_CUSTOMER_NUMBER INTEGER NOT NULL,  
SOURCE_SYSTEM CHAR(4) NOT NULL,  
POLICY_NUMBER CHAR(10),  
CLAIM_NUMBER CHAR(10),  
CLAIM_DATE DATE,  
CLAIM_AMOUNT DECIMAL(10,2),  
DELMSK_IND CHAR(1),  
DELDB2G_IND CHAR(1)  
) IN USERSPACE1 ;

Step 7b: Synchronize the CIF ODS with the data unloaded earlier

The CIF ODS database must be initialized with the data unloaded in Section 3.3.3, “STEP 3: Unload auto insurance and home insurance data” on page 96.

Although it is beyond the scope of this redbook, anytime replication is involved, the target database needs to be initially populated and potentially refreshed at regular intervals. Ideally, when the source is non-relational, WebSphere Information Integrator Classic Federation should be used to achieve this end. Additionally, at periodic intervals, the database administrator should perform “diffs” between the target database and the source database. Once again, when the source is non-relational, WebSphere Information Integrator Classic Federation should be used for this.

Attention: The population of the CUTLINK table is assumed to occur by a process outside the scope of our event publishing environment — both the initial mapping and subsequent mappings when customers get added and deleted in the auto insurance and home insurance databases. Figure 3-84 shows the typical contents of a CUSTLINK table — note the numeric surrogate key of the CIF ODS database, the alphanumeric customer key of the auto insurance IMS database, and numeric customer key of the home insurance database.

<table>
<thead>
<tr>
<th>CIF_CUSTOMER_NUMBER</th>
<th>IMSK_CUSTKEY</th>
<th>DB2G_CUSTOMER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>G0000000050</td>
<td>41028</td>
</tr>
<tr>
<td>2</td>
<td>H0000000010</td>
<td>24928</td>
</tr>
<tr>
<td>3</td>
<td>H0000000090</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>K0000000030</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>F0000000070</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>S0000000060</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>T0000000020</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>-</td>
<td>73664</td>
</tr>
<tr>
<td>9</td>
<td>-</td>
<td>63382</td>
</tr>
<tr>
<td>10</td>
<td>-</td>
<td>44920</td>
</tr>
<tr>
<td>11</td>
<td>-</td>
<td>33842</td>
</tr>
</tbody>
</table>

Figure 3-84  Sample CUSTLINK mapping table

STEP 7c: Configure CIF Updater and Event Notification programs

The general flow of an XML message consuming application is as follows:

1. Connect to WebSphere MQ
2. Read the XML message from the queue
3. Parse the XML in the message
4. Dispatch the message to the application
5. Perform application-specific code
XML event message consumption applications may be written in many ways:

1. Using a Java program:
   - Java is particularly suitable because IBM has published a set of java classes in the Event Publishing Toolkit to help with many aspects this programming task.

2. Using any other program that has an interface to WebSphere MQ:
   - Additional programming will be involved because the XML Publishing Toolkit (discussed in Appendix A.4, “XML Publishing Toolkit” on page 238) is only available in Java.

3. Using DB2 MQ Listener:
   - The XML Extender and XML wrapper provide considerable assistance in processing XML contents.

4. Using WebSphere Business Integration Message Broker:
   - Java adapters and XML parsers are available within the WBI framework to help with the tasks of processing XML messages.

The CIF Updater and Event Notification programs used in the Trustworthy Insurance scenario are described here.

**CIF Updater programs**

The CIF Updater programs in our scenario have been written to use Java and the XML Publishing Toolkit for the following reasons:

- The XML Publishing Toolkit is a powerful part of the WebSphere Information Integration Event Publishing product suite and is the strategic direction.
- Provide guidance on developing XML message consuming applications without the use of advanced tools such as the WebSphere Business Integrator Message Broker for developers who might prefer such a route.

The CIF Updater program functions are implemented as follows:

1. Connect to MQ implemented using Java MQ classes.
2. Read an XML event message from the queue, and parse the XML in the message using Java programming with XML Publishing Toolkit classes.
3. Dispatch the message to the application, and perform application-specific code using JDBC calls to a DB2 SQL Stored Procedure (2PC).

Figure 3-85 shows the components used in the CIF Updater programs, while Figure 3-86 and Figure 3-87 provide an overview of how changes originating from WIICEP for IMS and WIIEP for DB2 UDB for z/OS are processed respectively. Figure 3-88 describes the processing of an individual XML message originating from WIIEP for IMS.

The individual classes, stored procedure, and application code are shown in Example 3-25 through Example 3-33.
Note: The CIF Updater programs are downloadable from:
ftp://www.redbooks.ibm.com/redbooks/SG247132

Figure 3-85  CIF Updater program structure

MQ Listener class
mqlimsk.class
Connect to WebSphere MQ
Read the message from the queue
Parse the XML in the message

Data Handler class
Replims.class
Detect Publication Type
Dispatch the message to the application

DB2 UDB SQL Stored procedures
Do application-specific code

RECVQ

Figure 3-86  CIF Updater program functional details 1/3

IMS & Classic Event Publisher

WMQ

WMQ

DB2 UDB – Customer Information Facility

Java Consumer Application
• WMQ as Txn co-ordinator
• DB2 as participant

Begin Txn
Receive Trans Message
Parse XML
Loop - for each row:
• Determine Pub Tyoe
• Read xml element values
• Assign values to variables
• Call Stored Procedure
Commit Txn

CUSTOMER

POLICY

CLAIM

PREMIUM

CIF

CUSTOMER

POLICY

ADDRESS

CLAIMS

PREMIUMS

CIF
Figure 3-87  CIF Updater program functional details 2/3

Figure 3-88  CIF Updater program functional details 3/3
Example 3-25  MQ Listener class

```java
import com.ibm.mq.*;
import com.ibm.db2.tools.repl.publication.*;
import com.ibm.db2.tools.repl.publication.support.*;

class mqlimsk{
    public static void main(String args[]) throws Exception{
        MQQueueManager qmgr = null;
        MQQueue q = null;
        MQMessage getMessage = null;
        MQGetMessageOptions pmo = null;

        try{
            // This message handler class performs the following functions //
            // 1. establish a connection to a Message Queue that receives XML Publications //
            // 2. listen for new messages continuously (subject to a 120 second timeout) //
            // 3. parse the message using the XML Publication toolkit //
            // 4. determine if the message is transactional XML Event publication message //
            // 5. If it is transactional XML Event publication message, call the replims data
            // handler class //
            // see Websphere MQ publication SC34-6066 "Using Java" for programming details for
            // Websphere MQ //

            // define a parser class using the xml Publication Toolkit PublicationParser //
            PublicationParser parser = new PublicationParser();

            // create a connection to the Queue Manager //
            java.util.Hashtable mqProperties = new java.util.Hashtable();
            mqProperties.put(MQC.THREAD_AFFINITY, new Boolean(true));
            qmgr = new MQQueueManager("QMCIF",mqProperties);

            // establish a two phase commit connect to the Queue manager, and the XA database
            // resource (CIF) //
            Replims replims = new Replims(qmgr);

            // setup the options for the queue we wish to open, including a 120 second timeout //
            int openOption = MQC.MQOO_INPUT_AS_Q_DEF;
            q = qmgr.accessQueue("IMSK_RECVQ1", openOption);
            getMessage= new MQMessage();
            MQGetMessageOptions mqgmo = new MQGetMessageOptions();
            mqgmo.options = MQC.MQGMO_WAIT | MQC.MQGMO_SYNCPOINT;
            mqgmo.matchOptions = MQC.MQMO_NONE ;
            mqgmo.waitInterval = 120000; // 120 sec.

            // begin an MQ logical unit of work //
            qmgr.begin();

            // loop whilst a get message can be retrieved within the 120 second timeout interval
            //
            while(true){
                // get next message off the queue //
```
Example 3-26  XML Parser and Data handler class (IMS)

```java
import java.sql.*;
import com.ibm.mq.*;
import javax.transaction.xa.*;
import com.ibm.db2.tools.repl.publication.*;
import com.ibm.db2.jcc.*;

public class Replims {

    // setup variables for jdbc connection to database CIF //
    private String url  = "jdbc:db2:CIF";
    private String id   = "db2admin";
    private String pass = "itsosj";

    // setup variables for db2 connection as an XA two phase commit resource //

    getMessage.clearMessage();
    q.get(getMessage, mqgmo);

    // get message length and print the message to std out //

    String xmlMsg = getMessage.readString(getMessage.getMessageLength());
    System.out.println(xmlMsg);

    // parse message using the WMQ xml parser //

    Msg msg = parser.parse(xmlMsg);
    if (msg instanceof TransactionMsg){
        // if it is a transactional message call class replims using method ccdinsert, then commit and read the next message //
        if (replims.ccdInsert(msg)) {
            replims.registerUpdate();
            qmgr.commit();
            qmgr.begin();
        }
        // if it is not a transactional message (maybe a rowop message) do not process the message - just commit and read the next message //
    } else {
        qmgr.commit();
        qmgr.begin();
    }
}
```
private Connection con;
private DB2XDataSource ds;

// setup variables for four parameterised call types to stored procedures //

private String sql1 = "call DB2ADMIN.SPIMSKCUST(?,?,?,?,?,?,?,?,?,?,?,?,?)";
private String sql2 = "call DB2ADMIN.SPIMSKPOLICY(?,?,?,?,?,?,?,?,?,?,?,?)";
private String sql3 = "call DB2ADMIN.SPIMSKPREMIUM(?,?,?,?,?,?,?)";
private String sql4 = "call DB2ADMIN.SPIMSKCLAIM(?,?,?,?,?,?,?,?)";

private PreparedStatement stmt1;
private PreparedStatement stmt2;
private PreparedStatement stmt3;
private PreparedStatement stmt4;

// setup variables to store change capture log sequence number and timestamp //

private byte[] cmtLSN;
private Timestamp cmtTM;

// method to establish connection to DB2 //

public Replims() throws Exception {
    cmtLSN = new byte[10];
    System.out.println("url = " + url);
    Class.forName("com.ibm.db2.jcc.DB2Driver").newInstance();
    con = java.sql.DriverManager.getConnection(url, id, pass);
    con.setAutoCommit(false);
    System.out.println("Connect DB");
    stmt1 = con.prepareStatement(sql1);
    stmt2 = con.prepareStatement(sql2);
    stmt3 = con.prepareStatement(sql3);
    stmt4 = con.prepareStatement(sql4);
}

// method to establish two phase commit connection to WMQ and DB2 //

public Replims(MQQueueManager qMgr) throws Exception {
    cmtLSN = new byte[10];
    ds = new DB2XDataSource();
    ds.setDatabaseName("CIF");
    con = qMgr.getJDBCConnection(ds, id, pass);
    con.setAutoCommit(false);
    System.out.println("Connect DB");
    stmt1 = con.prepareStatement(sql1);
    stmt2 = con.prepareStatement(sql2);
    stmt3 = con.prepareStatement(sql3);
    stmt4 = con.prepareStatement(sql4);
}

public void commit() throws Exception {
    con.commit();
}

public void rollback() throws Exception {
    con.rollback();
}

public void registerUpdate() throws Exception {
    String DUMMYONE = null;
    DUMMYONE = "ABC";
public boolean ccdInsert(Msg pubMsg) throws Exception {

    TransactionMsg tranMsg = (TransactionMsg) pubMsg;
    cmtLSN = format_CMTSEQ(tranMsg.getCommitLSN());
    cmtTM  = format_CMTTM(tranMsg.getCommitTime());

    java.util.Vector rows = tranMsg.getRows();
    for (int i=0; i < rows.size(); i++) {
        Row row = (Row) rows.elementAt(i);
        if (row.getSrcName().equals("CUSTADDR")) {
            applycustaddr(i,row);
        } else if (row.getSrcName().equals("POLICY")) {
            applypolicy(i,row);
        } else if (row.getSrcName().equals("PREMIUMS")) {
            applypremiums(i,row);
        } else if (row.getSrcName().equals("CLAIMS")) {
            applyclaims(i,row);
        }
    }
    return tranMsg.isLast();
}

private void applycustaddr(int i,Row row) throws Exception {

    // method to perform data handling for a customer record //
    // 1. set all before-image and after-image variables to null //
    // 2. use XML Publication Toolkit row and column methods to //
    //    - determine the row operation type I U or D //
    //    - set the before-image and after-image variable for each column //
    // call the CIF database stored procedure to handle the change row //

    String CUSTKEY   = null;
    String FIRSTNME  = null;
    String LASTNAME  = null;
    String ADDRTYPE   = null;
    String ADDRESS1  = null;
    String ADDRESS2  = null;
    String ADDRESS3  = null;
    String ZIPCODE   = null;
    String XCUSTKEY   = null;
    String XFIRSTNME  = null;
    String XLASTNAME  = null;
    String XADDRTYPE   = null;
    String XADDRESS1  = null;
    String XADDRESS2  = null;
    String XADDRESS3  = null;
    String XZIPCODE   = null;

    String rowOp  = format_ROWOP(row.getRowOperation());
    // byte[] intSEQ = new byte[10];
    // intSEQ = format_INTSEQ(i);
}
java.util.Vector cols = row.getColumns();
for (int j=0; j < cols.size(); j++) {
  Column col = (Column) cols.elementAt(j);
  String colName = col.getName();
  if (colName.equals("CUSTKEY")){
    CUSTKEY = (String) col.getValue();
  }else if (colName.equals("XCUSTKEY")){
    XCUSTKEY = (String) col.getValue();
  }else if (colName.equals("FIRSTNME")){
    FIRSTNME = (String) col.getValue();
  }else if (colName.equals("XFIRSTNME")){
    XFIRSTNME = (String) col.getValue();
  }else if (colName.equals("LASTNAME")){
    LASTNAME = (String) col.getValue();
  }else if (colName.equals("XLASTNAME")){
    XLASTNAME = (String) col.getValue();
  }else if (colName.equals("ADDRTYPE")){
    ADDRTYPE = (String) col.getValue();
  }else if (colName.equals("XADDRTYPE")){
    XADDRTYPE = (String) col.getValue();
  }else if (colName.equals("ADDRESS1")){
    ADDRESS1 = (String) col.getValue();
  }else if (colName.equals("XADDRESS1")){
    XADDRESS1 = (String) col.getValue();
  }else if (colName.equals("ADDRESS2")){
    ADDRESS2 = (String) col.getValue();
  }else if (colName.equals("XADDRESS2")){
    XADDRESS2 = (String) col.getValue();
  }else if (colName.equals("ADDRESS3")){
    ADDRESS3 = (String) col.getValue();
  }else if (colName.equals("XADDRESS3")){
    XADDRESS3 = (String) col.getValue();
  }else if (colName.equals("ZIPCODE")){
    ZIPCODE = (String) col.getValue();
  }else if (colName.equals("XZIPCODE")){
    XZIPCODE = (String) col.getValue();
  }
}
stmt1.setString    (1,rowOp);
stmt1.setString    (2,CUSTKEY);
stmt1.setString    (3,FIRSTNME);
stmt1.setString    (4,LASTNAME);
stmt1.setString    (5,ADDRTYPE);
stmt1.setString    (6,ADDRESS1);
stmt1.setString    (7,ADDRESS2);
stmt1.setString    (8,ADDRESS3);
stmt1.setString    (9,ZIPCODE);
stmt1.setString    (10, XCUSTKEY);
stmt1.setString    (11, XFIRSTNME);
stmt1.setString    (12, XLASTNAME);
stmt1.setString    (13, XADDRTYPE);
stmt1.setString    (14, XADDRESS1);
stmt1.setString    (15, XADDRESS2);
stmt1.setString    (16, XADDRESS3);
stmt1.setString    (17, XZIPCODE);
stmt1.executeUpdate();
private void applyPolicy(int i, Row row) throws Exception {

    // method to perform data handling for a policy record
    // 1. set all before-image and after-image variables to null
    // 2. use XML Publication Toolkit row and column methods to
    //    - determine the row operation type I U or D
    //    - set the before-image and after-image variable for each column
    // call the CIF database stored procedure to handle the change row

    String CUSTKEY   = null;
    String POLICYNM  = null;
    String POLYTYPE  = null;
    String POLCOVER  = null;
    String POLSTDTE  = null;
    String POLNOTES  = null;
    String XCUSTKEY   = null;
    String XPOLICYNM  = null;
    String XPOLYTYPE  = null;
    String XPOLCOVER  = null;
    String XPOLSTDTE  = null;
    String XPOLNOTES  = null;

    String rowOp  = format_ROWOP(row.getRowOperation());
    // byte[] intSEQ = new byte[10];
    // intSEQ = format_INTSEQ(i);

    java.util.Vector cols = row.getColumns();
    for (int j=0; j < cols.size(); j++) {
        Column col = (Column) cols.elementAt(j);
        String colName = col.getName();
        if       (colName.equals("CUSTKEY")){
            CUSTKEY = (String) col.getValue();
        }else if (colName.equals("XCUSTKEY")){
            XCUSTKEY = (String) col.getValue();
        }else if (colName.equals("POLICYNM")){
            POLICYNM = (String) col.getValue();
        }else if (colName.equals("XPOLICYNM")){
            XPOLICYNM = (String) col.getValue();
        }else if (colName.equals("POLYTYPE")){
            POLYTYPE = (String) col.getValue();
        }else if (colName.equals("XPOLYTYPE")){
            XPOLYTYPE = (String) col.getValue();
        }else if (colName.equals("POLCOVER")){
            POLCOVER = (String) col.getValue();
        }else if (colName.equals("XPOLCOVER")){
            XPOLCOVER = (String) col.getValue();
        }else if (colName.equals("POLSTDTE")){
            POLSTDTE = (String) col.getValue();
        }else if (colName.equals("XPOLSTDTE")){
            XPOLSTDTE = (String) col.getValue();
        }else if (colName.equals("POLNOTES")){
            POLNOTES = (String) col.getValue();
        }else if (colName.equals("XPOLNOTES")){
            XPOLNOTES = (String) col.getValue();
    }
}
stmt2.setString(1, rowOp);
stmt2.setString(2, CUSTKEY);
stmt2.setString(3, POLICYNM);
stmt2.setString(4, POLYTYPE);
stmt2.setString(5, POLCOVER);
stmt2.setString(6, POLSTDTE);
stmt2.setString(7, POLNOTES);
stmt2.setString(8, XCUSTKEY);
stmt2.setString(9, XPOLICYNM);
stmt2.setString(10, XPOLYTYPE);
stmt2.setString(11, XPOLCOVER);
stmt2.setString(12, XPOLSTDTE);
stmt2.setString(13, XPOLNOTES);

stmt2.executeUpdate();
System.out.println("i = " + i);

private void applypremiums(int i, Row row) throws Exception {

// method to perform data handling for a premium record //
// 1. set all before-image and after-image variables to null //
// 2. use XML Publication Toolkit row and column methods to //
//    - determine the row operation type I U or D //
//    - set the before-image and after-image variable for each column //
// call the CIF database stored procedure to handle the change row //

String CUSTKEY = null;
String POLICYNM = null;
String RISKLINM = null;
java.math.BigDecimal RPREMIUM = null;
String XCUSTKEY = null;
String XPOLICYNM = null;
String XRISKLINM = null;
java.math.BigDecimal XRPREMIUM = null;

String rowOp = format_ROWOP(row.getRowOperation());
// byte[] intSEQ = new byte[10];
// intSEQ = format_INTSEQ(i);

java.util.Vector cols = row.getColumns();
for (int j=0; j < cols.size(); j++) {
  Column col = (Column) cols.elementAt(j);
  String colName = col.getName();
  if (colName.equals("CUSTKEY")) {
    CUSTKEY = (String) col.getValue();
  } else if (colName.equals("XCUSTKEY")) {
    XCUSTKEY = (String) col.getValue();
  } else if (colName.equals("POLICYNM")) {
    POLICYNM = (String) col.getValue();
  } else if (colName.equals("XPOLICYNM")) {
    XPOLICYNM = (String) col.getValue();
  } else if (colName.equals("RISKLINM")) {
    RISKLINM = (String) col.getValue();
  } else if (colName.equals("XRISKLINM")) {
    XRISKLINM = (String) col.getValue();
  } else if (colName.equals("RPREMIUM")) {
    RPREMIUM = (java.math.BigDecimal) col.getValue();
  } else if (colName.equals("XRPREMIUM")) {
    XRPREMIUM = (java.math.BigDecimal) col.getValue();
  }
}
RISKLINM = (String) col.getValue();
} else if (colName.equals("XRISKLINM")){
    XRISKLINM = (String) col.getValue();
} else if (colName.equals("RPREMIUM")){
    RPREMIUM = (java.math.BigDecimal) col.getValue();
} else if (colName.equals("XRPREMIUM")){
    XRPREMIUM = (java.math.BigDecimal) col.getValue();
}

stmt3.setString    (1,rowOp);
stmt3.setString    (2,CUSTKEY);
stmt3.setString    (3,POLICYNM);
stmt3.setString    (4,RISKLINM);
stmt3.setBigDecimal      (5,RPREMIUM);
stmt3.setString    (6,CUSTKEY);
stmt3.setString    (7,POLICYNM);
stmt3.setString    (8,RISKLINM);
stmt3.setBigDecimal      (9,RPREMIUM);
stmt3.executeUpdate();
System.out.println("i = " + i);
}

private void applyclaims(int i,Row row) throws Exception {
    // method to perform data handling for a claim record //
    // 1. set all before-image and after-image variables to null //
    // 2. use XML Publication Toolkit row and column methods to //
    //    - determine the row operation type I U or D //
    //    - set the before-image and after-image valuable for each column //
    // call the CIF database stored procedure to handle the change row //

    String CUSTKEY   = null;
    String POLICYNM  = null;
    String CLAIMNM  = null;
    String CLAIMDTE  = null;
    java.math.BigDecimal CLAIMAMT  = null;
    String CLAIMDET  = null;
    String XCUSTKEY  = null;
    String XPOLICYNM = null;
    String XCLAIMNM = null;
    String XCLAIMDTE = null;
    java.math.BigDecimal XCLAIMAMT = null;
    String XCLAIMDET = null;

    String rowOp  = format_ROWOP(row.getRowOperation());
    byte[] intSEQ = new byte[10];
    // byte[] intSEQ = format_INTSEQ(i);

    java.util.Vector cols = row.getColumns();
    for (int j=0; j < cols.size(); j++) {
        Column col = (Column) cols.elementAt(j);
        String colName = col.getName();
        if (colName.equals("CUSTKEY")){
            CUSTKEY   = col.getValue();
        } else if (colName.equals("POLICYNM")){
            POLICYNM  = col.getValue();
        } else if (colName.equals("CLAIMNM")){
            CLAIMNM  = col.getValue();
        } else if (colName.equals("CLAIMDTE")){
            CLAIMDTE  = col.getValue();
        } else if (colName.equals("CLAIMAMT")){
            CLAIMAMT  = col.getValue();
        } else if (colName.equals("CLAIMDET")){
            CLAIMDET  = col.getValue();
        } else if (colName.equals("XCUSTKEY")){
            XCUSTKEY  = col.getValue();
        } else if (colName.equals("XPOLICYNM")){
            XPOLICYNM = col.getValue();
        } else if (colName.equals("XCLAIMNM")){
            XCLAIMNM = col.getValue();
        } else if (colName.equals("XCLAIMDTE")){
            XCLAIMDTE = col.getValue();
        } else if (colName.equals("XCLAIMAMT")){
            XCLAIMAMT = col.getValue();
        } else if (colName.equals("XCLAIMDET")){
            XCLAIMDET = col.getValue();
        }
    }
}
CUSTKEY = (String) col.getValue();
} else if (colName.equals("XCUSTKEY")){
    XCUSTKEY = (String) col.getValue();
} else if (colName.equals("POLICYNM")){
    POLICYNM = (String) col.getValue();
} else if (colName.equals("XPOLICYM")){
    XPOLICYNM = (String) col.getValue();
} else if (colName.equals("CLAIMNM")){
    CLAIMNM = (String) col.getValue();
} else if (colName.equals("XCLAIMNM")){
    XCLAIMNM = (String) col.getValue();
} else if (colName.equals("CLAIMDTE")){
    CLAIMDTE = (String) col.getValue();
} else if (colName.equals("XCLAIMDTE")){
    XCLAIMDTE = (String) col.getValue();
} else if (colName.equals("CLAIMAMT")){
    CLAIMAMT = (java.math.BigDecimal) col.getValue();
} else if (colName.equals("XCLAIMAMT")){
    XCLAIMAMT = (java.math.BigDecimal) col.getValue();
} else if (colName.equals("CLAIMDET")){
    CLAIMDET = (String) col.getValue();
} else if (colName.equals("XCLAIMDET")){
    XCLAIMDET = (String) col.getValue();
}

stmt4.setString (1,rowOp);
stmt4.setString (2,CUSTKEY);
stmt4.setString (3,POLICYNM);
stmt4.setString (4,CLAIMNM);
stmt4.setString (5,CLAIMDTE);
stmt4.setBigDecimal (6,CLAIMAMT);
stmt4.setString (7,CLAIMDET);
stmt4.setString (8,XCUSTKEY);
stmt4.setString (9,XPOLICYNM);
stmt4.setString (10,XCLAIMNM);
stmt4.setString (11,XCLAIMDTE);
stmt4.setString (12,XCLAIMAMT);
stmt4.setString (13,XCLAIMDET);

stmt4.executeUpdate();
System.out.println("i = " + i);

private byte[] format_CMTSEQ(String cmtLSN){
    String s1 = cmtLSN.replaceAll( ":" , "" ) ;
    return toBinary (s1 ) ;
}

private byte[] format_INTSEQ(int intSEQ){
    String s1 = Integer.toString(intSEQ).trim();
    int ln = 20 - s1.length();
    for (int i = 1; i <= ln ; i++){
        s1 = "0" + s1;
    }
    return toBinary(s1);
Chapter 3. Trustworthy Insurance Customer Information Facility (CIF) and Event Alert System scenario

Example 3-27  XML Parser and Data handler class (DB2)

```java
import java.sql.*;
import com.ibm.mq.*;
import javax.transaction.xa.*;
import com.ibm.db2.tools.repl.publication.*;
import com.ibm.db2.jcc.*;

public class Repldb2 {

    // setup variables for jdbc connection to database CIF //
    private String url = "jdbc:db2:CIF";
    private String id = "db2admin";
    private String pass = "itsosj";

    // setup variables for db2 connection as an XA two phase commit resource //
    private Connection con;
    private DB2XDataSource ds;

    // setup variables for four parameterised call typess to stored procedures //
    private String sql1 = "call DB2ADMIN.SPDB2GCUST(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)";
    private String sql2 = "call DB2ADMIN.SPDB2GPOLICY(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)";
    private String sql3 = "call DB2ADMIN.SPDB2GPREMIUM(?,?,?,?,?,?,?,?,?)";
    private String sql4 = "call DB2ADMIN.SPDB2GCLAIM(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)";
```

```java
private byte[] toBinary(String inStr) {
    String sbStr;
    byte[] bytes = new byte[10];
    for(int i=0 ; i < 10; i++){
        sbStr = inStr.substring(2*i,2*i+2);
        int a = Integer.parseInt(sbStr,16);
        if ( a <= 127 ){
            bytes[i] = Byte.parseByte(Integer.toString(a));
        }else{
            bytes[i] = Byte.parseByte(Integer.toString(a - 256 ));
        }
    }
    return bytes;
}

private Timestamp format_CMTTM(String cmtTM) {
    return Timestamp.valueOf(cmtTM.replace('T', ' '));
}

private String format_ROWOP(int rowOp) {
    switch (rowOp)  {
    case Row.InsertOperation : return  "I";
    case Row.DeleteOperation : return  "D";
    case Row.UpdateOperation : return  "U";
    default :                  return  "?";
    }
}
```
private PreparedStatement stmt4;

// setup variables to store change log session number and timestamp //

private byte[] cmtLSN;
private Timestamp cmtTM;

// method to establish connection to DB2 //

public Repldb2() throws Exception {
    cmtLSN = new byte[10];
    System.out.println("url = " + url);
    Class.forName("com.ibm.db2.jcc.DB2Driver").newInstance();
    con = java.sql.DriverManager.getConnection(url, id, pass);
    con.setAutoCommit(false);
    System.out.println("Connect DB");
    stmt1 = con.prepareStatement(sql1);
    stmt2 = con.prepareStatement(sql2);
    stmt3 = con.prepareStatement(sql3);
    stmt4 = con.prepareStatement(sql4);
}

// method to establish two phase commit connection to WMQ and DB2 //

public Repldb2(MQQueueManager qMgr) throws Exception {
    cmtLSN = new byte[10];
    ds = new DB2XADataSource();
    ds.setDatabaseName("CIF");
    con = qMgr.getJDBCConnection(ds, id, pass);
    con.setAutoCommit(false);
    System.out.println("Connect DB");
    stmt1 = con.prepareStatement(sql1);
    stmt2 = con.prepareStatement(sql2);
    stmt3 = con.prepareStatement(sql3);
    stmt4 = con.prepareStatement(sql4);
}

public void commit() throws Exception {
    con.commit();
}

public void rollback() throws Exception {
    con.rollback();
}

public void registerUpdate() throws Exception {
    String DUMMYONE = null;
    DUMMYONE = "ABC";
}

// method to perform data handling //
// 1. get the commit sequence and the commit timestamp from the message //
// 2. for each row, determine the table name, and call the method for that table //

public boolean ccdInsert(Msg pubMsg) throws Exception {
    TransactionMsg tranMsg = (TransactionMsg) pubMsg;
    cmtLSN = format_CMTSEQ(tranMsg.getCommitLSN());
cmtTM = format_CMTTM(tranMsg.getCommitTime());

java.util.Vector rows = tranMsg.getRows();
for (int i=0; i < rows.size(); i++) {
    Row row = (Row) rows.elementAt(i);
    if (row.getSrcName().equals("CUSTOMER")) {
        applycustomer(i,row);
    } else if (row.getSrcName().equals("POLICY")) {
        applypolicy(i,row);
    } else if (row.getSrcName().equals("PREMIUM")) {
        applypremium(i,row);
    } else if (row.getSrcName().equals("CLAIM")) {
        applyclaim(i,row);
    }
}
return tranMsg.isLast();

private void applycustomer(int i, Row row) throws Exception {
    // method to perform data handling for a customer record //
    // 1. set all before-image and after-image variables to null //
    // 2. use XML Publication Toolkit row and column methods to //
    //    - determine the row operation type I U or D //
    //    - set the before-image and after-image variable for each column //
    // call the CIF database stored procedure to handle the change row //
    Integer CUSTOMER_NUMBER = null;
    String FIRST_NAME = null;
    String LAST_NAME = null;
    String ADDRESS_LINE1 = null;
    String ADDRESS_LINE2 = null;
    String ADDRESS_LINE3 = null;
    String ADDRESS_LINE4 = null;
    String ZIPCODE = null;
    Integer XCUSTOMER_NUMBER = null;
    String XFIRST_NAME = null;
    String XLAST_NAME = null;
    String XADDRESS_LINE1 = null;
    String XADDRESS_LINE2 = null;
    String XADDRESS_LINE3 = null;
    String XADDRESS_LINE4 = null;
    String XZIPCODE = null;
    String rowOp = format_ROWOP(row.getRowOperation());
    //    byte[] intSEQ = new byte[10];
    //    intSEQ = format_INTSEQ(i);
    java.util.Vector cols = row.getColumns();
    for (int j=0; j < cols.size(); j++) {
        Column col = (Column) cols.elementAt(j);
        String colName = col.getName();
        if (colName.equals("CUSTOMER_NUMBER")) {
            CUSTOMER_NUMBER = (Integer) col.getValue();
        } else if (colName.equals("XCUSTOMER_NUMBER")) {
            XCUSTOMER_NUMBER = (Integer) col.getValue();
        } else if (colName.equals("FIRST_NAME")) {
            FIRST_NAME = (String) col.getValue();
        }
    }
}
else if (colName.equals("XFIRST_NAME"))
    XFIRST_NAME = (String) col.getValue();
else if (colName.equals("LAST_NAME")){
    LAST_NAME = (String) col.getValue();
}else if (colName.equals("XLAST_NAME")){
    XLAST_NAME = (String) col.getValue();
}else if (colName.equals("ADDRESS_LINE1")){
    ADDRESS_LINE1 = (String) col.getValue();
}else if (colName.equals("XADDRESS_LINE1")){
    XADDRESS_LINE1 = (String) col.getValue();
}else if (colName.equals("ADDRESS_LINE2")){
    ADDRESS_LINE2 = (String) col.getValue();
}else if (colName.equals("XADDRESS_LINE2")){
    XADDRESS_LINE2 = (String) col.getValue();
}else if (colName.equals("ADDRESS_LINE3")){
    ADDRESS_LINE3 = (String) col.getValue();
}else if (colName.equals("XADDRESS_LINE3")){
    XADDRESS_LINE3 = (String) col.getValue();
}else if (colName.equals("ADDRESS_LINE4")){
    ADDRESS_LINE4 = (String) col.getValue();
}else if (colName.equals("XADDRESS_LINE4")){
    XADDRESS_LINE4 = (String) col.getValue();
}else if (colName.equals("ZIPCODE")){
    ZIPCODE = (String) col.getValue();
}else if (colName.equals("XZIPCODE")){
    XZIPCODE = (String) col.getValue();
}

stmt1.setString    (1,rowOp);
stmt1.setInt    (2,CUSTOMER_NUMBER.intValue());
stmt1.setString    (3,FIRST_NAME);
stmt1.setString    (4,LAST_NAME);
stmt1.setString    (5,ADDRESS_LINE1);
stmt1.setString    (6,ADDRESS_LINE2);
stmt1.setString    (7,ADDRESS_LINE3);
stmt1.setString    (8,ADDRESS_LINE4);
stmt1.setString    (9,ZIPCODE);
if(XCUSTOMER_NUMBER!=null){
    stmt1.setInt    (10,XCUSTOMER_NUMBER.intValue());
}else{
    stmt1.setNull(10, java.sql.Types.INTEGER);
}

stmt1.setString    (11,XFIRST_NAME);
stmt1.setString    (12,XLAST_NAME);
stmt1.setString    (13,XADDRESS_LINE1);
stmt1.setString    (14,XADDRESS_LINE2);
stmt1.setString    (15,XADDRESS_LINE3);
stmt1.setString    (16,XADDRESS_LINE4);
stmt1.setString    (17,XZIPCODE);

stmt1.executeUpdate();
System.out.println("i = " + i);
private void applyPolicy(int i, Row row) throws Exception {

    // method to perform data handling for a policy record //
    // 1. set all before-image and after-image variables to null //
    // 2. use XML Publication Toolkit row and column methods to //
    //    - determine the row operation type I U or D //
    //    - set the before-image and after-image variable for each column //
    // call the CIF database stored procedure to handle the change row //

    Integer CUSTOMER_NUMBER = null;
    String POLICY_NUMBER = null;
    String POLICY_START = null;
    String POLICY_END = null;
    String POLICY_SUBTYPE = null;
    java.math.BigDecimal POLICY_EXCESS = null;
    Integer XCUSTOMER_NUMBER = null;
    String XPOLICY_NUMBER = null;
    String XPOLICY_START = null;
    String XPOLICY_END = null;
    String XPOLICY_SUBTYPE = null;
    java.math.BigDecimal XPOLICY_EXCESS = null;

    String rowOp = format_ROWOP(row.getRowOperation());
    byte[] intSEQ = new byte[10];
    //
    // byte[] intSEQ = format_INTSEQ(i);
    //
    java.util.Vector cols = row.getColumns();
    for (int j=0; j < cols.size(); j++) {
        Column col = (Column) cols.elementAt(j);
        String colName = col.getName();
        if (colName.equals("CUSTOMER_NUMBER")) {
            CUSTOMER_NUMBER = (Integer) col.getValue();
        } else if (colName.equals("XCUSTOMER_NUMBER")) {
            XCUSTOMER_NUMBER = (Integer) col.getValue();
        } else if (colName.equals("POLICY_NUMBER")) {
            POLICY_NUMBER = (String) col.getValue();
        } else if (colName.equals("XPOLICY_NUMBER")) {
            XPOLICY_NUMBER = (String) col.getValue();
        } else if (colName.equals("POLICY_START")) {
            POLICY_START = (String) col.getValue();
        } else if (colName.equals("XPOLICY_START")) {
            XPOLICY_START = (String) col.getValue();
        } else if (colName.equals("POLICY_END")) {
            POLICY_END = (String) col.getValue();
        } else if (colName.equals("XPOLICY_END")) {
            XPOLICY_END = (String) col.getValue();
        } else if (colName.equals("POLICY_SUBTYPE")) {
            POLICY_SUBTYPE = (String) col.getValue();
        } else if (colName.equals("XPOLICY_SUBTYPE")) {
            XPOLICY_SUBTYPE = (String) col.getValue();
        } else if (colName.equals("POLICY_EXCESS")) {
            POLICY_EXCESS = (java.math.BigDecimal) col.getValue();
        } else if (colName.equals("XPOLICY_EXCESS")) {
            XPOLICY_EXCESS = (java.math.BigDecimal) col.getValue();
        }
    }
}
stmt2.setString (1,rowOp);
stmt2.setInt   (2,CUSTOMER_NUMBER.intValue());
stmt2.setString    (3,POLICY_NUMBER);
stmt2.setString    (4,POLICY_START);
stmt2.setString    (5,POLICY_END);
stmt2.setString    (6,POLICY_SUBTYPE);
stmt2.setBigDecimal    (7,POLICY_EXCESS);

if(XCUSTOMER_NUMBER!=null){
  stmt2.setInt   (8,XCUSTOMER_NUMBER.intValue());
}else{
  stmt2.setNull(8, java.sql.Types.INTEGER);
}

stmt2.setString    (9,XPOLICY_NUMBER);
stmt2.setString    (10,XPOLICY_START);
stmt2.setString    (11,XPOLICY_END);
stmt2.setString    (12,XPOLICY_SUBTYPE);
stmt2.setBigDecimal    (13,XPOLICY_EXCESS);

stmt2.executeUpdate();
System.out.println("i = " + i);

private void applypremium(int i,Row row) throws Exception {

// method to perform data handling for a premium record //
// 1. set all before-image and after-image variables to null //
// 2. use XML Publication Toolkit row and column methods to //
//    - determine the row operation type I U or D //
//    - set the before-image and after-image variable for each column //
// call the CIF database stored procedure to handle the change row //

Integer CUSTOMER_NUMBER   = null;
String POLICY_NUMBER  = null;
String RISK_LINE_NUMBER  = null;
java.math.BigDecimal RISK_LINE_PREMIUM   = null;
Integer XCUSTOMER_NUMBER   = null;
String XPOLICY_NUMBER  = null;
String XRISK_LINE_NUMBER  = null;
java.math.BigDecimal XRISK_LINE_PREMIUM   = null;
String rowOp  = format_ROWOP(row.getRowOperation());
//     byte[][] intSEQ = new byte[10];
//            intSEQ = format_INTSEQ(i);
java.util.Vector cols = row.getColumns();
for (int j=0; j < cols.size(); j++) {
  Column col = (Column) cols.elementAt(j);
  String colName = col.getName();
  if       (colName.equals("CUSTOMER_NUMBER")){
    CUSTOMER_NUMBER = (Integer) col.getValue();
  }else if (colName.equals("XCUSTOMER_NUMBER")){
    XCUSTOMER_NUMBER = (Integer) col.getValue();
  }else if (colName.equals("POLICY_NUMBER")){
    POLICY_NUMBER = (String) col.getValue();
  }else if (colName.equals("RISK_LINE_NUMBER")){
    RISK_LINE_NUMBER = (String) col.getValue();
  }else if (colName.equals("RISK_LINE_PREMIUM")){
    RISK_LINE_PREMIUM = (BigDecimal) col.getValue();
  }
}

// System.err.println("CUSTOMER_NUMBER: "+CUSTOMER_NUMBER);
// System.err.println("XCUSTOMER_NUMBER: "+XCUSTOMER_NUMBER);
// System.err.println("POLICY_NUMBER: "+POLICY_NUMBER);
// System.err.println("RISK_LINE_NUMBER: "+RISK_LINE_NUMBER);
// System.err.println("RISK_LINE_PREMIUM: "+RISK_LINE_PREMIUM);
// System.err.println("rowOp: "+rowOp);
// System.err.println("intSEQ: "+intSEQ);
// System.err.println("cols: "+cols);
}


```java
private void applyClaim(int i, Row row) throws Exception {
    // method to perform data handling for a claim record 
    // 1. set all before-image and after-image variables to null 
    // 2. use XML Publication Toolkit row and column methods to 
    //    - determine the row operation type I U or D 
    //    - set the before-image and after-image variables for each column 
    // call the CIF database stored procedure to handle the change row 

    Integer CUSTOMER_NUMBER = null;
    String POLICY_NUMBER = null;
    Short CLAIM_ID = null;
    String ACCIDENT_DATE = null;
    String CLAIM_DATE = null;
    java.math.BigDecimal CLAIM_AMOUNT = null;
    Integer XCUSTOMER_NUMBER = null;
    String XPOLICY_NUMBER = null;
    Short XCLAIM_ID = null;
    String XACCIDENT_DATE = null;
    String XCLAIM_DATE = null;
    java.math.BigDecimal XCLAIM_AMOUNT = null;

    else if (colName.equals("XPOLICY_NUMBER")) {
        XPOLICY_NUMBER = (String) col.getValue();
    } else if (colName.equals("RISK_LINE_NUMBER")) {
        RISK_LINE_NUMBER = (String) col.getValue();
    } else if (colName.equals("XRISK_LINE_NUMBER")) {
        XRISK_LINE_NUMBER = (String) col.getValue();
    } else if (colName.equals("RISK_LINE_PREMIUM")) {
        RISK_LINE_PREMIUM = (java.math.BigDecimal) col.getValue();
    } else if (colName.equals("XRISK_LINE_PREMIUM")) {
        XRISK_LINE_PREMIUM = (java.math.BigDecimal) col.getValue();
    }

    stmt3.setString (1, rowOp);
    stmt3.setInt (2, CUSTOMER_NUMBER.intValue());
    stmt3.setString (3, POLICY_NUMBER);
    stmt3.setString (4, RISK_LINE_NUMBER);
    stmt3.setBigDecimal (5, RISK_LINE_PREMIUM);

    if (XCUSTOMER_NUMBER != null) {
        stmt3.setInt (6, XCUSTOMER_NUMBER.intValue());
    } else {
        stmt3.setNull (6, java.sql.Types.INTEGER);
    }

    stmt3.setString (7, XPOLICY_NUMBER);
    stmt3.setString (8, XRISK_LINE_NUMBER);
    stmt3.setBigDecimal (9, XRISK_LINE_PREMIUM);

    stmt3.executeUpdate();
    System.out.println("i = " + i);
}
```
String rowOp = format_ROWOP(row.getRowOperation());
// byte[] intSEQ = new byte[10];
// intSEQ = format_INTSEQ(i);

java.util.Vector cols = row.getColumns();
for (int j=0; j < cols.size(); j++) {
    Column col = (Column) cols.elementAt(j);
    String colName = col.getName();
    if (colName.equals("CUSTOMER_NUMBER")){
        CUSTOMER_NUMBER = (Integer) col.getValue();
    }else if (colName.equals("XCUSTOMER_NUMBER")){
        XCUSTOMER_NUMBER = (Integer) col.getValue();
    }else if (colName.equals("POLICY_NUMBER")){
        POLICY_NUMBER = (String) col.getValue();
    }else if (colName.equals("XPOLICY_NUMBER")){
        XPOLICY_NUMBER = (String) col.getValue();
    }else if (colName.equals("CLAIM_ID")){
        CLAIM_ID = (Short) col.getValue();
    }else if (colName.equals("XCLAIM_ID")){
        XCLAIM_ID = (Short) col.getValue();
    }else if (colName.equals("ACCIDENT_DATE")){
        ACCIDENT_DATE = (String) col.getValue();
    }else if (colName.equals("XACCIDENT_DATE")){
        XACCIDENT_DATE = (String) col.getValue();
    }else if (colName.equals("CLAIM_DATE")){
        CLAIM_DATE = (String) col.getValue();
    }else if (colName.equals("XCLAIM_DATE")){
        XCLAIM_DATE = (String) col.getValue();
    }else if (colName.equals("CLAIM_AMOUNT")){
        CLAIM_AMOUNT = (java.math.BigDecimal) col.getValue();
    }else if (colName.equals("XCLAIM_AMOUNT")){
        XCLAIM_AMOUNT = (java.math.BigDecimal) col.getValue();
    }
}

stmt4.setString    (1,rowOp);
stmt4.setInt    (2,CUSTOMER_NUMBER.intValue());
stmt4.setString    (3,POLICY_NUMBER);
stmt4.setShort    (4,CLAIM_ID.shortValue());
stmt4.setString    (5,ACCIDENT_DATE);
stmt4.setString    (6,CLAIM_DATE);
stmt4.setBigDecimal    (7,CLAIM_AMOUNT);
if(XCUSTOMER_NUMBER!=null){
    stmt4.setInt    (8,XCUSTOMER_NUMBER.intValue());
}else{
    stmt4.setNull(8, java.sql.Types.INTEGER);
}
stmt4.setString    (9,XPOLICY_NUMBER);
if(XCLAIM_ID!=null){
    stmt4.setShort    (10,XCLAIM_ID.shortValue());
}else{
    stmt4.setNull(10, java.sql.Types.INTEGER);
}
stmt4.setString(11, XACCIDENT_DATE);
stmt4.setString(12, XCLAIM_DATE);
stmt4.setBigDecimal(13, XCLAIM_AMOUNT);

stmt4.executeUpdate();
System.out.println("i = " + i);
}

private byte[] format_CMTSEQ(String cmtLSN) {
    String s1 = cmtLSN.replaceAll(":" , "");
    return toBinary(s1);
}

private byte[] format_INTSEQ(int intSEQ) {
    String s1 = Integer.toString(intSEQ).trim();
    int ln = 20 - s1.length();
    for (int i = 1; i <= ln ; i++) {
        s1 = "0" + s1;
    }
    return toBinary(s1);
}

private byte[] toBinary(String inStr) {
    String sbStr;
    byte[] bytes = new byte[10];
    for(int i=0; i < 10; i++) {
        sbStr = inStr.substring(2*i, 2*i+2);
        int a = Integer.parseInt(sbStr, 16);
        if (a <= 127) {
            bytes[i] = Byte.parseByte(Integer.toString(a));
        } else {
            bytes[i] = Byte.parseByte(Integer.toString(a - 256));
        }
    }
    return bytes;
}

private Timestamp format_CMTTM(String cmtTM) {
    return Timestamp.valueOf(cmtTM.replace('T', ' '));
}

private String format_ROWOP(int rowOp) {
    switch (rowOp) {
    case Row.InsertOperation: return "I";
    case Row.DeleteOperation: return "D";
    case Row.UpdateOperation: return "U";
    default: return "?";
    }
}

Example 3-28  mqldb2g class

import com.ibm.mq.*;
import com.ibm.db2.tools.repl.publication.*;
import com.ibm.db2.tools.repl.publication.support.*;
class mqldb2g{
    public static void main(String args[]) throws Exception{

        MQQueueManager qmgr = null;
        MQQueue q = null;
        MQMessage getMessage = null;
        MQGetMessageOptions pmo = null;

        try{
            // This message handler class performs the following functions /
            // 1. establish a connection to a Message Queue that receives XML Publications /
            // 2. listen for new messages continuously (subject to a 120 second timeout) /
            // 3. parse the message using the XML Publication toolkit /
            // 4. determine if the message is transactional XML Event publication message /
            // 5. If it is transactional XML Event publication message, call the replims data
            // handler class /
            // see Websphere MQ publication SC34-6066 "Using Java" for programming details for
            // Webphere MQ /

            // define a parser class using the xml Publication Toolkit PublicationParser /
            PublicationParser parser = new PublicationParser();

            // create a connection to the Queue Manager /
            java.util.Hashtable mqProperties = new java.util.Hashtable();
            mqProperties.put(MQC.THREAD_AFFINITY, new Boolean(true));
            qmgr = new MQQueueManager("QMCIF",mqProperties);

            // establish a two phase commit connect to the Queue manager, and the XA database
            resource (CIF) /
            Repldb2 repldb2 = new Repldb2(qmgr);

            // setup the options for the queue we wish to open, including a 120 second timeout /
            int openOption = MQC.MQOO_INPUT_AS_Q_DEF;
            q = qmgr.accessQueue("DB2G_RECVQ1", openOption);
            getMessage= new MQMessage();
            MQGetMessageOptions mqgmo = new MQGetMessageOptions();
            mqgmo.options = MQC.MQGMO_WAIT | MQC.MQGMO_SYNCPOINT;
            mqgmo.matchOptions = MQC.MQMO_NONE ;
            mqgmo.waitInterval = 120000; // 120 sec.

            // begin an MQ logical unit of work /
            qmgr.begin();

            // loop whilst a get message can be retrieved within the 120 second timeout interval /
            while(true){
                // get next message off the queue /
                getMessage.clearMessage();
                q.get(getMessage, mqgmo);

                // get message length and print the message to std out /
        }
    }
}

Publishing IMS and DB2 Data using WebSphere Information Integrator: Configuration and Monitoring Guide
String xmlMsg = getMessage.readString (getMessage.getMessageLength());
System.out.println( xmlMsg );

// parse message using the WMQ xml parser //

Msg msg = parser.parse(xmlMsg);
if (msg instanceof TransactionMsg){
  // if it is a transactional message call class replims using method ccdinsert, then
  commit and read the next message //
  if (repldb2.ccdInsert(msg)) {
    repldb2.registerUpdate();
    qmgr.commit();
    qmgr.begin();
  }
  // if it is not a transactional message (maybe a rowop message) do not process the
  message - just commit and read the next message //
} else {
  qmgr.commit();
  qmgr.begin();
}
}
} catch(MQException ex) {
  System.out.println("MQException occurred
cc:" + ex.completionCode + "
rc:" + ex.reasonCode);
  ex.printStackTrace();
}finally{
  try{
    qmgr.backout();
  }catch(Exception e){
    e.printStackTrace();
  }
}

Example 3-29  DB2 Stored Procedure SPIMSKCUSTOMER

```
CREATE PROCEDURE DB2ADMIN.SPIMSKCUSTOMER
( IN rowop CHAR(1),
  V_CUSTKEY CHAR(10), V_FIRSTNME CHAR(20), V_LASTNAME CHAR(25),
  VX_CUSTKEY CHAR(10), VX_FIRSTNME CHAR(20), VX_LASTNAME CHAR(25) )
LANGUAGE SQL
BEGIN
  DECLARE SQLCODE INT;--
  DECLARE RC INT;--
  declare V_CIF_CUSTOMER_NUMBER int--;--

  DECLARE CONTINUE HANDLER FOR SQLSTATE '23505' SET RC = SQLCODE;--
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET RC = SQLCODE;--

  -- Lookup the ODS common customer number
  set V_CIF_CUSTOMER_NUMBER = (select cif_customer_number from CIF.CUSTLINK where
    IMSK_CUSTKEY = V_CUSTKEY );--

  --Action on Insert to IMSK CUSTOMER SEGMENT : None
  --The common customer identification process is outside the scope of these examples.
  --It is assumed that another process maintains the CIF.CUSTLINK Table in the ODS
```
-- Action on Update to IMSK CUSTOMER SEGMENT: Apply Update to the CIF.CUSTOMER table  
IF rowop = 'U' THEN  
  UPDATE CIF.CUSTOMER SET  
    FIRST_NAME = V_FIRSTNAME, LAST_NAME = V_LASTNAME  
    WHERE CIF_CUSTOMER_NUMBER = V_CIF_CUSTOMER_NUMBER;--  
END IF;--

-- Action on Delete to IMSK CUSTOMER SEGMENT: Perform Logical delete: Mark the "IMSK customer" as logically deleted  
IF rowop = 'D' THEN  
  UPDATE CIF.CUSTOMER SET DELIMSK_IND = 'Y' WHERE CIF_CUSTOMER_NUMBER = V_CIF_CUSTOMER_NUMBER;--  
END IF;--  
END ;

Example 3-30  DB2 Stored Procedure SPIMSKADDRESS

CREATE PROCEDURE DB2ADMIN.SPIMSKADDRESS  
( IN rowop CHAR(1),  
  V_CUSTKEY CHAR(10), V_ADDRTYPE CHAR(1),  
  V_ADDRESS1 CHAR(25), V_ADDRESS2 CHAR(25), V_ADDRESS3 CHAR(25), V_ZIPCODE CHAR(10),  
  VX_CUSTKEY CHAR(10), VX_ADDRTYPE CHAR(1),  
  VX_ADDRESS1 CHAR(25), VX_ADDRESS2 CHAR(25), VX_ADDRESS3 CHAR(25), VX_ZIPCODE CHAR(10) )  
LANGUAGE SQL  
BEGIN  
  DECLARE SQLCODE INT;--  
  DECLARE RC INT;--  
  declare V_CIF_CUSTOMER_NUMBER int;--  

  DECLARE CONTINUE HANDLER FOR SQLSTATE '23505' SET RC = SQLCODE;--  
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET RC = SQLCODE;--

  -- Lookup the ODS common customer number  
  set V_CIF_CUSTOMER_NUMBER = (select cif_customer_number from CIF.CUSTLINK where  
    IMSK_CUSTKEY = V_CUSTKEY );--

  -- Action on Insert to the IMSK ADDRESS SEGMENT: Update CIF.CUSTOMER table with new address field values  
  IF rowop = 'I' THEN  
    UPDATE CIF.CUSTOMER SET  
      ADDRESS_LINE1 = V_ADDRESS1, ADDRESS_LINE2 = V_ADDRESS2, ADDRESS_LINE3 = V_ADDRESS3,  
      ADDRESS_LINE4 = NULL, ZIPCODE = V_ZIPCODE  
      WHERE CIF_CUSTOMER_NUMBER = V_CIF_CUSTOMER_NUMBER;--  
  END IF;--

  -- Action on Update to the IMSK ADDRESS SEGMENT: Apply Update to the CIF.CUSTOMER table  
  IF rowop = 'U' THEN  
    UPDATE CIF.CUSTOMER SET  
      ADDRESS_LINE1 = V_ADDRESS1, ADDRESS_LINE2 = V_ADDRESS2, ADDRESS_LINE3 = V_ADDRESS3,  
      ADDRESS_LINE4 = NULL, ZIPCODE = V_ZIPCODE  
      WHERE CIF_CUSTOMER_NUMBER = V_CIF_CUSTOMER_NUMBER;--  
  END IF;--

  -- Action on Delete to the IMSK ADDRESS SEGMENT: Perform Logical delete: Mark the "IMSK customer" as logically deleted  
  IF rowop = 'D' THEN  
    UPDATE CIF.CUSTOMER SET DELIMSK_IND = 'Y' WHERE CIF_CUSTOMER_NUMBER = V_CIF_CUSTOMER_NUMBER;--  
  END IF;--
Example 3-31  DB2 Stored Procedure SPIMSKPOLICY

drop procedure DB2ADMIN.SPIMSKPOLICY;
CREATE PROCEDURE DB2ADMIN.SPIMSKPOLICY
(IN rowop CHAR(1),
V_CUSTKEY CHAR(10), V_POLICYNM CHAR(10), V_POLYTYPE CHAR(3),
V_POLCOVER CHAR(6), V_POLSTDTE DATE, V_POLNOTES CHAR(200),
VX_CUSTKEY CHAR(10), VX_POLICYNM CHAR(10), VX_POLYTYPE CHAR(3),
VX_POLCOVER CHAR(6), VX_POLSTDTE DATE, VX_POLNOTES CHAR(200))
LANGUAGE SQL
BEGIN
DECLARE SQLCODE INT;--
DECLARE RC INT;--
declare V_CIF_CUSTOMER_NUMBER int;--
declare V_POLDATE DATE;--

DECLARE CONTINUE HANDLER FOR SQLSTATE '23505' SET RC = SQLCODE;--
DECLARE CONTINUE HANDLER FOR NOT FOUND SET RC = SQLCODE;--

-- MARKER - PENDING VIEW OF DATE FIELD FORMAT IN NAGRAJ IMS DATABASE
-- set V_POLDATE = ( values current date );--

-- Lookup the ODS common customer number
set V_CIF_CUSTOMER_NUMBER = ( select cif_customer_number from CIF.CUSTLINK where
IMSK_CUSTKEY = V_CUSTKEY );--

-- Action on Insert to IMSK Policy SEGM: Insert Row into CIF.POLICY TABLE
-- If a row already exists, update it with the new data values
IF rowop = 'I' THEN
    INSERT INTO CIF.POLICY (CIF_CUSTOMER_NUMBER, SOURCE_SYSTEM, POLICY_NUMBER, POLICY_START,
DELIMSK_IND, DELDB2G_IND)
    VALUES (V_CIF_CUSTOMER_NUMBER, 'IMSK', V_POLICYNM, V_POLDATE, NULL, NULL);--
END IF;--
IF RC=-803 THEN
    UPDATE CIF.POLICY SET
    POLICY_START = V_POLDATE, DELIMSK_IND = NULL, DELDB2G_IND = NULL
    WHERE CIF_CUSTOMER_NUMBER = V_CIF_CUSTOMER_NUMBER AND SOURCE_SYSTEM = 'IMSK' AND
    POLICY_NUMBER = V_POLICYNM;--
END IF;--

-- Action on Update to IMSK Policy SEGM: Apply Update to the CIF.POLICY table
-- If the row does not exist, create it with the new data values
IF rowop = 'U' THEN
    UPDATE CIF.POLICY SET
    POLICY_START = V_POLDATE, DELIMSK_IND = NULL, DELDB2G_IND = NULL
    WHERE CIF_CUSTOMER_NUMBER = V_CIF_CUSTOMER_NUMBER AND SOURCE_SYSTEM = 'IMSK' AND
    POLICY_NUMBER = V_POLICYNM;--
END IF;--
IF RC=100 THEN
    INSERT INTO CIF.POLICY (CIF_CUSTOMER_NUMBER, SOURCE_SYSTEM, POLICY_NUMBER, POLICY_START,
DELIMSK_IND, DELDB2G_IND)
    VALUES (V_CIF_CUSTOMER_NUMBER, 'IMSK', V_POLICYNM, V_POLDATE, NULL, NULL);--
END IF;--

-- Action on Delete to IMSK Policy SEGM: Perform Logical delete : Mark the "IMSK customer"
as logically deleted
IF rowop = 'D' THEN
   UPDATE CIF.POLICY SET DELIMSK_IND = 'Y'
       WHERE CIF_CUSTOMER_NUMBER = V_CIF_CUSTOMER_NUMBER AND SOURCE_SYSTEM = 'IMSK' AND
       POLICY_NUMBER  = V_POLICYNM;--
END IF;--

END ;

Example 3-32   DB2 Stored Procedure SPIMSKCLAIM

drop procedure DB2ADMIN.SPIMSKCLAIM;
CREATE PROCEDURE DB2ADMIN.SPIMSKCLAIM
(IN rowop CHAR(1),
 V_CUSTKEY CHAR(10), V_POLICYNM CHAR(10), V_CLAIMNM CHAR(10),
 V_CLAIMDATE DATE, V_CLAIMAMT DECIMAL(8,2), V_CLAIMDET CHAR(200),
 VX_CUSTKEY CHAR(10), VX_POLICYNM CHAR(10), VX_CLAIMNM CHAR(10),
 VX_CLAIMDATE DATE, VX_CLAIMAMT DECIMAL(8,2), VX_CLAIMDET CHAR(200) )
LANGUAGE SQL
BEGIN
   DECLARE SQLCODE INT;--
   DECLARE RC   INT;--
   declare V_CIF_CUSTOMER_NUMBER int;--
   declare V_CLAIMDATE DATE;--

   DECLARE CONTINUE HANDLER FOR SQLSTATE '23505' SET RC = SQLCODE;--
   DECLARE CONTINUE HANDLER FOR NOT FOUND SET RC = SQLCODE;--

   -- MARKER - PENDING VIEW OF DATE FIELD FORMAT IN NAGRAJ IMS DATABASE
   -- set V_CLAIMDATE = ( values current date );--

   -- Lookup the ODS common customer number
   set V_CIF_CUSTOMER_NUMBER = (select cif_customer_number from CIF.CUSTLINK where
      IMSK_CUSTKEY = V_CUSTKEY );--

   --Action on Insert to IMSK Claim SEGM : Insert Row into CIF.CLAIM TABLE
   --If a row already exists, update it with the new data values
   IF rowop = 'I' THEN
      INSERT INTO CIF.CLAIM (CIF_CUSTOMER_NUMBER, SOURCE_SYSTEM, POLICY_NUMBER, CLAIM_NUMBER,
      CLAIM_DATE, CLAIM_AMOUNT, DELIMSK_IND, DELDB2G_IND)
      VALUES (V_CIF_CUSTOMER_NUMBER, 'IMSK', V_POLICYNM, V_CLAIMNM, V_CLAIMDATE, V_CLAIMAMT,
      NULL, NULL);--
   END IF;--
   IF RC=-803 THEN
      UPDATE CIF.CLAIM SET
         CLAIM_DATE = V_CLAIMDATE, CLAIM_AMOUNT = V_CLAIMAMT, DELIMSK_IND = NULL, DELDB2G_IND =
      NULL
      WHERE CIF_CUSTOMER_NUMBER = V_CIF_CUSTOMER_NUMBER AND SOURCE_SYSTEM = 'IMSK' AND
      POLICY_NUMBER  = V_POLICYNM AND CLAIM_NUMBER = V_CLAIMNM;--
   END IF;--

   --Action on Update to IMSK Claim SEGM: Apply Update to the CIF.CLAIM table
   --if row not found, then insert a row with the new data values
   IF rowop = 'U' THEN
      UPDATE CIF.CLAIM SET
         CLAIM_DATE = V_CLAIMDATE, CLAIM_AMOUNT = V_CLAIMAMT, DELIMSK_IND = NULL, DELDB2G_IND =
      NULL
      WHERE CIF_CUSTOMER_NUMBER = V_CIF_CUSTOMER_NUMBER AND SOURCE_SYSTEM = 'IMSK' AND
      POLICY_NUMBER  = V_POLICYNM AND CLAIM_NUMBER = V_CLAIMNM;--
   END IF;--
   IF RC= 100 THEN
INSERT INTO CIF.CLAIM (CIF_CUSTOMER_NUMBER, SOURCE_SYSTEM, POLICY_NUMBER, CLAIM_NUMBER, CLAIM_DATE, CLAIM_AMOUNT, DELIMSK_IND, DELDB2G_IND) VALUES (V_CIF_CUSTOMER_NUMBER, 'IMSK', V_POLICYNM, V_CLAIMNM, V_CLAIMDATE, V_CLAIMAMT, NULL, NULL);--
END IF;--

--Action on Delete to IMSK Claim SEGM: Perform Logical delete : Mark the "DB2G customer" as logically deleted
IF rowop = 'D' THEN
    UPDATE CIF.CLAIM SET DELIMSK_IND = 'Y'
    WHERE CIF_CUSTOMER_NUMBER = V_CIF_CUSTOMER_NUMBER AND SOURCE_SYSTEM = 'IMSK' AND POLICY_NUMBER = V_POLICYNM AND CLAIM_NUMBER = V_CLAIMNM;--
END IF;--
END  ;

Example 3-33  DB2 Stored Procedure SPIMSKPREMIUM

drop procedure DB2ADMIN.SPIMSKPREMIUM;
CREATE PROCEDURE DB2ADMIN.SPIMSKPREMIUM
  ( IN rowop CHAR(1),
    V_CUSTKEY CHAR(10), V_POLICYNM CHAR(10), V_RISKLINM CHAR(2), V_RPREMIUM DECIMAL(6,2),
    VX_CUSTKEY CHAR(10), VX_POLICYNM CHAR(10), VX_RISKLINM CHAR(2), VX_RPREMIUM DECIMAL(6,2) )
LANGUAGE SQL
BEGIN
DECLARE SQLCODE      INT;--
DECLARE RC           INT;--
declare V_CIF_CUSTOMER_NUMBER int;--
DECLARE CONTINUE HANDLER FOR SQLSTATE '23505' SET RC = SQLCODE;--
DECLARE CONTINUE HANDLER FOR NOT FOUND SET RC = SQLCODE;--

-- Lookup the ODS common customer number
set V_CIF_CUSTOMER_NUMBER = (select cif_customer_number from CIF.CUSTLINK where IMSK_CUSTKEY = V_CUSTKEY );--

--Action on Insert to IMSK Premium SEGM : Insert Row into CIF.PREMIUM TABLE
--If a row already exists, update it with the new data values
IF rowop = 'I' THEN
    INSERT INTO CIF.PREMIUM (CIF_CUSTOMER_NUMBER, SOURCE_SYSTEM, POLICY_NUMBER, RISK_LINE_NUMBER, RISK_LINE_PREMIUM, DELIMSK_IND, DELDB2G_IND)
    VALUES (V_CIF_CUSTOMER_NUMBER, 'IMSK', V_POLICYNM, V_RISKLINM, V_RPREMIUM, NULL, NULL);--
END IF;--
IF RC=-803 THEN
    UPDATE CIF.PREMIUM SET
        RISK_LINE_NUMBER = V_RISKLINM, RISK_LINE_PREMIUM = V_RPREMIUM, DELIMSK_IND  = NULL,
        DELDB2G_IND = NULL
    WHERE CIF_CUSTOMER_NUMBER = V_CIF_CUSTOMER_NUMBER AND SOURCE_SYSTEM = 'IMSK' AND POLICY_NUMBER = V_POLICYNM AND RISK_LINE_NUMBER = V_RISKLINM;--
END IF;--

--Action on Update to IMSK Premium SEGM: Apply Update to the CIF.CUSTOMER table
--if row not found, then insert a row with the new data values
IF rowop = 'U' THEN
    UPDATE CIF.PREMIUM SET
        RISK_LINE_NUMBER = V_RISKLINM, RISK_LINE_PREMIUM = V_RPREMIUM, DELIMSK_IND  = NULL,
        DELDB2G_IND = NULL
    WHERE CIF_CUSTOMER_NUMBER = V_CIF_CUSTOMER_NUMBER AND SOURCE_SYSTEM = 'IMSK' AND POLICY_NUMBER = V_POLICYNM AND RISK_LINE_NUMBER = V_RISKLINM;--
END IF;--

--Action on Delete to IMSK Claim SEGM: Perform Logical delete : Mark the "DB2G customer" as logically deleted
IF rowop = 'D' THEN
    UPDATE CIF.CLAIM SET DELIMSK_IND = 'Y'
    WHERE CIF_CUSTOMER_NUMBER = V_CIF_CUSTOMER_NUMBER AND SOURCE_SYSTEM = 'IMSK' AND POLICY_NUMBER = V_POLICYNM AND CLAIM_NUMBER = V_CLAIMNM;--
END IF;--
END  ;
WHERE CIF_CUSTOMER_NUMBER = V_CIF_CUSTOMER_NUMBER AND SOURCE_SYSTEM = 'IMSK' AND POLICY_NUMBER = V_POLICYNM AND RISK_LINE_NUMBER = V_RISKLINM;--
END IF;--
IF RC= 100 THEN
  INSERT INTO CIF.PREMIUM (CIF_CUSTOMER_NUMBER, SOURCE_SYSTEM, POLICY_NUMBER, RISK_LINE_NUMBER, RISK_LINE_PREMIUM, DELIMSK_IND, DELDB2G_IND)
  VALUES (V_CIF_CUSTOMER_NUMBER, 'IMSK', V_POLICYNM, V_RISKLINM, V_RPREMIUM, NULL, NULL);--
END IF;--

--Action on Delete to IMSK Premium SEG: Perform Logical delete : Mark the "DB2G customer" as logically deleted
IF rowop = 'D' THEN
  UPDATE CIF.PREMIUM SET DELIMSK_IND = 'Y'
  WHERE CIF_CUSTOMER_NUMBER = V_CIF_CUSTOMER_NUMBER AND SOURCE_SYSTEM = 'IMSK' AND POLICY_NUMBER = V_POLICYNM AND RISK_LINE_NUMBER = V_RISKLINM;--
END IF;--
END  ;

---

**Attention:** The CIF Updater program is operational when the MQ Listener class mqldb2g.class (Example 3-28 on page 179) is started.

### Event Notification programs

Event Notification programs can be used to perform many functions, including:

- Automatic generation of an e-mail or letter based on captured data changes.
- Updating of another data store with event information such as the CIF Updater program.
- Invoking another application, and pass it parameters derived from the published event

In general, event notification program provide a means to execute a standard business process based on the receipt of a notification of a data change. Event notification processing is particularly appropriate for systems where important events can be detected and processed with very low latency without human intervention.

The event notification programs in this scenario have been developed using Java programming and the XML Event Publishing Toolkit.

Based on detailed analysis of industry and its own fraud case statistics, Trustworthy Insurance designed an event notification program to forestall potential fraud incidents. The objective of the event notification program is to alert a fraud inspector by e-mail when claim requests that fall within certain defined parametric bounds are received — for example, when the number of days between the inception of the policy and the claim request is less than a certain number of days, an e-mail alert should be triggered.

The Java Event Notification program is very similar in structure and content to the CIF Updater programs described in “CIF Updater programs” on page 159. The significant difference is that the Event Notification Program does not connect to the CIF database, but simply reads XML event messages and uses application logic to determine whether certain conditions apply before sending an E-mail.
The Event Notification program functions are implemented as follows:

1. Connect to WebSphere MQ, reading an XML event message from the queue, and parsing the XML message is performed by the mqlimsk2.class, which involves Java programming with XML Publishing Toolkit.

2. Dispatch the message to the application, and perform application-specific code using the Replims2.class — this class identifies the publication that has issued the event, extracts the data changes from the parsed XML message, evaluate the data condition, and send an e-mail if appropriate.

Figure 3-89 describes the event notification program processing claim events originating in the WIICEP for IMS auto insurance environment.

**Attention:** We did not develop the corresponding event notification program for processing claim events originating in the WIIEP for DB2 UDB for z/OS for the home insurance environment due to time constraints. However, the approach would be more or less identical.

The individual classes, stored procedure, and application code are shown in Example 3-34 through Example 3-35.

**Note:** The Event Notification program is downloadable from:
ftp://www.redbooks.ibm.com/redbooks/SG247132

---

Example 3-34  MQ Listener class

```java
import com.ibm.mq.*;
import com.ibm.db2.tools.repl.publication.*;
```
```java
import com.ibm.db2.tools.repl.publication.support.*;

class mqlimsk2{
    public static void main(String args[]) throws Exception{
        MQQueueManager qmgr = null;
        MQQueue q = null;
        MQMessage getMessage = null;
        MQGetMessageOptions pmo = null;

        try{
            PublicationParser parser = new PublicationParser();

            java.util.Hashtable mqProperties = new java.util.Hashtable();
            mqProperties.put(MQC.THREAD_AFFINITY, new Boolean(true));
            qmgr = new MQQueueManager("QMCIF",mqProperties);

            Replims2 replims2 = new Replims2(qmgr);

            int openOption = MQC.MQOO_INPUT_AS_Q_DEF;
            q = qmgr.accessQueue("IMSK_RECVQ2", openOption);

            getMessage= new MQMessage();
            MQGetMessageOptions mqgmo = new MQGetMessageOptions();
            mqgmo.options = MQC.MQGMO_WAIT | MQC.MQGMO_SYNCPOINT;
            mqgmo.matchOptions = MQC.MQMO_NONE ;
            // mqgmo.waitInterval = 60000; // 60 sec.
            mqgmo.waitInterval = 120000; // 120 sec.

            qmgr.begin();
            while(true){
                getMessage.clearMessage();
                q.get(getMessage, mqgmo);
                String xmlMsg = getMessage.readString (getMessage.getMessageLength());
                System.out.println( xmlMsg );
                Msg msg = parser.parse(xmlMsg);
                if (msg instanceof TransactionMsg){
                    if (replims2.applytrans(msg)) {
                        qmgr.commit();
                        qmgr.begin();
                    }
                }else {
                    qmgr.commit();
                    qmgr.begin();
                }
            }
        } catch(MQException ex) {
            System.out.println("MQException occurred\nc:\nrc:" + ex.completionCode + "\nrc:" + ex.reasonCode);
            ex.printStackTrace();
        } finally{
            try{
                qmgr.backout();
            } catch(Exception e){
                e.printStackTrace();
            }
        }
    }
}
```
Example 3-35  Data Handler and Email class

```java
import java.sql.*;
import com.ibm.mq.*;
import javax.transaction.xa.*;
import com.ibm.db2.tools.repl.publication.*;
import com.ibm.db2.jcc.*/;
import java.util.Date;
import java.util.Calendar;
import java.text.SimpleDateFormat;
import java.util.*;
public class Replims2 {

    private String url = "jdbc:db2:CIF";
    private String id = "db2admin";
    private String pass = "l0nepine";
    private Connection con;
    private DB2XADataSource ds;
    private PreparedStatement stmt1;
    private byte[]    cmtLSN;
    private Timestamp cmtTM;

    public Replims2() throws Exception {
        cmtLSN = new byte[10];
        System.out.println("url = " + url);
        Class.forName("com.ibm.db2.jcc.DB2Driver").newInstance();
        con = java.sql.DriverManager.getConnection(url, id, pass);
        con.setAutoCommit(false);
        System.out.println("Connect DB");
    }

    public Replims2(MQQueueManager qMgr) throws Exception {
        cmtLSN = new byte[10];
        ds = new DB2XADataSource();
        ds.setDatabaseName("CIF");
        con = qMgr.getJDBCConnection(ds, id, pass);
        con.setAutoCommit(false);
        System.out.println("Connect DB");
    }

    public void commit() throws Exception {
        con.commit();
    }

    public void rollback() throws Exception {
        con.rollback();
    }

    public boolean applytrans(Msg pubMsg) throws Exception {
        TransactionMsg tranMsg = (TransactionMsg) pubMsg;
        cmtLSN = format_CMTSEQ(tranMsg.getCommitLSN());
        cmtTM  = format_CMTTM(tranMsg.getCommitTime());
        // the log sequence number and timestamp of the commit are not
        // used in this example, but may be useful in other cases.
        java.util.Vector rows = tranMsg.getRows();
        for (int i=0; i < rows.size(); i++) {
            Row row = (Row) rows.elementAt(i);
            if (row.getSrcName().equals("QUIKCLM")) {
```

Chapter 3. Trustworthy Insurance Customer Information Facility (CIF) and Event Alert System scenario  189
private void checkclaims(int i, Row row) throws Exception {

    // This code is used to calculate the number of days elapsed between the policy inception,
    // and the first claim
    // This is an indicator of potential fraud, and will be used to generate an email to the
    // fraud investigation department
    long DAYCOUNT = 0;
    String CUSTKEY = null;
    String POLICYNM = null;
    String POLSTDTE = null;
    String CLAIMNM = null;
    String CLAIMDTE = null;
    java.math.BigDecimal CLAIMAMT = null;
    String CLAIMDET = null;
    String XCUSTKEY = null;
    String XPOLICYNM = null;
    String XPOLSTDTE = null;
    String XCLAIMNM = null;
    String XCLAIMDTE = null;
    java.math.BigDecimal XCLAIMAMT = null;
    String XCLAIMDET = null;

    // The date fields need to be read in from CHAR(8) format, and converted to java date
    // format
    // The following fields are used to stage the incoming CHAR(8) data, before calculating
    // POLSTDTE, CLAIMDET, and XCLAIMDET
    String IN_POLSTDTE = null;
    String IN_XPOLSTDTE = null;
    String IN_CLAIMDTE = null;
    String IN_XCLAIMDTE = null;

    String rowOp = format_ROWOP(row.getRowOperation());
    java.util.Vector cols = row.getColumns();
    for (int j=0; j < cols.size(); j++) {
        Column col = (Column) cols.elementAt(j);
        String colName = col.getName();
        if (colName.equals("CUSTKEY")){
            CUSTKEY = (String) col.getValue();
        } else if (colName.equals("XCUSTKEY")){
            XCUSTKEY = (String) col.getValue();
        } else if (colName.equals("POLICYNM")){
            POLICYNM = (String) col.getValue();
        } else if (colName.equals("XPOLICYNM")){
            XPOLICYNM = (String) col.getValue();
        } else if (colName.equals("POLSTDTE")){
            IN_POLSTDTE = (String) col.getValue();
        } else if (colName.equals("XPOLSTDTE")){
            IN_XPOLSTDTE = (String) col.getValue();
        } else if (colName.equals("CLAIMAMT")){
            CLAIMAMT = java.math.BigDecimal.valueOf((BigDecimal) col.getValue());
        } else if (colName.equals("CLAIMDTE")){
            CLAIMDTE = (String) col.getValue();
        } else if (colName.equals("XCLAIMAMT")){
            XCLAIMAMT = java.math.BigDecimal.valueOf((BigDecimal) col.getValue());
        } else if (colName.equals("XCLAIMDTE")){
            XCLAIMDTE = (String) col.getValue();
        }
    }
    checkclaims(i, row);
}
return tranMsg.isLast();
}
Chapter 3. Trustworthy Insurance Customer Information Facility (CIF) and Event Alert System scenario

```java
POLSTDTE = IN_POLSTDTE.substring(4,8) + "-" + IN_POLSTDTE.substring(0,2) + "-" + IN_POLSTDTE.substring(2,4);

// For development and debug purposes, the values of xml fields will be printed to standard output
System.out.println("CUSTKEY = " + CUSTKEY);
System.out.println("POLICYNM = " + POLICYNM);
System.out.println("POLSTDTE = " + POLSTDTE);
System.out.println("CLAIMNM = " + CLAIMNM);
System.out.println("CLAIMDTE = " + CLAIMDTE);
System.out.println("CLAIMAMT = " + CLAIMAMT);
System.out.println("CLAIMDET = " + CLAIMDET);
System.out.println("XCUSTKEY = " + XCUSTKEY);
System.out.println("XPOLICYNM = " + XPOLICYNM);
System.out.println("XPOLSTDTE = " + XPOLSTDTE);
System.out.println("XCLAIMNM = " + XCLAIMNM);
System.out.println("XCLAIMDTE = " + XCLAIMDTE);
System.out.println("XCLAIMAMT = " + XCLAIMAMT);
System.out.println("XCLAIMDET = " + XCLAIMDET);

// Work out the java date format for the Policy Start Date.
int mm1 = Integer.parseInt(IN_POLSTDTE.substring(0,2));
System.out.println("month " + mm1);
```
```java
int dd1 = Integer.parseInt(IN_POLSTDTE.substring(2,4));
System.out.println("day " + dd1);

int yyyy1 = Integer.parseInt(IN_POLSTDTE.substring(4,8));
System.out.println("year " + yyyy1);

Date poldate = new GregorianCalendar(yyyy1, mm1, dd1, 00, 00).getTime();

// Work out the java date format for the Claim Date.

int mm2 = Integer.parseInt(IN_CLAIMDTE.substring(0,2));
System.out.println("month " + mm2);

int dd2 = Integer.parseInt(IN_CLAIMDTE.substring(2,4));
System.out.println("day " + dd2);

int yyyy2 = Integer.parseInt(IN_CLAIMDTE.substring(4,8));
System.out.println("year " + yyyy2);

Date claimdate = new GregorianCalendar(yyyy2, mm2, dd2, 00, 00).getTime();

// Work out the elapsed time difference between poldate and claimdate, first in milliseconds, then in days

long diff1 = claimdate.getTime() - poldate.getTime();
DAYCOUNT = diff1 / 86400000;

// Now Generate email to fraud department

System.out.println("CUSTOMER " + CUSTKEY + " WITH POLICY " + POLICYNM + " HAS MADE A QUICK CLAIM :");
System.out.println("CLAIM NUMBER " + CLAIMNM);
System.out.println("The days from policy start to policy claim is = " + DAYCOUNT);

// JAVA EMAIL CALL IS OUTSIDE SCOPE OF PROGRAM

System.out.println("End of loop i = " + i);
```

```java
private byte[] format_CMTSEQ(String cmtLSN){
    String s1 = cmtLSN.replaceAll(":" , "");
    return toBinary(s1);
}

private byte[] format_INTSEQ(int intSEQ){
    String s1 = Integer.toString(intSEQ).trim();
    int ln = 20 - s1.length();
    for (int i = 1; i <= ln ; i++) {
        s1 = "0" + s1;
    }
    return toBinary(s1);
}

private byte[] toBinary(String inStr) {
    String sbStr;
    byte[] bytes = new byte[10];
    for(int i=0 ; i < 10; i++){
```
sbStr = inStr.substring(2*i,2*i+2);
int a = Integer.parseInt(sbStr,16);
if ( a <= 127 ){  
    bytes[i] = Byte.parseByte(Integer.toString(a));}else{
    bytes[i] = Byte.parseByte(Integer.toString(a - 256 ));
}
return bytes;
}
private Timestamp format_CMTTM(String cmtTM) {
    return Timestamp.valueOf(cmtTM.replace('T' ,' '));
}
private String format_ROWOP(int rowOp) {
    switch (rowOp)  {
        case Row.InsertOperation : return  "I";
        case Row.DeleteOperation : return  "D";
        case Row.UpdateOperation : return  "U";
        default :                  return  "?";
    }
}

3.3.9  STEP 8: Start WIICEP for IMS, WIIEP for DB2 UDB for z/OS and IMS

"STEP 3i: Start WIICEP for IMS" on page 29 describes the process for starting WIICEP for IMS.

WIIEP for DB2 UDB for z/OS is started using JCL shown in Figure 3-36 on page 193 — either as a batch job or as a started task.

Example 3-36   JCL to start WIIEP for DB2 UDB for z/OS

//QCAP      EXEC PGM=ASNQCAP,REGION=64M,
//      PARM='STORAGE(FF,FF,FF)/CAPTURE_SERVER=DB8G1 CAPTURE_SCHEMA=ASN
//      STARTMODE=COLD'
//STEPLIB  DD DSN=ASN.V8R2M0.SASNLOAD,DISP=SHR
//         DD DSN=DB8G8.SDSNLOAD,DISP=SHR
//MSGS     DD PATH='/u/nalur/db2asn.cat'
//CEEDUMP  DD  DUMMY
//SYSPRINT DD  SYSOUT=*  
//CAPSPILL DD DSN=CAC.CAPSPL,DISP=(NEW,DELETE,DELETE),
//      UNIT=VIO,SPACE=(CYL,(50,100)),
//      DCB=(RECFM=VB,BLKSIZE=6404)
//SYSUDUMP DD DUMMY
//SYSTERM  DD DUMMY
//*

Attention: The Event Notification Program is operational when the MQ Listener class mqlimsk2.class () is started.

Attention: After WIIEP for IMS starts successfully, IMS can be started to commence capture of changes occurring in the auto insurance IMS database. The capture of changes from the home insurance DB2 for z/OS database commences as soon as WIIEP for DB2 UDB for z/OS (Q Capture program) is started.
Recovery mode scenarios

In this chapter we describe our recommended approach for resolving the occurrence of recovery mode in IMS active CCAs.

The topics covered are:
- Simple recovery mode scenario
- Recovery mode resolution recommended approach
- Complex recovery mode scenario
4.1 Introduction

As discussed in Section 1.6, “WIICEP for IMS recovery mode” on page 44, the incidence of recovery mode in an event publishing environment requires very special handling to ensure that no ongoing changes to the database are lost. In this section, we describe the following situations:

► A simple event publishing environment involving two CCAs (an online IMS region and a batch region) that update the same database serially. The documented approach for detecting recovery mode state and restoring active mode state is described here.

► Recovery mode resolution recommended approach describes our recommendation for detecting and restoring active mode state for related CCAs.

► Complex recovery mode scenario that applies our recommended approach, and leveraging the REXX tool we developed to identify all the logs required as input to the recovery CCA batch job.

4.2 Simple recovery mode scenario

Figure 4-1 shows a timeline of operations in an IMS event publishing environment involving two IMS CCAs that update the same database in serial fashion — the database is updated initially by the online region (IMS810K), then deallocated from the online region, updated by the batch job (CACBACHG), reallocated to the online region after the batch job completes to be again updated by the IMS online region.

Attention: A Microsoft Power Point animated slide show presentation of this scenario can be downloaded from:

ftp://www.redbooks.ibm.com/redbooks/SG247132
The scenario shown in Figure 4-1 assumes the following facts about the IMS event publishing environment:

- The auto insurance IMS database AUTODB described in Chapter 3, “Trustworthy Insurance Customer Information Facility (CIF) and Event Alert System scenario” on page 85 is the data source, and that there are only two related CCAs (IMS810K online and CACBACHG batch) associated with this event publishing environment.
- DBRC controls both the IMS online and batch job running in the event publishing environment.
- Recovery data sets are defined for each CCA.
- No GDG is used for the log used by the batch job CACBACHG.

The processing at each numbered point in the timeline shown in Figure 4-1 is described as follows:

1. Time ‘t1’ to ‘t2’

   During this interval normal operations are in progress. The correlation service and publication have been started successfully as shown in Example 4-1, followed by the start of the IMS control region. A BMP application updates the IMS AUTODB database as shown in Example 4-2 — resulting in a message in the SDSF log that changes are being received by the correlation service as shown in Example 4-3. These changes are captured by the IMS810K active CCA, published as per the publication definition shown in Example 4-4, and received at the IMSK_RECVQ1 target queue as shown in Example 4-5.
Example 4-1  Correlation service and publication service startup messages

12.43.36 STC26800 ---- TUESDAY, 06 SEP 2005 ----
12.43.36 STC26800 IEF695I START CACCS2 WITH JOBNAME CACCS2 IS ASSIGNED TO U
12.43.36 STC26800 $HASP373 CACCS2 STARTED
12.43.37 STC26800 CAC00100I CONTROLLER: LOGGING STARTED
12.43.37 STC26800 CAC00105I LOG V8.2 03102005: STARTED
12.43.37 STC26800 CACG000I CORRELATION SERVICE IS WARM STARTING
12.43.37 STC26800 CACG002I CORRELATION SERVICE RUNNING IN STORAGE KEY 08
12.43.37 STC26800 CACG005I CHANGE CAPTURE SET FOR IMS TABLE IMSKAUTO.ADDRESS
12.43.37 STC26800 CACG005I CHANGE CAPTURE SET FOR IMS TABLE IMSKAUTO.CLAIMS
12.43.37 STC26800 CACG005I CHANGE CAPTURE SET FOR IMS TABLE IMSKAUTO.CUSTOMER
12.43.37 STC26800 CACG005I CHANGE CAPTURE SET FOR IMS TABLE IMSKAUTO.POLICY
12.43.37 STC26800 CACG005I CHANGE CAPTURE SET FOR IMS TABLE IMSKAUTO.PREMIUMS
12.43.37 STC26800 CACG005I CHANGE CAPTURE SET FOR IMS TABLE IMSKAUTO.QUIKCLM
12.43.38 STC26800 CAC00102I CONTROLLER: STARTED CACECA2
12.43.38 STC26800 CACG100I CORRELATION SERVICE (V8.2 03102005) STARTED
12.43.38 STC26800 CAC00105I CACPUB SERVICE V8.2 03102005: STARTED
12.43.38 STC26800 CAC00102I CONTROLLER: STARTED CACPUB
12.43.38 STC26800 CAC00103I DATA SERVER: V8.2 03102005 READY
12.43.39 STC26800 CACG101I DISTRIBUTION SERVICE CONNECTED

Example 4-2  BMP DDLT0 job to update the AUTODB database

//CACBMPCH JOB (999,POK),'CHRIS',CLASS=A,MSGCLASS=H,MSGLEVEL=(1,1),
// NOTIFY=&SYSUID,REGION=128M
//*
//*
//   JCLLIB ORDER=(IMS810K.PROCLIB)
/*JOBPARM L=9999,SYSAFF=*  
//*
//AUTODBJ EXEC PROC=IMSBATCH,MBR=DFSDDLTO,PSB=AUTOPSB1,
//    IMSID=IMSK,OUT='*',AGN=IVP
//*
//DFSSTAT DD SYSOUT=*  
//AUTODB DD DSN=IMS810K.AUTODB,DISP=SHR
//AUTODBO DD DSN=IMS810K.AUTODBO,DISP=SHR
//CACRCV DD DISP=SHR,DSN=CAC.V8R2MD.CCARECOV
//*
//PRINTDD DD SYSOUT=*  
//*
//SYSIN DD *,DCB=BLKSIZE=80
S 1 1 2 1 1 2AUTODB
L   GHU   CUSTOMER(CUSTKEY = H000000010)
L   REPL
L   DATA  H000000010CHRISTIAN     HAUS     0X
L   1011965
L   GHU   CUSTOMER(CUSTKEY = T000000020)
L   REPL
L   DATA  T000000020MICHAEL         THOMPSONS   1X
L   0101973
//*

Example 4-3  Receiving changes from the IMS810K CCA

....
12.48.50 STC26800 CACG109I RECEIVING CHANGES FROM ACTIVE AGENT 'IMS.IMS810K'
....
Example 4-4  Publication definition in the configuration file

PUB ALIAS=PUBCUSTN,
    TOPIC=DB2INST1/TESTOPIC,
    MSGTYPE=TRANS,
    QUEUE=MQI/MQ8G/IMSK_SENDQ1,
    TABLE=IMSKAUTO.CUSTOMER,
    BEFORE_VALUES=YES

Example 4-5  Published XML message in the IMSK_RECVQ1 target queue

@@@
<psc>
    <Command>Publish</Command>
</psc>
<msd>None</msd>
<xml version="1.0" encoding="UTF-8">
    <msg xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="mqcap.xsd" version="1.0.0"
         dbName="IMS"
         seqNum="IMSIMSK0000004900000000_1"
         cmitLSN="IMSIMSK0000004900000000">
        <trans isLast="1" segmentNum="1"
               cmitLSN="IMSIMSK0000004900000000"
               cmitTime="2005-09-06:12:48:51.000000">
            <updateRow subName="PUBCUSTN"
                       srcOwner="IMSKAUTO"
                       srcName="CUSTOMER">
                <col name="CUSTKEY">
                    <char>
                        <beforeVal>H000000010</beforeVal>
                        <afterVal>H000000010</afterVal>
                    </char>
                </col>
                <col name="FIRSTNME">
                    <char>
                        <beforeVal>CHRISTINE</beforeVal>
                        <afterVal>CHRISTIAN</afterVal>
                    </char>
                </col>
                <col name="LASTNAME">
                    <char>
                        <beforeVal>HAAS</beforeVal>
                        <afterVal>HAUS</afterVal>
                    </char>
                </col>
            </updateRow>
            <updateRow subName="PUBCUSTN"
                       srcOwner="IMSKAUTO"
                       srcName="CUSTOMER">
                <col name="CUSTKEY">
                    <char>
                        <beforeVal>T000000020</beforeVal>
                        <afterVal>T000000020</afterVal>
                    </char>
                </col>
                <col name="FIRSTNME">
                    <char>
                        <beforeVal>MICHA</beforeVal>
                        <afterVal>MICHAEL</afterVal>
                    </char>
                </col>
                <col name="LASTNAME">
                    <char>
                        <beforeVal>THOMPSONS</beforeVal>
                        <afterVal>THOMPSONS</afterVal>
                    </char>
                </col>
        </trans>
    </msg>
</xml>
2. At time ‘t2’

We force the online region to go into recovery mode by stopping the correlation service using the following command from SDSF which generates the messages shown in Example 4-6:

/p caccs2

**Example 4-6  Correlation service stop messages**

```
CAC002001 STOP,ALL
CAC002041 STOP ALL COMMAND ACCEPTED
CAC01061 CACPUB SERVICE: TERMINATING
CACG150I CORRELATION SERVICE ACTIVITY REPORT

*************** Transactions ***************
Agent          Processed Sent to Rules Confirmed Pending State
---------------- ---------  -------------  ---------  -------   --------
IMS_IMS810K       0000001      0000001      0000001   0000000   Active

CACG151I END OF REPORT, AGENT TOTAL=1
CACG152I PENDINGQ(0) MSGQ(0) UNCONFIRMED(0)
CACG102W DISTRIBUTION SERVICE DISCONNECTED
CACG310I IMS SUMMARY INFORMATION AGENTS: 1
CACG311IBUFFERS: 2 RECORDS: 3 UORS: 1 COMMITS: 1 ROLLBACKS: 0
CACG312I EVENTS: 1 CASC: 0 DLET: 0 DLLP: 0 ISRT: 0 REPL 2
CACG314W SUMMARY INFORMATION FOR IMS AGENT IMSK IMS810K STC26655 8
CACG315WBUFFERS: 2 RECORDS: 3 UORS: 1 COMMITS: 1 ROLLBACKS: 0
CACG313I NO IN-FLIGHT IMS UORS
CACG801W CORRELATION SERVICE TERMINATING
CAC001061 LOG: TERMINATING
```

**Note:** Recovery mode is set when the active CCA attempts to send an XM data gram to the correlation service and gets a bad return code — it writes a WTOR message as shown in Example 4-7.

**Example 4-7  Correlation service unavailable message**

```
CACH002A EVENT PUBLISHER SERVICE ’(noname)’ NOT FOUND BY
CHANGE-CAPTURE AGENT ’IMS_IMS810K ’, REPLY ’R’ OR ’A’ RECOVER
```

The IMS active CCA automatically disables itself until an operator response is received and the correlation service is started. During this period, database activity continues normally and changes passed to the CCA are ignored.

- The operator should respond ‘R’ to this message to ensure that database changes are not lost.
- The recovery mode indicator is reset when you reply ‘A’ to the message, and on the next time the CCA is called it will attempt to send an XM data gram to the correlation service. In this particular instance, the CCA will detect that the correlation service is not active and go back into recovery mode. The contents of the recovery dataset are not updated with the restart point unless there is already a record in the recovery dataset that contains prior restart information.
When the response is ‘R’ to the CACH002A message, the IMS logger exit is unable to write to the CSA (because it was removed when the correlation service shutdown and the CSARLSE parameter in the SIE was set to 3 seconds), and therefore writes the restart information (as shown in Example 4-8) at time ‘t3’, into its recovery data set (CAC.RECOVDS.IMS810K) when it issues the CACH002A WTOR message.

**Attention:** The IMS active CCA IMS810K is now in recovery mode, and no further changes that may occur in the IMS online region are published.

**Example 4-8  Restart information in the IMS810K recovery data set in hexmode**

```
************ Top of Data ************
IMS_IMS810K .......cki................... "cki--........ "cki--....... 
CDE6DDEFFC44420001122770000000000000000000000002F890C766200000002F890C7662000000 
942D94210200000059631832800000000000000000007AD27984060000007AD2798406000000 
************ Bottom of Data ************
```
Figure 4-2 shows the recovery data set metadata.

<table>
<thead>
<tr>
<th>Name</th>
<th>Starting Offset</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent Type</td>
<td>1</td>
<td>4</td>
<td>Identifies the type of agent that is being recovered. See this to IMS.</td>
</tr>
<tr>
<td>Agent Name</td>
<td>5</td>
<td>12</td>
<td>Name of the IMS job or started task that you want to recovery. The name must be padded with trailing blanks.</td>
</tr>
<tr>
<td>DB2 Time Stamp</td>
<td>17</td>
<td>10</td>
<td>Approximate recent time in DB2 timestamp format.</td>
</tr>
<tr>
<td>IMS UOR Identifier</td>
<td>27</td>
<td>16</td>
<td>The IMS unit-of-recovery identifier for a committed UOR that was being processed when the agent went into recovery. If the UOR was not committed at the time the agent went into recovery, this field contains null values (binary zeroes).</td>
</tr>
<tr>
<td>Current UOR Starting Log Record Sequence Number</td>
<td>43</td>
<td>8</td>
<td>Binary, double-word IMS log sequence number for the first type 99 data capture log record associated with the UOR identified by the IMS UOR Identifier field, or, if the IMS UOR Identifier is null, the IMS log sequence number of the first IMS log record in the file when the IMS subsystem was started.</td>
</tr>
<tr>
<td>Current UOR Starting System Clock</td>
<td>51</td>
<td>8</td>
<td>Binary, system clock value from the IMS log record suffix for the first type 99 data capture log record associated with the UOR identified by the IMS UOR Identifier field, or, if the IMS UOR Identifier is null, the system clock value from the log record suffix of the first IMS log record in the file when the IMS subsystem was started.</td>
</tr>
<tr>
<td>Oldest UOR Log Record Sequence Number</td>
<td>59</td>
<td>8</td>
<td>Binary, double-word IMS log sequence number for the first type 99 data capture log record for the UOR that is the &quot;oldest&quot; UOR when the UOR identified by the IMS UOR Identifier field started. If the UOR Identifier is null, the value that must be supplied is the same value that is contained in the Current UOR Starting Log Record Sequence Number field.</td>
</tr>
<tr>
<td>Oldest UOR System Clock</td>
<td>67</td>
<td>8</td>
<td>Binary, system clock value from the IMS log record suffix for the first type 99 data capture log record for the &quot;oldest&quot; UOR when the UOR identified by the IMS UOR Identifier field started. If the UOR Identifier is null, the value that must be supplied is the same value that is contained in the Current UOR Starting System Clock field.</td>
</tr>
</tbody>
</table>

Attention: When a CCA starts, it looks into its recovery data set, and if there is an entry containing restart information, the CCA will not start in active mode. Therefore, this entry in the recovery data set must be deleted before a CCA can become active — the recovery CCA batch job deletes this entry in the recovery data set when it places a CCA in active mode.

3. At time ‘t4a’

Before the batch job can be run against the AUTODB database, it must be deallocated from the IMS online region using the following command:

```
/.,/DBR DB AUTOBD NOFEOV
```
4. At time ‘t4b’ and ‘t4c’

Start the CACBACHG batch. When the CACBACHG CCA can not connect to the correlation service because of its unavailability, it is unable to write to the CSA and therefore writes a WTOR message (Example 4-9), and restart information (as shown in Example 4-10) into its recovery data set (CAC.RECOVDS.CACBACHG). It then goes into recovery mode, but continues processing updates until termination at time ‘t4c’.

**Note:** No changes are published since the batch CACBACHG CCA is in recovery mode.

Example 4-9  Correlation service unavailable message

```
13.18.18 JOB26846 0558 CACH002A EVENT PUBLISHER SERVICE '{(noname)'} NOT FOUND BY CHANGE-CAPTURE AGENT 'IMS_CACBACHG ', REPLY 'R' OR 'A' RECOVER
```

Example 4-10  Restart information in the CACBACHG recovery data set in hexmode

```
********************************** Top of Data **********************************
IMS_CACBACHG .......qm.........................¨kú;^.)À........¨kú;^.)À......
CDE6CCCCCCCCC4444200011119930000000000000000000000000089D583560000000000089D583560000000
942D31321387000059638784200000000000000000000000000001D2EE0CD400000001D2EE0CD40000000
********************************** Bottom of Data **********************************
```

**Attention:** The restart log sequence number (LSN) is 1 for the batch job as shown in Example 4-10 because the log data set is new and empty.

5. At time ‘t4d’

After the batch job has terminated, the AUTODB database needs to be reallocated to the IMS online region using the command:

```
/nnn,/START DB AUTODB
```

At this point:

- None of the changes to the AUTODB database since time ‘t2’ have been published.
- Even though both the IMS810K and CACBACHG CCAs are in recovery mode, the correlation service is unaware of the recovery mode state of these two CCAs.

**Attention:** It is therefore essential that the correlation service be made aware of the recovery mode state of these CCAs, before recovery CCA can be run to restore the two CCAs to active mode.

Recovery CCA can only be run on CCAs that the correlation service thinks are in recovery mode.

This is achieved by:

- Checking the status of the CCAs by running recovery CCA in ‘C’ mode (check agent).
- Setting the status of the CCAs to recovery mode by running recovery CCA in ‘S’ mode (set agent in recovery mode).

**Important:** The correlation service must be running to run the recovery CCA, and must therefore be started.
6. At time ‘t5’

The correlation service is started using the following command, which causes CCAs not in
recovery mode to commence event publishing:

```
/S CACCS2
```

The messages generated are shown in Example 4-11.

**Example 4-11  Correlation service startup messages**

<table>
<thead>
<tr>
<th>Time</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>13.23.53</td>
<td>STC26849  IEF6951 START CACCS2  WITH JOBNAME CACCS2  IS ASSIGNED TO USER STC, GROUP SYS1</td>
</tr>
<tr>
<td>13.23.53</td>
<td>STC26849  $HASP373 CACCS2 STARTED</td>
</tr>
<tr>
<td>13.23.53</td>
<td>STC26849  CAC00105I LOG V8.2 03102005: STARTED</td>
</tr>
<tr>
<td>13.23.53</td>
<td>STC26849  CAC00100I CONTROLLER: LOGGING STARTED</td>
</tr>
<tr>
<td>13.23.54</td>
<td>STC26849  CACG002I CORRELATION SERVICE RUNNING IN STORAGE KEY 08</td>
</tr>
<tr>
<td>13.23.54</td>
<td>STC26849  CACG005I CHANGE CAPTURE SET FOR IMS TABLE IMSKAUTO.ADDRESS</td>
</tr>
<tr>
<td>13.23.54</td>
<td>STC26849  CACG005I CHANGE CAPTURE SET FOR IMS TABLE IMSKAUTO.CLAIMS</td>
</tr>
<tr>
<td>13.23.54</td>
<td>STC26849  CACG005I CHANGE CAPTURE SET FOR IMS TABLE IMSKAUTO.CUSTOMER</td>
</tr>
<tr>
<td>13.23.54</td>
<td>STC26849  CACG005I CHANGE CAPTURE SET FOR IMS TABLE IMSKAUTO.POLICY</td>
</tr>
<tr>
<td>13.23.54</td>
<td>STC26849  CACG005I CHANGE CAPTURE SET FOR IMS TABLE IMSKAUTO.PREMIUMS</td>
</tr>
<tr>
<td>13.23.54</td>
<td>STC26849  CACG005I CHANGE CAPTURE SET FOR IMS TABLE IMSKAUTO.QUIKCLM</td>
</tr>
<tr>
<td>13.23.54</td>
<td>STC26849  CACG100I CORRELATION SERVICE (V8.2 03102005) STARTED</td>
</tr>
<tr>
<td>13.23.54</td>
<td>STC26849  CAC00102I CONTROLLER: STARTED CACECA2</td>
</tr>
<tr>
<td>13.23.54</td>
<td>STC26849  CAC00105I CACPUB SERVICE V8.2 03102005: STARTED</td>
</tr>
<tr>
<td>13.23.55</td>
<td>STC26849  CAC00102I CONTROLLER: STARTED CACPUB</td>
</tr>
<tr>
<td>13.23.55</td>
<td>STC26849  CAC00103I DATA SERVER: V8.2 03102005 READY</td>
</tr>
</tbody>
</table>

7. At time ‘t6’

Run the recovery CCA in ‘C’ mode identifying both the CCAs as shown in Example 4-12
— the results of this job are shown in Example 4-13.

**Example 4-12  Recovery CCA ‘C’ mode control cards**

```
C IMS81OK 4 CAC.RECOVDS.IMS81OK
C CACBACHG 4 CAC.RECOVDS.CACBACHG
```

**Example 4-13  Recovery CCA ‘C’ mode output**

<table>
<thead>
<tr>
<th>Time</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>13.26.02</td>
<td>JOB26855  +CACH061I RECOVERY MODE: CHECK AGENT STATUS</td>
</tr>
<tr>
<td>13.26.02</td>
<td>JOB26855  +CACH048I RECOVERY DATASET CAC.RECOVDS.IMS81OK</td>
</tr>
<tr>
<td>13.26.02</td>
<td>JOB26855  + OPENED</td>
</tr>
<tr>
<td>13.26.02</td>
<td>JOB26855  +CACH049I AGENT NAME IS 'IMS81OK '</td>
</tr>
<tr>
<td>13.26.02</td>
<td>JOB26855  +CACH039I DB2 RESTART TIME 20050906 13112823</td>
</tr>
<tr>
<td>13.26.02</td>
<td>JOB26855  +CACH040I IMS RESTART TIME 05.249 13:11:28.2</td>
</tr>
<tr>
<td>13.26.02</td>
<td>JOB26855  +CACH037I RESTART SYSTEM CLOCK BD92D7C978646026</td>
</tr>
<tr>
<td>13.26.02</td>
<td>JOB26855  +CACH038I RESTART LOG SEQ. # 00000000-000027FA</td>
</tr>
<tr>
<td>13.26.02</td>
<td>JOB26855  +CACH030I AGENT 'IMS81OK ' IS NOT IN RECOVERY MODE</td>
</tr>
<tr>
<td>13.26.02</td>
<td>JOB26855  +CACH067I LOG TRACKING FILE DOES NOT EXIST</td>
</tr>
<tr>
<td>13.26.02</td>
<td>JOB26855  +CACH048I RECOVERY DATASET CAC.RECOVDS.CACBACHG</td>
</tr>
<tr>
<td>13.26.02</td>
<td>JOB26855  + OPENED</td>
</tr>
<tr>
<td>13.26.02</td>
<td>JOB26855  +CACH049I AGENT NAME IS 'CACBACHG'</td>
</tr>
<tr>
<td>13.26.02</td>
<td>JOB26855  +CACH039I DB2 RESTART TIME 20050906 13181798</td>
</tr>
<tr>
<td>13.26.02</td>
<td>JOB26855  +CACH040I IMS RESTART TIME 05.249 13:18:17.9</td>
</tr>
<tr>
<td>13.26.02</td>
<td>JOB26855  +CACH037I RESTART SYSTEM CLOCK BD92D7C978646026</td>
</tr>
<tr>
<td>13.26.02</td>
<td>JOB26855  +CACH038I RESTART LOG SEQ. # 00000000-00000001</td>
</tr>
<tr>
<td>13.26.02</td>
<td>JOB26855  +CACH030I AGENT 'CACBACHG' IS NOT IN RECOVERY MODE</td>
</tr>
<tr>
<td>13.26.02</td>
<td>JOB26855  +CACH067I LOG TRACKING FILE DOES NOT EXIST</td>
</tr>
<tr>
<td>13.26.02</td>
<td>JOB26855  +CACH071I 2 IMS AGENTS ARE IN RECOVERY MODE</td>
</tr>
<tr>
<td>13.26.02</td>
<td>JOB26855  +CACH062I RECOVERY PROCESSING COMPLETED SUCCESSFULLY</td>
</tr>
</tbody>
</table>
The highlighted data in Example 4-13 shows the recovery CCA opening the CAC.RECOVDS.IMS810K recovery data set (message CACH048I) and finding restart information (compare it with Example 4-8 on page 201) in it, but not finding restart information in the correlation service restart data set (message CACH030I). The same applies to the CAC.RECOVDS.CACBACHG recovery data set. The check job finds a restart point in the recovery data set but not in the correlation service restart data set.

8. At time 't7'

To ensure that the correlation service is made aware of the recovery mode state of both CCAs, run the recovery CCA in 'S' mode identifying both the CCAs as shown in Example 4-14 — the results of this job are shown in Example 4-15.

**Example 4-14  Recovery CCA 'S' mode control cards**

```
S IMS810K 4 CAC.RECOVDS.IMS810K
S CACBACHG 4 CAC.RECOVDS.CACBACHG
```

**Example 4-15  Recovery CCA 'S' mode output**

```
13.30.54 JOB26862 +CACH061I RECOVERY MODE: SET AGENT IN RECOVERY
13.30.54 JOB26862 +CACH048I RECOVERY DATASET CAC.RECOVDS.IMS810K
13.30.54 JOB26862 + OPENED
13.30.54 JOB26862 +CACH052I AGENT 'IMS810K ' IS NOW IN RECOVERY MODE
13.30.54 JOB26862 +CACH039I DB2 RESTART TIME    20050906 13112823
13.30.54 JOB26862 +CACH040I IMS RESTART TIME    05.249 13:11:28.2
13.30.54 JOB26862 +CACH037I RESTART SYSTEM CLOCK BD92D7C978646026
13.30.54 JOB26862 +CACH038I RESTART LOG SEQ. #  00000000-000027FA
13.30.54 JOB26862 +CACH067I LOG TRACKING FILE DOES NOT EXIST
13.30.54 JOB26862 +CACH048I RECOVERY DATASET CAC.RECOVDS.CACBACHG
13.30.54 JOB26862 + OPENED
13.30.54 JOB26862 +CACH052I AGENT 'CACBACHG' IS NOW IN RECOVERY MODE
13.30.54 JOB26862 +CACH039I DB2 RESTART TIME    20050906 12485087
13.30.54 JOB26862 +CACH040I IMS RESTART TIME    05.249 12:48:50.8
13.30.54 JOB26862 +CACH037I RESTART SYSTEM CLOCK BD92DE5EB03C5D64
13.30.54 JOB26862 +CACH038I RESTART LOG SEQ. #  00000000-00000001
13.30.54 JOB26862 +CACH067I LOG TRACKING FILE DOES NOT EXIST
13.30.54 JOB26862 +CACH062I RECOVERY PROCESSING COMPLETED SUCCESSFULLY
```

The recovery mode state can be verified by running the recovery CCA batch job in 'C' mode — the results of which are shown in Example 4-16.

**Example 4-16  Verify recovery mode state by running recovery CCA 'C' mode**

```
+CACH061I RECOVERY MODE: CHECK AGENT STATUS
+CACH048I RECOVERY DATASET CAC.RECOVDS.IMS810K
+ OPENED
+CACH049I AGENT NAME IS 'IMS810K '
+CACH039I DB2 RESTART TIME    20050906 13112823
+CACH040I IMS RESTART TIME    05.249 13:11:28.2
+CACH037I RESTART SYSTEM CLOCK BD92D7C978646026
+CACH038I RESTART LOG SEQ. #  00000000-000027FA
+CACH051I CORRELATION SERVICE REPORTS AGENT IN RECOVERY MODE
+CACH049I AGENT NAME IS 'IMS810K '
+CACH039I DB2 RESTART TIME    20050906 12485087
+CACH040I IMS RESTART TIME    05.249 12:48:50.8
+CACH037I RESTART SYSTEM CLOCK BD92DE5EB03C5D64
+CACH038I RESTART LOG SEQ. #  00000000-00000001
+CACH067I LOG TRACKING FILE DOES NOT EXIST
+CACH048I RECOVERY DATASET CAC.RECOVDS.CACBACHG
+ OPENED
```
Since the correlation service recognizes the recovery mode state of IMS810K and CACBACHG CCAs, you can begin the process of recovering the two CCAs.

The next steps involved are as follows:

a. Identify the logs required for the two CCAs (time ‘t8’).

b. Run recovery CCA in ‘L’ mode for IMS810K and CACBACHG with the required logs (time ‘t9’).

c. Run recovery CCA in ‘A’ mode which sets the IMS810K and CACBACHG in to active mode (time ‘t10’).

9. At time ‘t8’

You need to identify the logs required for the two CCAs as follows:

– The log data set required for the batch CACBACHG CCA is NALUR.LOG.

– The log data sets required for the online region IMS810K CCA is determined by executing the LIST.LOG ALL command. Partial output for PRIOLD (primary online logs) that contain records after “IMS RESTART TIME 05.249 12:48:50.8” (from Example 4-16) is shown in Example 4-17 — this time corresponds to the earliest restart time as noted by the recovery CCA running in ‘C’ mode.

A STOP value of ‘0’ indicates that this is the current log and has not finished yet. The only log file required to recover IMS810K CCA is IMS810K.OLP03.

Example 4-17 LIST.LOG ALL

0 LIST.LOG ALL
.............
PRIOLD
SSID=IMSK # DD ENTRIES=7
EARLIEST CHECKPOINT = 05.238 17:22:17.4

DDNAME=DFSOLP03 DSN=IMS810K.OLP03
START = 05.249 08:39:02.9 FIRST DS LSN= 0000000000002690
STOP = 00.000 00:00:00.0 LAST DS LSN= 0000000000000000
LOCK SEQUENCE# = 000000000000
STATUS=ACTIVE FEOV=NO AVAIL
PRILOG TIME=05.249 08:39:02.9
VERSION=8.1

DDNAME=DFSOLP04 DSN=IMS810K.OLP04
START = 05.245 14:45:55.6 FIRST DS LSN= 0000000000001E76
STOP = 05.245 17:28:38.1 LAST DS LSN= 0000000000002158
LOCK SEQUENCE# = 000000000000
STATUS=ARC COMPLT FEOV=NO AVAIL
10. At time 't9'

Recovery of the IMS810K and CACBACHG CCAs can commence using the recovery CCA using 'L' mode (log file recovery) control cards as shown in Example 4-18. The output of this job is shown in Example 4-19. The SDSF log shows the messages generated by the recovery CCA 'L' mode execution.

The target queue IMSK_RECVQ1 contains messages corresponding to changes since moments after time 't2'.

Example 4-18  Recovery CCA 'L' mode control cards

L IMS810K 4 IMS810K.OLP03
L CACBACHG 4 NALUR.LOG
Example 4-19  Recovery CCA ‘L’ mode output

14.01.01 JOB26881 +CACH061I RECOVERY MODE: IMS LOG FILE RECOVERY
14.01.01 JOB26881 +CACH055I STARTING LOG FILE SEQUENCE CHECKING
14.01.01 JOB26881 +CACH058I LOG FILE SEQUENCE CHECKING COMPLETED
14.01.02 JOB26881 +CACH044I AGENT 'IMS810K' LOG OPENED: IMS810K.OLP03
14.01.02 JOB26881 +CACH044I AGENT 'CACBACHG' LOG OPENED: NALUR.LOG
14.01.02 JOB26881 +CACH045I AGENT 'IMS810K' LOG CLOSED: IMS810K.OLP03
14.01.02 JOB26881 +CACH045I AGENT 'CACBACHG' LOG CLOSED: NALUR.LOG
14.01.02 JOB26881 +CACH072I BUFFERS SENT 2 RECORDS 28 THROTTLES 2
14.01.02 JOB26881 +CACH062I RECOVERY PROCESSING COMPLETED SUCCESSFULLY

Example 4-20  SDSF log messages of the recovery CCA ‘L’ mode

CACGI09I RECEIVING CHANGES FROM RECOVERY AGENT 'IMS_IMS810K'
CACG305I SUMMARY INFORMATION FOR IMS AGENT RECOVERY IMS810K IMS810K 48
CACG306I BUFFERS: 1 RECORDS: 24 UORS: 7 COMMITS: 7 ROLLBACKS: 0
CACGI109I RECEIVING CHANGES FROM RECOVERY AGENT 'IMS_CACBACHG'
CACG305I SUMMARY INFORMATION FOR IMS AGENT RECOVERY CACBACHG CACBACHG 48
CACG306I BUFFERS: 1 RECORDS: 4 UORS: 1 COMMITS: 1 ROLLBACKS: 0

11. At time ‘t10’

Once all the logs have been recovered, ensure that no more changes occur on the AUTODB database (by making the AUTODB read only or shutting down the online region) otherwise changes will be lost, before running the recovery CCA in ‘A’ mode (place agent in active mode) using the control cards shown in Example 4-21. The output of this job is shown in Example 4-22.

Example 4-21  Recovery CCA ‘A’ mode control cards

A IMS810K 4 CAC.RECOVDS.IMS810K
A CACBACHG 4 CAC.RECOVDS.CACBACHG

Example 4-22  Recovery CCA ‘A’ mode output

+CACH061I RECOVERY MODE: ACTIVATE AGENT
+CACH054I PREPARING TO ACTIVATE AGENT 'IMS810K'
+CACH047E RECOVERY DATASET OPEN ERROR FOR CAC.RECOVDS.IMS810K + ERRNO: 19
+CACH054I PREPARING TO ACTIVATE AGENT 'CACBACHG'
+CACH031I AGENT 'CACBACHG' SWITCHING TO ACTIVE MODE
+CACH062I RECOVERY PROCESSING ENDED WITH WARNINGS

In our scenario, since we did not shut down the IMS online region, the correlation service can not access the recovery data set (CAC.RECOVDS.IMS810K) of the IMS810K CCA but still activates the IMS810K CCA. However, the correlation service deletes the entry in CACBACHG CCA recovery data set CAC.RECOVDS.CACBACHG and activates the CACBACHG CCA.

The recovery CCA may be executed in ‘C’ mode (check agent) to verify that both the CCAs are active as shown in Example 4-23.
Chapter 4. Recovery mode scenarios

4.3 Recovery mode resolution recommended approach

This section describes our recommended approach for resolving recovery mode occurrences in the WIICEP for IMS event publishing environment. It differs somewhat from the more general approach described in Section 4.2, “Simple recovery mode scenario” on page 196 in that it describes a more complex environment with multiple related CCAs, all of which may not be active when recovery mode is initiated for any one member of the related CCAs. Section 4.4, “Complex recovery mode scenario” on page 215 describes a complex scenario that applies the recommended approach.

Important: Our recommended approach is dependent on defining the “optional” recovery data set for each CCA.
Figure 4-3 shows the overall flow of our recommended approach.

Figure 4-3 is best explained using a scenario involving multiple CCAs. Assume that an organization has one IMS control region (IMSO) that updates a SAVINGDB IMS database along with five batch jobs Batch1 through Batch5 that also update the same database in serial fashion (that is, there is no overlap of batch job execution). Not all the batch jobs may be executed in a particular session as shown in Figure 4-4, where two timelines are presented. Six related CCAs exist for this event publishing environment.
We will use the environment described in Figure 4-4 to explain our recommended approach. This figure shows two potentially different timelines, A and B.

SAVINGDB IMS database event publishing

Six related CCAs that update SAVINGDB in serial fashion (no overlap)
- One IMS online region IMSO
- Five Batch jobs Batch1, Batch2, Batch3, Batch4, Batch5

Timeline A

Timeline B

1. WTO messages:
   When an active CCA goes into recovery mode, it writes WTO messages to the system log. An automated operator facility is strongly recommended for early detection and event publisher administrator (EPA) notification to prevent out-of-sequence published messages as discussed in “STEP 1: Identify status of all related IMS CCAs” on page 55.

2. Are there related CCAs?
   As soon as a recovery mode situation is detected for an active CCA, the EPA needs to immediately check whether this active CCA has any related CCAs, so that they may also be forced into recovery mode to ensure that no out-of-sequence messages are published.

   Attention: There is no facility to automatically detect all related CCAs. We therefore strongly recommend that a list of all related CCAs be compiled before the event publishing solution is put into production.

In Figure 4-4, there are six related CCAs — IMSO, Batch1, Batch2, Batch3, Batch4 and Batch5.

3. Force all related CCAs into recovery mode.
   This is most easily achieved by stopping the correlation service and releasing the CSA by issuing the following command:

   /P CACCS

   Releasing the CSA depends upon the CSARLSE parameter for the correlation service in the configuration file as discussed in Table 1-3 on page 24. It can also be released by submitting the job shown in Example 4-24.
Example 4-24  CSA release job

```
//CACE2RPT JOB (POK,999),MSGLEVEL=(1,1),MSGCLASS=H,
//    CLASS=A,NOTIFY=&SYSUID
//CACE2RPT PROC CAC='CAC'        INSTALLED HIGH LEVEL QUALIFIER
//*
//CACE2RPT EXEC PGM=CACECA2U,
//    PARM='REPORT,RELEASE'
//*
//STEPLIB DD DISP=SHR,DSN=&CAC..V8R2M0.SCACLOAD
//CTRANS DD DISP=SHR,DSN=&CAC..V8R2M0.SCACSASC
//SYSPRINT DD SYSOUT=*   //SYSTERM DD SYSOUT=*  //PEND
//RUNE2RPT EXEC CACE2RPT
```

Even though recovery mode may have been set for a particular active CCA at a particular point-in-time, the detection of this state may not occur till a much later point-in-time as shown in Figure 4-4.

For example:

- For timeline A in Figure 4-4, the IMSO CCA goes into recovery mode before the SAVINGDB database is deallocated from the online region for batch processing, but the condition is only detected after Batch1 has completed processing and before Batch3 has begun. Also, the forcing of recovery mode occurs while Batch3 is updating the SAVINGDB database.

- For timeline B in Figure 4-4, the IMSO CCA goes into recovery mode before the SAVINGDB database is deallocated from the online region for batch processing, but the condition is only detected while Batch3 is still processing updates. In this case, the forcing of recovery mode occurs before Batch4 begins updating the SAVINGDB database.

**Note:** IMSO updates continue after recovery mode is set, up to the point when SAVINGDB is deallocated.

The related CCAs fall into three categories as follows:

- **Completed CCAs**
  These are CCAs that have completed execution prior to the force point. In Figure 4-4:
  - For timeline A in Figure 4-4, it is Batch1 — these updates have already been processed, and the updates are published out of sequence with the IMSO updates that occurred just prior to the SAVINGDB database being deallocated from the online control region.
  - For timeline B in Figure 4-4, it is Batch3 and its updates are also published out of sequence with the IMSO updates that occurred just prior to the SAVINGDB database being deallocated from the online control region.

- **Executing CCAs:**
  These are related CCAs that are executing when it is forced into recovery mode.
  - For timeline A in Figure 4-4, this is Batch3, and any updates that occurred prior to this point are published out-of-sequence with the IMSO updates that occurred just prior to the SAVINGDB database being deallocated from the online control region.
  - For Timeline B in Figure 4-4, there are no executing CCAs.

\[1\] Release of the CSA only occurs if there are no other correlation services are still active in the system.
– Potential future CCA executions:

These are related CCAs that may be scheduled for future execution, and can include completed CCAs that may be scheduled for execution again in future. All such CCAs must be prevented from starting in active mode which can cause out-of-sequence messages to be published.

- For timeline A in Figure 4-4, these are Batch1, Batch2, Batch4 and Batch5.
- For timeline B in Figure 4-4, these are Batch1, Batch2, Batch3, Batch4 and Batch5.

This can be achieved by inserting a dummy line as shown in Example 4-25 into that CCA’s recovery data set.

**Example 4-25  Dummy line in a CCA’s recovery data set**

<table>
<thead>
<tr>
<th>IMS_DUMMY RECOVERY POINT</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMS_DUMMY RECOVERY POINT</td>
</tr>
</tbody>
</table>

What this achieves is that when this CCA is started/scheduled, it checks its recovery data set and finds an entry in it — this causes the CCA to immediately go into recovery mode.

This needs to be done because the correlation service will need to be started for recovery processing, and we do not want any freshly scheduled CCAs to operate in active mode and transmit out-of-sequence messages to the correlation service.

4. Startup the correlation service CACCS

Recovery processing using the recovery CCA requires the correlation service to be started. None of the related CCAs must be in active mode until the completion of recovery processing.

The correlation service is started by the following command:

```
/S CACCS
```

5. Check all related CCAs

To recover unpublished events from the logs, the correlation service must be aware of the recovery situation — which CCAs are in recovery mode (recovery CCA will only recover them), and which are not (recovery CCA will ignore these).

Therefore, ensure that the correlation service is aware that all the related CCAs are in recovery mode by executing the recovery CCA in ‘C’ mode (check agent). If not, then execute the recovery CCA in ‘S’ mode (set agent in recovery mode) for all those CCAs with a real restart point entry (not a dummy entry) that are not known to the correlation service to be in recovery mode.

**Note:** The recovery CCA checks the restart information in the correlation service recovery data set with the restart information in the CCA’s recovery data set to determine if the correlation service is aware of all the CCAs in recovery mode.

6. Get all logs using LIST.LOG ALL.

To recover unpublished events from the logs, the relevant IMS log data sets must be specified to the recover CCA job. The DBRC output LIST.LOG ALL is used to search for the logs required for each of the CCAs in recovery mode.

**Note:** This output will contain log files for the completed CCAs and executing CCAs, and may include some data sets for “potential future CCA executions” if they happen to be scheduled in the interim.
7. Recover completed and executing CCAs.

The completed and executing CCAs can be recovered now by the recovery CCA using the log data sets identified in the previous step.

**Important:** If all the logs required for recovery processing are *not* provided in a single recovery CCA job, then multiple iterations of the recovery CCA job will need to be run. Each of these iterations will require additional logs to be supplied.

8. Did any “potential future CCA executions” become a completed or executing CCA?

During the recovery processing of completed and executing CCAs, it is possible for one or more of the “potential future CCA executions” to have been scheduled and completed or currently executing — these CCAs were scheduled *after* the correlation service was started. However, they start in recovery mode because of the insertion of a dummy restart point entry into their recovery data set.

- If yes, then:
  - Get the real restart point from the log data sets for this CCA
    
    This CCA still has the dummy restart point entry in the recovery data set. In order to recover the unpublished events in the logs for this CCA, the restart point in the recovery data set *must exactly* match the restart point in its logs where the recovery mode was actually triggered.
    
    For a batch job, this is the first log entry in the first log dataset that was created for this CCA. The restart point consists of the first LSN (always 1 in a batch job) and the System Clock (first system clock specified in the log data set). Figure 4-2 on page 202 describes the format of the restart information that needs to be supplied. Double check the information provided. For batch CCAs, the real key is the oldest living UOR information, but both the current and oldest living UOR system clock and LSN should be the same.
    
  - Set the real restart point in the correlation service recovery data set for this CCA
    
    This is achieved by executing the recovery CCA in ‘S’ mode (set agent in recovery mode) for this CCA.

**Note:** This process may loop several times until no further “potential future CCA executions” change state to completed or executing.

- If no, then go to the next step

**Note:** The change of state of “potential future CCA executions” to completed or currently executing has to be detected by the event publishing administrator (EPA) by some external process such as reviewing the system log for WTO messages.

9. Quiesce IMS and no batch updates.

When all the logs have been input to the recovery CCA job for all CCAs that the correlation service knows to be in recovery mode, and no further updates are occurring in the database, then all the related CCAs can be set to active mode.

Such a state of no update activity is best achieved by quiescing the IMS system or setting the database to read-only.
10. Consistent state?

A consistent state is one where all the logs have been processed and there are no outstanding or ongoing updates occurring in the monitored database.

After quiescing IMS online and ensuring that there are no batch jobs executing, perform one last check for consistency by running recovery CCA now against all the related CCAs known by the correlation service to be in recovery mode. If the consistent state is not achieved, loop back as shown in Figure 4-3 on page 210.

11. Activate all related CCAs.

After achieving a consistent state, all the related CCAs can be made active again by executing the recovery CCA batch job in 'A' mode (place agent in active mode). This job deletes all the restart points in the CCA and correlation service recovery data sets.

Attention: All the dummy entries inserted into the “potential future CCA executions” recovery data sets must be deleted as well by the event publishing administrator.

12. Start IMS and allow batch updates.

Normal processing of the IMS online region and batch updates against the database can now resume.

4.4 Complex recovery mode scenario

We describe a complex event publishing environment involving multiple CCAs that experiences a recovery mode situation, and needs to be recovered without loss of event publishing messages. We discuss some of the challenges of recovery, and apply our recommended approach described in Section 4.3, “Recovery mode resolution recommended approach” on page 209 to recover the event publishing environment.

The normal processing environment is that the AUTODB database (which contain segments that are subject to event publishing) is updated in serial fashion — the database is updated initially by the online region (IMS810K), then deallocated from the online region, updated by batch jobs (CACBACH1, CACBACH2, and CACBACH3), reallocated to the online region after the batch jobs completes to be again updated by the IMS online region. Table 4-1 shows the details of the IMS CCAs in this environment.

<table>
<thead>
<tr>
<th>CCA name</th>
<th>Type</th>
<th>Recovery data set</th>
<th>Log data sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMS810K</td>
<td>IMS online</td>
<td>CAC.RECOVDS.IMS810K</td>
<td>Not known at this point</td>
</tr>
<tr>
<td>CACBACH1</td>
<td>IMS batch</td>
<td>CAC.RECOVDS.CACBACH1</td>
<td>NALUR.LOG1</td>
</tr>
<tr>
<td>CACBACH2</td>
<td>IMS batch</td>
<td>CAC.RECOVDS.CACBACH2</td>
<td>NALUR.LOG2</td>
</tr>
<tr>
<td>CACBACH3</td>
<td>IMS batch</td>
<td>CAC.RECOVDS.CACBACH3</td>
<td>NALUR.LOG3</td>
</tr>
</tbody>
</table>
The following considerations apply to the this complex recovery mode scenario:

- The auto insurance IMS database AUTODB database described in Chapter 3, “Trustworthy Insurance Customer Information Facility (CIF) and Event Alert System scenario” on page 85 is the data source.
- DBRC controls both the IMS online and batch jobs running in the event publishing environment.
- Recovery data sets are defined for each CCA as shown in Table 4-1.
- No GDG is used for the log used by the batch jobs CACBACH1, CACBACH2 and CACBACH3.

Figure 4-5 shows one particular timeline of operations where the IMS online region CCA IMS810K has recovery mode set (for a reason other than the correlation service going down) at time ‘1’.

Attention: A Microsoft Power Point animated slide show presentation of this scenario can be downloaded from ftp://www.redbooks.ibm.com/redbooks/SG247132

Figure 4-5  Complex recovery mode scenario

The occurrence of recovery mode at time ‘1’ is detected at some point between time ‘1’ and ‘8’, and all related CCAs (CACBACH1, CACBACH2 and CACBACH3) are forced into recovery mode sometime between time ‘8’ and time ‘9’. Recovery is then initiated beginning with starting the correlation service (time ‘11’) and ending with activating all the four CCAs (time ‘26’).
Each of the activities at each point in time is discussed in the following sections:

1. Normal processing prior to time ‘1’ involves updating of the AUTODB database and its changes being published to target WebSphere MQ queues.

2. WTO messages

   To simulate our complex recovery scenario, the IMS online region is artificially forced into recovery mode as follows:
   a. Shut down the correlation service by issuing the /P CACCS2 command.
   b. Update the AUTODB database, which causes the IMS control region try to communicate with the correlation service — when it fails to find the correlation service, it issues a WTOR message as follows and writes the restart information to its recovery data set CAC.RECOVDS.IMS810K:

   ```
   CACH002A EVENT PUBLISHER SERVICE '(noname)' NOT FOUND BY
   CHANGE-CAPTURE AGENT 'IMS_IMS810K     ', REPLY 'R' OR 'A' RECOVER
   Reply /nnn, R
   ```
   c. Warm start the correlation service by issuing the /S CACCS2 command.

   **Attention:** We executed this sequence of steps to cause the scenario where the IMS online region CCA goes into recovery mode, with the correlation service being unaware of this (recovery mode) condition.

3. Are there related CCAs?

   The next step is to determine all the related CCAs. This is a manual process and all the related CCAs are identified prior to the event publishing solution going into production as listed in Table 4-1.

4. Force all related CCAs into recovery mode.

   This corresponds to times ‘6’, ‘7’, ‘9’ and ‘10’. Once all the four CCAs have been identified, you need to force all the related CCAs into recovery mode as follows:
   a. Completed CCAs:

      The only completed CCA is batch job CACBACH1 which completed before the forcing of recovery mode was initiated. Since it was not in recovery mode, event messages generated by the batch jobs are published out-of-sequence with the changes that have occurred in the IMS online region after it was set in recovery mode.

   **Important:** The out of sequence messages may be an exposure in many event publishing solutions. To minimize the number of out of sequence messages, it is safer to detect recovery mode states as soon as they occur, and force recovery mode for all related CCAs.

   b. Executing CCAs:

      The CACBACH2 job is running when the force of recovery mode is initiated. This CCA is set to recovery mode by shutting down the correlation service with the /P CACCS2 command. Then release the CSA by running the CACE2RPT job with the RELEASE option resulting in the output shown in Example 4-26.

   **Example 4-26 Release CSA using CACE2RPT job**

   ```
   +---------- Event Publisher Correlation Service Report -----------------
   | Correlation Service CSA E2CSGBLA, Lth=2588, Lock=x0000
   ```
Correlation Service CSA Storage has been released.

When the batch job CACBACH2 continues with its updates, the CCA does not find the correlation service and writes the CACH002A WTOR message identified earlier, and updates the CAC.RECOVDS.CACBACH2 recovery data set with the restart point. The WTOR message should be replied with ‘R’.

However, all changes generated prior to the forcing of recovery mode will have been published out of sequence with the changes that have occurred in the IMS online region after it was set in recovery mode.

c. Possible future CCA executions

The CACBACH3 job has not yet begun to run, but it needs to be prevented from running in active mode by inserting a dummy entry in the CAC.RECOVDS.CACBACH3 recovery data set as shown in Example 4-27 using TSO ISPF option 3.4.

Example 4-27  Insert dummy restart point in the CAC.RECOVDS.CACBACH3 recovery data set

| IMS_DUMMY RP |

Note: There is no need to specify an accurate restart point in the recovery data set.

Example 4-28  /S CACCS2 command output

| 17.53.47 STC02332 CAC00105I LOG V8.2 03102005: STARTED |
| 17.53.47 STC02332 CAC00100I CONTROLLER: LOGGING STARTED |
| 17.53.47 STC02332 CACG001I CORRELATION SERVICE IS WARM STARTING |
| 17.53.47 STC02332 CACG002I CORRELATION SERVICE RUNNING IN STORAGE KEY 08 |
| 17.53.47 STC02332 CACG005I CHANGE CAPTURE SET FOR IMS TABLE IMSAUTO.ADDRESS |
| 17.53.47 STC02332 CACG005I CHANGE CAPTURE SET FOR IMS TABLE IMSAUTO.CLAIMS |
| 17.53.47 STC02332 CACG005I CHANGE CAPTURE SET FOR IMS TABLE IMSAUTO.CUSTOMER |
| 17.53.47 STC02332 CACG005I CHANGE CAPTURE SET FOR IMS TABLE IMSAUTO.POLICY |
| 17.53.47 STC02332 CACG005I CHANGE CAPTURE SET FOR IMS TABLE IMSAUTO.PREMIUMS |

Note: There is no need to specify an accurate restart point in the recovery data set.

Important: Even though batch job CACBACH1 completed, a dummy entry should be inserted into the CAC.RECOVDS.CACBACH1 recovery data set as well, since the job could be rerun again.

At this point, all the related CCAs are in recovery mode, or will be in recovery mode when they startup the next time around.

5. Startup the correlation service CACCS

Next step is to start the recovery process by first starting the correlation service via the /S CACCS2 command — the output of this command is shown in Example 4-28, which indicates that the correlation service is unaware of the recovery mode situation of the CCAs since there are no “CACG111W AGENT ‘IMS_IMS810K’ IN RECOVERY MODE ..........” messages.
6. Check all related CCAs.

This is achieved by executing the recovery CCA job in ‘C’ mode (check agent) with the control cards shown in Example 4-29. — the output is shown in Example 4-30. The restart information in Example 4-30 for each related CCA is as follows:

- IMS810K is in recovery mode, and has the correct restart point in its recovery dataset but has no restart information in the correlation service recovery data set.
- CACBACH1 is in recovery mode, and has dummy restart point in its recovery dataset but no restart information in the correlation service recovery data set.
- CACBACH2 is in recovery mode, and has the correct restart point in its recovery dataset but no restart information in the correlation service recovery data set.
- CACBACH3 is in recovery mode, and has the dummy restart point in its recovery dataset but no restart information in the correlation service recovery data set.

Example 4-29  Recovery CCA ‘C’ mode (check agent) control cards

C  IMS810K  4  CAC.RECOVDS.IMS810K
C  CACBACH1 4  CAC.RECOVDS.CACBACH1
C  CACBACH2 4  CAC.RECOVDS.CACBACH2
C  CACBACH3 4  CAC.RECOVDS.CACBACH3

Example 4-30  Recovery CCA ‘C’ mode (check agent) output

17.58.12  J0BO2333  +CACH061I  RECOVERY MODE: CHECK AGENT STATUS
17.58.12  J0BO2333  +CACH048I  RECOVERY DATASET CAC.RECOVDS.IMS810K
17.58.12  J0BO2333  + OPENED
17.58.12  J0BO2333  +CACH049I  AGENT NAME IS 'IMS810K '
17.58.12  J0BO2333  +CACH039I  DB2 RESTART TIME    20050913 17044047
17.58.12  J0BO2333  +CACH040I  IMS RESTART TIME    05.256 17:04:40.4
17.58.12  J0BO2333  +CACH037I  RESTART SYSTEM CLOCK BD9BDB8D8B0D300
17.58.12  J0BO2333  +CACH038I  RESTART LOG SEQ. #  00000000-0000022F
17.58.12  J0BO2333  +CACH030I  AGENT 'IMS810K ' IS NOT IN RECOVERY MODE
17.58.12  J0BO2333  +CACH067I  LOG TRACKING FILE DOES NOT EXIST
17.58.12  J0BO2333  +CACH048I  RECOVERY DATASET CAC.RECOVDS.CACBACH1
17.58.12  J0BO2333  + OPENED
17.58.12  J0BO2333  +CACH049I  AGENT NAME IS 'DUMMY RP'
17.58.13  J0BO2333  +CACH039I  DB2 RESTART TIME    40404040 40404040
17.58.13  J0BO2333  +CACH040I  IMS RESTART TIME    40.365 40:40:40.4
17.58.13  J0BO2333  +CACH037I  RESTART SYSTEM CLOCK BD9BDB8D0DBD300
17.58.13  J0BO2333  +CACH038I  RESTART LOG SEQ. #  00000000-00000001
17.58.13  J0BO2333  +CACH030I  AGENT 'CACBACH1' IS NOT IN RECOVERY MODE
17.58.13  J0BO2333  +CACH067I  LOG TRACKING FILE DOES NOT EXIST
17.58.13  J0BO2333  +CACH048I  RECOVERY DATASET CAC.RECOVDS.CACBACH2
17.58.13  J0BO2333  + OPENED
17.58.13  J0BO2333  +CACH049I  AGENT NAME IS 'CACBACH2'
17.58.13  J0BO2333  +CACH039I  DB2 RESTART TIME    20050913 17303172
17.58.13  J0BO2333  +CACH040I  IMS RESTART TIME    05.256 17:30:31.7
17.58.13  J0BO2333  +CACH037I  RESTART SYSTEM CLOCK BD9B3EC9599586A5
17.58.13  J0BO2333  +CACH038I  RESTART LOG SEQ. #  00000000-00000001
17.58.13  J0BO2333  +CACH030I  AGENT 'CACBACH2' IS NOT IN RECOVERY MODE
7. Is the correlation service aware of all related CCAs in recovery mode?

The IMS810K and CACBACH2 CCAs are the executing and completed CCAs that are not recognized by the correlation service as being in recovery mode. Therefore, the recovery CCA job must be executed in ‘S’ mode (set agent in recovery mode) using the control cards shown in Example 4-31 for the IMS810K and CACBACH2 CCAs — the output is shown in Example 4-32.

Example 4-31  Recovery CCA ‘S’ mode (set agent in recovery mode) control cards

<table>
<thead>
<tr>
<th>Command</th>
<th>Dataset</th>
<th>Output Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>S IMS810K 4</td>
<td>CAC.RECOVDS.IMS810K</td>
<td>+CACH061I RECOVERY MODE: SET AGENT IN RECOVERY</td>
</tr>
<tr>
<td>S CACBACH2 4</td>
<td>CAC.RECOVDS.CACBACH2</td>
<td>+CACH039I AGENT 'IMS810K' IS NOW IN RECOVERY MODE</td>
</tr>
</tbody>
</table>

Example 4-32  Recovery CCA ‘S’ mode (set agent in recovery mode) output

<table>
<thead>
<tr>
<th>Command</th>
<th>Dataset</th>
<th>Output Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>+CACH0501I</td>
<td>RECOVERY DATASET CAC.RECOVDS.IMS810K</td>
<td>+CACH061I RECOVERY MODE: SET AGENT IN RECOVERY</td>
</tr>
<tr>
<td>+CACH052I</td>
<td>AGENT 'IMS810K' IS NOW IN RECOVERY MODE</td>
<td>+CACH039I AGENT 'IMS810K' IS NOW IN RECOVERY MODE</td>
</tr>
<tr>
<td>+CACH039I</td>
<td>DB2 RESTART TIME 20050913 17044047</td>
<td>+CACH0401 IMS RESTART TIME 05.256 17:04:40.4</td>
</tr>
<tr>
<td>+CACH037I</td>
<td>DB2 RESTART TIME 20050913 17044047</td>
<td>+CACH0401 IMS RESTART TIME 05.256 17:04:40.4</td>
</tr>
<tr>
<td>+CACH038I</td>
<td>RESTART SYSTEM CLOCK BD9BE3CD959BE5A5</td>
<td>+CACH0671 LOG TRACKING FILE DOES NOT EXIST</td>
</tr>
<tr>
<td>+CACH048I</td>
<td>RECOVERY DATASET CAC.RECOVDS.CACBACH2</td>
<td>+CACH061I RECOVERY MODE: SET AGENT IN RECOVERY</td>
</tr>
<tr>
<td>+CACH052I</td>
<td>AGENT 'CACBACH2' IS NOW IN RECOVERY MODE</td>
<td>+CACH039I AGENT 'CACBACH2' IS NOW IN RECOVERY MODE</td>
</tr>
<tr>
<td>+CACH039I</td>
<td>DB2 RESTART TIME 20050913 17303172</td>
<td>+CACH0401 IMS RESTART TIME 05.256 17:30:31.7</td>
</tr>
<tr>
<td>+CACH037I</td>
<td>DB2 RESTART TIME 20050913 17303172</td>
<td>+CACH0401 IMS RESTART TIME 05.256 17:30:31.7</td>
</tr>
<tr>
<td>+CACH038I</td>
<td>RESTART SYSTEM CLOCK BD9BE3CD959BE5A5</td>
<td>+CACH0671 LOG TRACKING FILE DOES NOT EXIST</td>
</tr>
<tr>
<td>+CACH062W</td>
<td>RECOVERY PROCESSING ENDED WITH WARNINGS</td>
<td>+CACH061I RECOVERY MODE: SET AGENT IN RECOVERY</td>
</tr>
</tbody>
</table>

8. Get all logs LIST.LOG ALL.

Request a DBRC LIST.LOG ALL report to produce a report shown in Example 4-33.

This DBRC LIST.LOG ALL output must be used as input to the REXX based tool\(^2\) described in Appendix B, “REXX based tool to identify log files for the recovery change capture agent (CCA)” on page 241 with a configuration file shown in Example 4-34 on page 224 that has the following input:

- IMS810K with time of restart point as given in the latest recovery CCA ‘C’ mode (check agent) job (05.256 16:53:23.7)

\(^2\) During the development of this redbook, we thought that it would be nice to have a script that helped the administrator to identify all the relevant logs for input to the IMS recovery CCA job. We therefore developed a REXX script to help the administrator identify all the IMS logs required to recover the system.
- CACBACH1 with time of forcing into recovery mode (05.256 17:31:00.0)
- CACBACH2 with time as given in the latest recovery CCA ‘C’ mode (check agent) job (05.256 17:30:31.7)
- CACBACH3 with time of forcing into recovery mode (05.256 17:31:00.0)

The output of the REXX based tool with this input is shown in Example 4-35 on page 225.

Example 4-33  DBRC LIST.LOG ALL (partial) output

0-------------------------------------------------------------
PRILOG  RECORD SIZE= 304
START = 05.256 16:27:25.9  *  SSID=IMSK  VERSION=8.1
STOP  = 00.000 00:00:00.0  #DSN=1
GSGNAME=**NULL**
FIRST RECORD ID= 0000000000000001 PRILOG TOKEN= 0
EARLIEST CHECKPOINT = 05.256 16:27:28.2

DSN=IMS810K.RLSP.IMSK.D05256.T1627259.V18  UNIT=3390
START = 05.256 16:27:25.9  FIRST DS LSN= 0000000000000001
STOP  = 05.256 18:11:42.5  LAST DS LSN= 00000000000002C2
FILE SEQ=0001  #VOLUMES=0001
VOLSER=TOTMIN STOPTIME = 05.256 18:11:42.5
CKPTCT=2  CHKPT ID = 05.256 17:04:40.4
LOCK SEQUENCE#  000000000000

LOGALL
START  = 05.256 16:27:25.9  *
EARLIEST ALLOC TIME = 00.000 00:00:00.0
DBDS ALLOC=0

105.256 18:33:25.2  LISTING OF RECON  PAGE 0243
0-------------------------------------------------------------
PRILOG  RECORD SIZE= 304
START = 05.256 16:27:25.9  *  SSID=IMSK  VERSION=8.1
STOP  = 00.000 00:00:00.0  #DSN=1
GSGNAME=**NULL**
FIRST RECORD ID= 0000000000000001 PRILOG TOKEN= 0

DSN=IMS810K.RLSP.IMSK.D05256.T1627259.V18  UNIT=3390
START = 05.256 16:27:25.9  FIRST DS LSN= 0000000000000001
STOP  = 05.256 18:11:42.5  LAST DS LSN= 00000000000002C2
FILE SEQ=0001  #VOLUMES=0001
CHECKPOINT TYPES=80: SIMPLE=Y SNAPQ=N DUMPQ=N PURGE=N FREEZE=N

VOLSER=TOTMIN STOPTIME = 05.256 18:11:42.5
CKPTCT=2  CHKPT ID = 05.256 17:04:40.4
LOCK SEQUENCE#  000000000000

105.256 18:33:25.2  LISTING OF RECON  PAGE 0244
0-------------------------------------------------------------
PRILOG  RECORD SIZE= 304
START = 05.256 17:27:15.4  *  SSID=CACBACH1 VERSION=8.1
STOP  = 05.256 17:27:16.0  #DSN=1
GSGNAME=**NULL**
FIRST RECORD ID= 0000000000000001 PRILOG TOKEN= 0

DSN=NALUR.LOG1  UNIT=3390
START = 05.256 17:27:15.4  FIRST DS LSN= 0000000000000001
STOP  = 05.256 17:27:16.0  LAST DS LSN= 0000000000000001E
FILE SEQ=0001  #VOLUMES=0001
VOLSER=TSTO29 STOPTIME = 05.256 17:27:16.0
CKPTCT=0  CHKPT ID = 00.000 00:00:00.0
LOCK SEQUENCE#= 000000000000

LOGALL
START = 05.256 17:27:15.4  *
EARLIEST ALLOC TIME = 00.000 00:00:00.0
DBDS ALLOC=0

105.256 18:33:25.2  LISTING OF RECON  PAGE 0245

PRILOG
RECORD SIZE= 304
START = 05.256 17:30:31.6  *
STOP = 05.256 17:30:32.1  #DSN=1
GSGNAME=**NULL**
FIRST RECORD ID= 0000000000000001
PRILOG TOKEN= 0

DSN=NALUR.LOG2
UNIT=3390
START = 05.256 17:30:31.6
STOP = 05.256 17:30:32.1
FILE SEQ=0001  #VOLUMES=0001

VOLSER=TSTO15 STOPTIME = 05.256 17:30:32.1
CKPTCT=0  CHKPT ID = 00.000 00:00:00.0
LOCK SEQUENCE#= 000000000000
LOGALL
START = 05.256 17:30:31.6  *
EARLIEST ALLOC TIME = 00.000 00:00:00.0
DBDS ALLOC=0

105.256 18:33:25.2  LISTING OF RECON  PAGE 0246

PRIOLD
SSID=IMSK  # DD ENTRIES=7
EARLIEST CHECKPOINT = 05.256 16:27:28.2

DDNAME=DFSOLP03  DSN=IMS810K.OLP03
START = 05.252 13:18:48.7 FIRST DS LSN= 0000000000036A2
STOP = 05.252 17:24:11.3 LAST DS LSN= 00000000000385D
LOCK SEQUENCE# = 000000000000
STATUS=ARC COMPLT FEOV=NO  AVAIL
PRILOG TIME=05.252 13:18:48.7 ARCHIVE JOB NAME=IVPGNJCL
VERSION=8.1

DDNAME=DFSOLP04  DSN=IMS810K.OLP04
START = 05.252 17:26:28.1 FIRST DS LSN= 000000000000385E
STOP = 05.255 20:00:48.7 LAST DS LSN= 00000000000437D
LOCK SEQUENCE# = 000000000000
STATUS=ARC COMPLT FEOV=NO  AVAIL
PRILOG TIME=05.252 17:26:28.1 ARCHIVE JOB NAME=IVPGNJCL
VERSION=8.1

DDNAME=DFSOLP05  DSN=IMS810K.OLP05
START = 05.255 20:03:23.5 FIRST DS LSN= 00000000000437E
STOP = 05.255 20:03:54.5 LAST DS LSN= 000000000004CB3
LOCK SEQUENCE# = 000000000000
STATUS=ARC COMPLT FEOV=NO  AVAIL
PRILOG TIME=05.255 20:03:23.5 ARCHIVE JOB NAME=IVPGNJCL
VERSION=8.1

DDNAME=DFSOLP99  DSN=IMS810K.OLP99
START = 05.256 16:08:11.9 FIRST DS LSN= 0000000000004CB4
STOP = 05.256 16:08:41.1 LAST DS LSN= 00000000000055ED
LOCK SEQUENCE# = 000000000000
STATUS=ARC COMPLT
PRILOG TIME=05.256 16:08:11.9
VERSION=8.1
DDNAME=DFSOLP02 DSN=IMS810K.OLP02
START = 05.256 16:23:32.3 FIRST DS LSN= 0000000000006877
STOP = 05.256 16:24:03.3 LAST DS LSN= 00000000000071CF
LOCK SEQUENCE# = 000000000000
STATUS=ARC COMPLT
PRILOG TIME=05.256 16:23:32.3
VERSION=8.1
DDNAME=DFSOLP00 DSN=IMS810K.OLP00
START = 05.256 16:27:25.9 FIRST DS LSN= 0000000000000001
STOP = 05.256 16:27:33.2 LAST DS LSN= 0000000000000002C2
LOCK SEQUENCE# = 000000000000
STATUS=ARC COMPLT
PRILOG TIME=05.256 16:27:25.9
VERSION=8.1
DDNAME=DFSOLP01 DSN=IMS810K.OLP01
START = 05.252 13:18:48.7 FIRST DS LSN= 00000000000036A2
STOP = 05.252 17:24:11.3 LAST DS LSN= 000000000000385D
LOCK SEQUENCE# = 000000000000
STATUS=ACTIVE
PRILOG TIME=05.252 13:18:48.7
VERSION=8.1
DDNAME=DFSOLS03 DSN=IMS810K.OLS03
START = 05.252 13:18:48.7 FIRST DS LSN= 00000000000036A2
STOP = 05.252 17:24:11.3 LAST DS LSN= 000000000000385D
LOCK SEQUENCE# = 000000000000
PRILOG TIME=05.252 13:18:48.7
VERSION=8.1
DDNAME=DFSOLS04 DSN=IMS810K.OLS04
START = 05.252 17:26:28.1 FIRST DS LSN= 000000000000385E
STOP = 05.255 20:00:48.7 LAST DS LSN= 000000000000437D
LOCK SEQUENCE# = 000000000000
PRILOG TIME=05.252 17:26:28.1
VERSION=8.1
DDNAME=DFSOLS05 DSN=IMS810K.OLS05
START = 05.255 20:03:23.5 FIRST DS LSN= 000000000000437E
STOP = 05.255 20:03:54.5 LAST DS LSN= 0000000000004CB3
LOCK SEQUENCE# = 000000000000
PRILOG TIME=05.255 20:03:23.5
VERSION=8.1
DDNAME=DFSOLS99 DSN=IMS810K.OLS99
Example 4-34  REXX based tool configuration file

/* Give log types as abbreviations:
   AP   Archived Primary log
   AS   Archived Secondary log
   ASP  Archived Primary System log
   ASS  Archived Secondary System log
   OP   Primary Online log
   OS   Secondary Online log
*/
#
/* First SSID (online IMS in this case) */
SSID=IMSK
RESTART=05.256 16:53:23.7
LOGTYPE=AP
LOGTYPE=OP
#
/* Second SSID */
SSID=CACBACH1
RESTART=05.256 17:31:00.0
#
/* Third SSID */
SSID=CACBACH2
RESTART=05.256 17:30:31.7
#
/* Fourth SSID */
SSID=CACBACH3
RESTART=05.256 17:31:00.0
#
/* Fake SSID, used to show all logs after RESTART */
/* Specify an arbitrary time approx. 30 minutes */
/* prior to the earliest restart point. */
SSID=LIST ME
#RESTART=05.240 18:00:00.0
#
#
#
#
/* Name of the file that holds DBRC output */
DBRCLIST='NEALE.OUTPUT(LIST4)'
#DBRCLIST=dbrcoutput
#
/* Name of the file to write output to */
NEEDLOG=recover_these_logs.list
#
/* Width of the output (timeline) */
COLUMNS=120

Example 4-35   REXX based tool graphic output

#### Print 'Graphic' Log Timeline ####

| IMSK | AP | 05.256 16:27:25.9 IMS810K.RLDSP.IMSK.D05256.T1627259.V18 |
| IMSK | ASP | 05.256 16:27:25.9 IMS810K.SLDSP.IMSK.D05256.T1627259.V18 |
| IMSK | OP | 05.256 16:27:25.9 IMS810K.OLP00 |
| CACBACH1 | AP | 05.256 17:27:15.4 NALUR.LOG1 |
| CACBACH2 | AP | 05.256 17:30:31.6 NALUR.LOG2 |
| CACBACH3 | AP | 05.256 18:11:42.5 IMS810K.OLP01 |

Symbols:
< Start of log dataset.
= Log dataset is open.
> Stop of log dataset.
X Stop and Start of log dataset (log switch).
! Restart point of this SSID is here.
~ Restart point of this SSID is before this.
? Error while constructing graphical timeline.
9. Recover completed and executing CCAs.

- **IMS810K CCA**
  
  Example 4-35 shows a timeline indicating that you need logs IMS810K.OLP00 (instead of this primary online log, you could also use the archived primary log IMS810K.RLDSP.IMSK.D05256.T1627259.V18, or the archived primary system log IMS810K.SLDSP.IMSK.D05256.T1627259.V18) and IMS810K.OLP01 primary online log to recover the IMS810K CCA.

- **CACBACH2 CCA**
  
  This CCA went into recovery mode when the correlation service was stopped, and you need log dataset NALUR.LOG2 to recover the event publishing messages in it.

- **CACBACH3**
  
  This CCA has not yet started.

**Attention:** Since the batch job CACBACH1 finished before it was forced into recovery mode, all the updates in NALUR.LOG1 were published out-of-sequence. You need to evaluate this condition and take appropriate action.

The recovery CCA job must be run in 'L' mode (log file recovery) with the control cards shown in Example 4-36 to recover the CCAs IMS810K and CACBACH2. The output is shown in Example 4-37.

Appropriate messages corresponding to the events in the logs are published to their respective target queues.

**Example 4-36  Recovery CCA ‘L’ mode (log file recovery) control cards**

```
L IMS810K 4 IMS810K.OLP00
L IMS810K 4 IMS810K.OLP01
L CACBACH2 4 NALUR.LOG2
```

**Example 4-37  Recovery CCA ‘L’ mode (log file recovery) output**

```
+CACH061I RECOVERY MODE: IMS LOG FILE RECOVERY
+CACH055I STARTING LOG FILE SEQUENCE CHECKING
+CACH077I AGENT 'IMS810K' EXTRANEOUS LOG FILE: IMS810K.OLP01
+CACH058I LOG FILE SEQUENCE CHECKING COMPLETED
+CACH044I AGENT 'IMS810K' LOG OPENED: IMS810K.OLP01
+CACH046I AGENT 'IMS810K' LOG IGNORED: IMS810K.OLP01
+CACH045I AGENT 'IMS810K' LOG CLOSED: IMS810K.OLP01
+CACH044I AGENT 'IMS810K' LOG OPENED: IMS810K.OLP00
+CACH044I AGENT 'IMS810K' LOG OPENED: NALUR.LOG2
+CACH044I AGENT 'CACBACH2' LOG OPENED: NALUR.LOG2
+CACH045I AGENT 'CACBACH2' LOG CLOSED: NALUR.LOG2
+CACH045I AGENT 'IMS810K' LOG CLOSED: IMS810K.OLP00
+CACH072I BUFFERS SENT 3 RECORDS 9 THROTTLES 3
+CACH062I RECOVERY PROCESSING COMPLETED SUCCESSFULLY
```

In the timeline shown in Figure 4-5, the batch job CACBACH3 is started and goes into recovery mode because of the dummy entry we inserted earlier in its CAC.RECOVDS.CACBACH3 recovery data set. It writes a WTOR message as follows:

```
CACH003A RECOVERY RECORD EXISTS FOR CHANGE-CAPTURE AGENT 'IMS_CACBACH3', REPLY 'R' OR 'A' RECOVERY/ACTIVE
```

Reply /nnn,R to this message.
10. Did any “not yet scheduled” CCAs become completed or executing CCAs?

This information (which should identify the CACBACH3 CCA as an executing CCA) can be obtained by rerunning the REXX based tool using the same configuration file used earlier after generating a new DBRC LIST.LOG ALL report — this would identify the new log NALUR.LOG3 created by the CACBACH3. The output of the REXX based tool is shown in Figure 4-38.

**Example 4-38  REXX based tool graphic output**

```
 **** Print 'Graphic' Log Timeline ****

IMSK  AP : <=1================================>
IMSK  ASP: <=2================================>
IMSK  OP : <=3=================X=15=====>
IMSK  * : !~~~~~~~~~~~~~~~~~~~~~~~~~~

CACBACH1 AP : <=5=>
CACBACH1  * : !~~~~~~~~~~~~~~~~~~~~~~~~~~

CACBACH2 AP : <=7=>
CACBACH2  * : !~~~~~~~~~~~~~~~~

CACBACH3  * : !~~~~~~~~~~~~~~~~
CACBACH3 AP : <=16=>

List of Logs and DSNs:

#### SSID Typ start

<table>
<thead>
<tr>
<th>DSN</th>
<th>SSID</th>
<th>Typ</th>
<th>start</th>
<th>Data Set Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 IMSK</td>
<td>AP</td>
<td>05.256 16:27:25.9</td>
<td>IMS810K.RLDSP.IMSK.D05256.T1627259.V18</td>
<td></td>
</tr>
<tr>
<td>2 IMSK</td>
<td>ASP</td>
<td>05.256 16:27:25.9</td>
<td>IMS810K.SLDSP.IMSK.D05256.T1627259.V18</td>
<td></td>
</tr>
<tr>
<td>3 IMSK</td>
<td>OP</td>
<td>05.256 16:27:25.9</td>
<td>IMS810K.OLP00</td>
<td></td>
</tr>
<tr>
<td>5 CACBACH1 AP</td>
<td>05.256 17:27:15.4</td>
<td>NALUR.LOG1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7 CACBACH2 AP</td>
<td>05.256 17:30:31.6</td>
<td>NALUR.LOG2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15 IMSK</td>
<td>OP</td>
<td>05.256 18:11:42.5</td>
<td>IMS810K.OLP01</td>
<td></td>
</tr>
<tr>
<td>16 CACBACH3 AP</td>
<td>05.256 20:10:34.1</td>
<td>NALUR.LOG3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

11. Get real restart point from the log data sets.

The restart point of the CACBACH3 CCA stored in its CAC.RECOVDS.CACBACH3 recovery data set is a dummy restart point. This has to be replaced with the correct restart point in order to perform recovery on the CACBACH3 CCA.

For a batch job (which is what CACBACH3 is), the correct restart point is simply the first log entry of the log dataset NALUR.LOG3. The log sequence number (LSN) for the first log entry of a batch job is always 1. To complete the restart information, you need to determine the system clock time for the first entry of NALUR.LOG3 by opening the NALUR.LOG3 data set in browse mode as shown in Figure 4-39. The first system clock entry is BD9C07932E680140 which needs to be inserted into the CAC.RECOVDS.CACBACH3 recovery data set of the CACBACH3 CCA as follows:

- Copy the entry of CACBACH2 to CACBACH3.
- Change the LSN and SYSTEM Clock information to ‘1’ and BD9C07932E680140 respectively — the timestamp is optional information and can be ignore.

The modified CAC.RECOVDS.CACBACH3 recovery data set is shown in Figure 4-40.
Example 4-39   Browse NALUR.LOG3 to determine system clock time

Example 4-40   CAC.RECOVDS.CACBACH3 recovery data set restart information entry

Example 4-41   Recovery CCA ‘S’ mode (set agent in recovery mode) control cards

Example 4-42   Recovery CCA ‘S’ mode (set agent in recovery mode) output

The recovery CCA ‘L’ mode (log file recovery) should be run with the control cards shown in Example 4-43. The output is shown in Example 4-44. Appropriate messages corresponding to the events in the logs are published to their respective target queues.

Attention: This process may have to be repeated multiple time in the real world — we chose to run this only once in our controlled environment.

Example 4-43   Recovery CCA ‘L’ mode (log file recovery) control cards
Example 4-44  Recovery CCA 'L' mode (log file recovery) output

+CACH061I RECOVERY MODE: IMS LOG FILE RECOVERY
+CACH055I STARTING LOG FILE SEQUENCE CHECKING
+CACH077I AGENT 'IMS810K' 'EXTRANEOUS LOG FILE: IMS810K.OLP01
+CACH058I LOG FILE SEQUENCE CHECKING COMPLETED
+CACH044I AGENT 'IMS810K' 'LOG OPENED: IMS810K.OLP01
+CACH046I AGENT 'IMS810K' 'LOG IGNORED: IMS810K.OLP01
+CACH045I AGENT 'IMS810K' 'LOG CLOSED: IMS810K.OLP01
+CACH044I AGENT 'CACBACH2' 'LOG OPENED: NALUR.LOG2
+CACH044I AGENT 'CACBACH3' 'LOG OPENED: NALUR.LOG3
+CACH045I AGENT 'CACBACH2' 'LOG CLOSED: NALUR.LOG2
+CACH045I AGENT 'IMS810K' 'LOG CLOSED: IMS810K.OLP00
+CACH045I AGENT 'CACBACH3' 'LOG CLOSED: NALUR.LOG3
+CACH072I BUFFERS SENT 4 RECORDS 13 THROTTLES 4
+CACH062I RECOVERY PROCESSING COMPLETED SUCCESSFULLY

Example 4-45  Recovery CCA 'A' mode (place an agent in active mode) control cards

A IMS810K  4 CAC.RECOVDS.IMS810K
A CACBACH2 4 CAC.RECOVDS.CACBACH2
A CACBACH3 4 CAC.RECOVDS.CACBACH3

Example 4-46  Recovery CCA 'A' mode (place an agent in active mode) output

+CACH061I RECOVERY MODE: ACTIVATE AGENT
+CACH054I PREPARING TO ACTIVATE AGENT 'IMS810K'
+CACH031I AGENT 'IMS810K' SWITCHING TO ACTIVE MODE
+CACH054I PREPARING TO ACTIVATE AGENT 'CACBACH2'
+CACH031I AGENT 'CACBACH2' SWITCHING TO ACTIVE MODE
+CACH054I PREPARING TO ACTIVATE AGENT 'CACBACH3'
+CACH031I AGENT 'CACBACH3' SWITCHING TO ACTIVE MODE
+CACH062I RECOVERY PROCESSING COMPLETED SUCCESSFULLY

Important: The dummy restart entry in the CAC.RECOVDS.CACBACH1 recovery data set should be deleted since it remained in the "not yet scheduled" CCAs.
16. Start IMS and allow batch updates.

   With the recovery process completed for the CCAs, the IMS control region can be started again using the following commands:
   
   \texttt{/S IMS810K}
   \texttt{/nnn,/NRE}

   Batch updates can resume now as well.

   This completes the recovery of the complex recovery mode scenario.
XML message structure and handling

In this appendix we provide an overview of the structure of XML messages for event publishing, describe the different approaches to processing these messages, and introduce the XML Publishing Toolkit.

The topics covered are:
- Types of XML messages for event publishing
- Processing XML event messages
- RFHUTIL tool
- XML Publishing Toolkit
A.1 Types of XML messages for event publishing


Note: In PTF UK09686 released on December 5th 2005, messages may also be published in native EBCDIC format.

There are three types of XML event messages, as follows:

- Data messages:
  These messages describe the changed data captured from the source system and include all or part of a transaction (TRANS message type), or a single row operation (ROWOP message type) within a transaction.

Note: Both WebSphere Information Integrator Classic Event Publisher and WebSphere Information Integrator Event Publisher generate these types of messages.

The content of XML messages for event publishing is described in the definition of an XML publication — parameters include BEFORE_VALUES (YES or NO), TOPIC and MSGTYPE (TRANS or ROWOP).

- BEFORE_VALUES (YES or NO):
  The default value of NO specifies that the before values of an updated row are not included in the XML message — a value of YES will include the before values.

- TOPIC:
  Specifying a topic enables an XML message consuming application to subscribe to a publishing message queue and only receive messages for topics they are interested in.

- MSGTYPE parameter (TRANS or ROWOP):
  - TRANS message type:
    A publication defined with the TRANS message type contains changes to multiple rows in multiple tables that occurred within a single unit-of-work.
    An example of a TRANS message type XML message with BEFORE_VALUES of YES and TOPIC of DB2INST1/TOPIC is shown in Example A-1. It shows two updates to the CUSTOMER segment in the AUTODB within a single unit-of-work. Multiple <updateRow tags are contained within the <trans tags.

Example: A-1  TRANS message type XML message for event publishing

```xml
<?xml version="1.0" encoding="UTF-8"?>
<msg xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="mqcap.xsd" version="1.0.0" dbName="IMSK" seqNum="IMSIMSK000004600000000_1">
  <trans iisLast="1" segmentNum="1" cmitLSN="IMSIMSK 0000046000000000_1">cmitTime="2005-09-06:05:49:25.000000"
  <updateRow subName="PUBCUSTN" srcOwner="IMSKAUTO" srcName="CUSTOMER">
    <col name="CUSTKEY">
      <char>
        <beforeVal>H000000010</beforeVal>
      </char>
    </col>
  </updateRow>
</trans>
</msg>
```

Note:
In PTF UK09686 released on December 5th 2005, messages may also be published in native EBCDIC format.

Both WebSphere Information Integrator Classic Event Publisher and WebSphere Information Integrator Event Publisher generate these types of messages.
• ROWOP message type:

A publication defined with the ROWOP message type contains changes to a single row in a single table within a single unit-of-work.

An example of a ROWOP message type XML message is shown in Example A-2. It shows a single update to the CUSTOMER segment in the AUTODB within a single unit-of-work. A single <updateRow> tag is contained within the <rowOp> tags.

Example: A-2  ROWOP message type XML message for event publishing

```xml
<psc>
  <Command>publish</Command>
  <Topic>DB2INST1/TESTOPIC</Topic>
</psc>
<msd>
  <msd>None</msd>
</msd>
<xml version="1.0" encoding="UTF-8"?>
  <msg xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="mqcap.xsd" version="1.0.0" dbName="$IMS" seqNum="IMSIMSK 000002F00000000 _1">
```
```
  <updateRow subName="MQLCUST" srcOwner="IMSKAUTO" srcName="CUSTOMER">
  
  <col name="CUSTKEY">
    <char>
      <beforeVal>T000000020</beforeVal>
      <afterVal>T000000020</afterVal>
    </char>
  </col>

  <col name="FIRSTNME">
    <char>
      <beforeVal>MICHAELIM</beforeVal>
      <afterVal>MICHAELIT</afterVal>
    </char>
  </col>

  <col name="LASTNAME">
    <char>
      <beforeVal>THOMPSONS</beforeVal>
      <afterVal>THOMPSONS</afterVal>
    </char>
  </col>

  </updateRow>
</rowOp>

Information messages:
These messages provide information about the status of the Q Capture program or an XML publication.

Note: WebSphere Information Integrator Classic Event Publisher does not generate these types of messages.

Control messages:
These messages ask the Q Capture program to activate or deactivate an XML publication, invalidate a send queue, or confirm that a target table is loaded.

Note: WebSphere Information Integrator Classic Event Publisher does not generate these types of messages.


A.2 Processing XML event messages
As discussed earlier, there are five main steps that an XML message consuming application needs to perform, as follows:
1. Connect to WebSphere MQ.
2. Read the XML message from the queue.
3. Parse the XML in the message to determine the type of database write operation(s) and the corresponding data values.
4. Dispatch the extracted message contents to the appropriate application to deal with the event.
5. Perform application-specific code.
The requirements of XML message consuming applications can vary considerably depending upon the nature and frequency of the events, as follows:

- The arrival rate of event messages might be as little as one per day, or as much as 100 per second.
- A simple XML message consuming application might be written as a single-threaded processing loop that processes each message in turn, while more sophisticated applications might have separate threads to listen to the message queue and perform application processing.
- Some XML message consuming applications might choose to process simple database row operation changes that meet certain criteria, and perform a simple action such as sending an E-mail. Other more complex applications might need to perform a complex analysis of all the changes in a transaction, perform database lookups, and call external functions before choosing to take a particular action.
- Actions that may be taken by XML message consuming applications include one or more of the following possibilities:
  - Send an E-mail or instant/sametime message
  - Invoke a database transaction
  - Call an application
  - Write an audit log to file
  - Invoke a business process comprising multiple steps

Given the broad range of functionality possible with XML message consuming applications, multiple approaches may be used to develop such applications, as described next.

### A.2.1 WebSphere Business Integration Message Broker

WebSphere Business Integration Message Broker is a powerful application development and runtime environment that provides a wide range of tools to make XML message consumption easy. It provides pre-built processing nodes to:

- Listen to a message queue.
- Read arriving messages.
- Parse XML messages into their constituent tagged fields.
- Perform filtering or transformation logic
- Make calls to databases and applications to augment the captured change data with additional information from other databases or applications.
- Drive one or more output methods, covering databases, files, message queues applications, E-mail etc.

For further details on WebSphere Business Integration Message Broker, visit:

A.2.2  WebSphere DataStage

Formerly Ascential DataStage®, this product an advanced ETL\(^1\) tool designed to extract data from various sources, transform the data, and load it into a target database.

WebSphere DataStage is often used to populate data warehouses, which contain data derived from operational data sources and then transformed to load into a data warehouse. In addition to extracting data from data sources such as databases and files, WebSphere DataStage can also exploit real time feeds from event publishing such as those generated by WebSphere Information Integrator event publishing solution to support low latency data warehousing systems.

For further details on WebSphere DataStage, visit:


**Note:** WebSphere Information Integrator Classic Federation enables WebSphere DataStage to access additional data sources such as CA-IDMS™.

A.2.3  Java with the XML Publishing Toolkit

WebSphere MQ supports a wide range of programming languages for accessing its queues. However, Java is a natural choice for consuming XML messages for event publishing because of the availability of the IBM XML Publishing Toolkit, which is a set of java classes to perform many of the basic tasks associated with parsing XML messages, and preparing the data for processing by application-specific code.

IBM developerworks has an article “Event publishing made easy: An application developer's guide on using the XML Publication Toolkit” by Tom Jacopi:


It provides a step by step guide to writing Java programs with the XML Publishing Toolkit.

Refer to Section A.4, “XML Publishing Toolkit” on page 238 for a brief description of the XML Publishing Toolkit.

A.2.4  DB2 Message Queue Listener

DB2 UDB has a Message Queue Listener which can listen on a WebSphere MQ message queue, and invoke a stored procedure whenever a message is received. It can be combined with the following XML capabilities of DB2 to rapidly develop XML message consuming applications:

- DB2 XML Extender provides the ability to process the XML message with XPATH queries.
- DB2 XML Wrapper provides the ability to represent XML strings as SQL Table structures, which can then be queried with normal SQL statements.

By using either the XML Extender, or the XML Wrapper, it is possible to quickly parse XML messages for event publishing, and process the extracted data values within a standard DB2 stored procedure.

For further details on the DB2 Message Queue Listener, visit:


\(^1\) Extract, Transform, and Load
A.3 RFHUTIL tool

It is possible to browse an XML message in a WebSphere MQ queue by using a free download tool for WebSphere MQ from IBM called RFHUTIL. It has a powerful testing facility that allows WebSphere MQ queues to be read and manipulated — it is particularly useful for event publishing testing since it has an easy option to format message content in XML. The RFHUTIL tool can be downloaded from:


Start RFHUTIL to view Figure A-1 that shows the screen for providing details of the queue to be browsed under the Main tab and click Read Q. Figure A-2 shows the contents of an XML message under the Data tab showing changes made to the CUSTOMER segment in the UTODB database.

![Figure A-1 Identify queue of interest](image-url)
The XML Publishing Toolkit is a set of java classes that perform the majority of the basic tasks required to process an XML message for event publishing. It is shipped in the samples directory of DB2 UDB and WebSphere Information Integrator products for Linux, UNIX and Windows.

To install the XML Publishing Toolkit on Windows:

- Change directory to ..\sqlib\samples\repl and unzip asnqwxml.zip:
  - asnqwxml.zip file contains a Web-based event publication sample, as well as the xmlPubTk.zip file, which is the toolkit.
- Create a new directory and unzip xmlPubTk.zip into it. The toolkit comes with the following components:
  - readme.txt file that contains the installation instructions.
  - doc directory containing the Javadoc for the toolkit.
  - src.zip file containing the toolkit source code. The entire toolkit is considered a sample — the source is therefore shipped for your reference and modification.

A set of sample directories. The samples are documented in:
Installation of the XML Publishing Toolkit involves modifying the CLASSPATH environment variable as follows:

- Add xmlPubTk.jar to the CLASSPATH. This is the XML Publishing Toolkit code.
- The XML Publishing Toolkit has to parse XML and therefore needs access to an XML parser which is not shipped with the XML Publishing Toolkit. However, an XML parser is shipped as part of the DB2 Control Center. You can use that parser by adding the following jar files to the CLASSPATH:
  - ../sql\lib\tools\xml-apis.jar
  - ../sql\lib\tools\xercesImpl.jar
- The XML Publishing Toolkit also requires JMS to access WebSphere MQ. When you installed WebSphere MQ, these items should have already been placed into the CLASSPATH. If you get any compile errors about JMS, ensure that the following jar files are in the CLASSPATH:
  - ../WebSphere MQ\Java\lib\com.ibm.mqjms.jar
  - ../WebSphere MQ\Java\lib\jms.jar

The Javadoc documentation can be accessed at doc\overview-tree.html. This includes documentation of all the classes and methods supplied with the XML Publishing toolkit.

Table A-1 lists a subset of the XML Publishing toolkit classes most useful in processing XML data messages.

<table>
<thead>
<tr>
<th>Class name</th>
<th>Class description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column</td>
<td>The Column class represents an actual value of a column in a specific row.</td>
</tr>
<tr>
<td>ColumnSchema</td>
<td>The ColumnSchema class represents a column definition in a table.</td>
</tr>
<tr>
<td>DataMsg</td>
<td>A DataMsg is a XML Publication message about modified data, relating to a transaction that updated, inserted, or deleted rows.</td>
</tr>
<tr>
<td>HeartbeatMsg</td>
<td>The HeartbeatMsg is sent by the Q Capture program to advise that Q Capture is still running.</td>
</tr>
<tr>
<td>PublicationMsgProviderFactory</td>
<td>The PublicationMsgProviderFactory class is used to create a default implementation of a PublicationMsgProvider.</td>
</tr>
<tr>
<td>Row</td>
<td>The Row class represents one row that was published.</td>
</tr>
<tr>
<td>RowOperationMsg</td>
<td>A RowOperationMsg contains one insert, update, or delete operation from the source table.</td>
</tr>
<tr>
<td>TransactionMsg</td>
<td>A TransactionMsg contains one or more insert, update, or delete row operations on the source table.</td>
</tr>
<tr>
<td>Utils</td>
<td>This class contains a bunch of static methods to help format an object as a readable textual string.</td>
</tr>
</tbody>
</table>
REXX based tool to identify log files for the recovery change capture agent (CCA)

In this appendix we describe a REXX based tool that is provided on an “as-is” basis to help you identify log files that need to be provided to the recovery CCA

The topics covered are:
- Problem domain
- REXX based tool overview
- REXX based tool usage
B.1 Problem domain

When an IMS active CCA goes into recovery mode, the event publishing administrator (EPA) needs to restore it to active mode. In most cases, this involves executing the recovery CCA batch job in log file recovery mode. This ensures that none of the changes that occur after an active CCA is set to recovery mode state are lost as long as all the necessary logs are provided to it.

However, the task of identifying all the necessary logs is a manual process and can be error prone when multiple CCAs and multiple log files are involved.

Attention: The REXX based tool is provided on an “as is” basis to help you identify all the logs required for the recovery CCA log file recovery job.

The following sections provide:
- REXX based tool overview
- REXX based tool usage

B.2 REXX based tool overview

We developed a REXX based tool that accepts as input the subsystem ids of all the IMS CCAs, and the restart time associated with each one of them. It then looks up the list of logs in DBRC and identifies the list of logs required as input to the recovery CCA job running log file recovery mode.

Figure B-1 shows the flowchart of the REXX based tool, and Example B-8 shows the source code which has some minimal documentation of its functionality and usage considerations.

A very brief overview of this tool’s input/output, functions and configuration file is described in the following sections.

Note: The REXX based tool source code is available for download from:
ftp://www.redbooks.ibm.com/redbooks/SG247132
Appendix B. REXX based tool to identify log files for the recovery change capture agent (CCA)
B.2.1 Inputs and outputs

The inputs to the REXX tool are as follows:

- Configuration file that lists all the SSIDs for the IMS CCAs with their status and individual restart information (IMS Restart point + Log Sequence Number), as well as the names of the DBRC LIST.LOG ALL output file and the graphic output file.

- Sequential file containing the DBRC LIST.LOG ALL output for all SSIDs.

The REXX program writes the output to standard-out that lists all the logs required by the recovery CCA for the identified CCAs. The standard-out can then be directed to a sequential file using a DD card in the z/OS case, and modifying the script in the Linux/UNIX case.

B.2.2 Functions

The main functions in the REXX based tool are described briefly in Table B-1.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>initialize</td>
<td>Sets constants and defaults for some variables used throughout the program</td>
</tr>
<tr>
<td>configure</td>
<td>Reads the configuration file and sets the variables accordingly</td>
</tr>
<tr>
<td>readloglist</td>
<td>Interprets all the line output in the DBRC LIST.LOG ALL output file, and builds an array of events such as log start and log stop times</td>
</tr>
<tr>
<td>includerestart</td>
<td>Stores the restart points in the array of events</td>
</tr>
<tr>
<td>sortpoints</td>
<td>Sorts the events in the array by time of occurrence</td>
</tr>
<tr>
<td>timeline</td>
<td>Constructs a graphical timeline (ASCII art) of events and writes this timeline to the output file</td>
</tr>
<tr>
<td>printlogdsn</td>
<td>Prints log details such as the index number in the timeline, DSN, start and stop times</td>
</tr>
<tr>
<td>printhowto</td>
<td>Prints the legend of the symbols written to the output file</td>
</tr>
</tbody>
</table>

B.2.3 Configuration file

Example B-1 shows a sample configuration file involving four related CCAs, with a brief explanation of the configuration parameters.

Example: B-1  Sample configuration file

/* Configuration file of REXX tool to identify log files for the recovery CCA

Mark comments by "slash-star comment star-slash"
or with leading pound sign.
Do not leave blank lines outside of a comment.)

For each IMS, include the following information:
SSID=ssid_of_this_ims
RESTART=cca_restart_time_for_this_ims
LOGTYPE=logtype_to_search_for

--------------------------------------------------------------------------------
ATTENTION: LOGTYPE IS NOT USED TODAY -- THE TOOL PROCESSES ALL THE LOG FILES
--------------------------------------------------------------------------------
There can be several LOGTYPE entries.
Give log types as abbreviations:
AP   Archived Primary log
AS   Archived Secondary log
ASP  Archived Primary System log
ASS  Archived Secondary System log
OP   Primary Online log
OS   Secondary Online log

DBRCLIST=file_that_holds_list_output_of_dbrc
NEEDLOG=file_to_write_output_to
COLUMNS=width_of_timeline_output

# /* First SSID (online IMS in this case) */
SSID=IMSK
RESTART=05.256 16:53:23.7
LOGTYPE=AP
OGTYPE=OP
#
# /* Second SSID */
SSID=CACBACH1
RESTART=05.256 17:31:00.0
#
# /* Third SSID */
SSID=CACBACH2
RESTART=05.256 17:30:31.7
#
# /* Fourth SSID */
SSID=CACBACH3
RESTART=05.256 17:31:00.0
#
# /* Fake SSID, used to show all logs after RESTART */
# /* Specify an arbitrary time approx. 30 minutes */
# /* prior to the earliest restart point. */
#SSID=LIST ME
#RESTART=05.240 18:00:00.0
#
# /* Name of the file that holds DBRC output */
DBRCLIST='NEALE.OUTPUT(LIST5)'
DBRCLIST=dbrcoutput
#
# /* Name of the file to write output to */
NEEDLOG=recover_these_logs.list
#
# /* Width of the output (timeline) */
COLUMNS=78

B.3 REXX based tool usage

This section provides detail on how the tool is invoked, and an explanation of a sample
graphic output it produces.

Section 4.4, “Complex recovery mode scenario” on page 215 shows an example of using this
tool.
B.3.1 Invocation

This tool may be invoked on the z/OS platform and Linux/UNIX platforms as follows:

- z/OS platform

  Submitting the RUNZJCL job as shown in Example B-2.

Example: B-2  RUNZJCL job

```plaintext
//IILOGS JOB 'MANUEL,ITSO,WSIICEP',
//         NOTIFY=&SYSUID,
//         CLASS=A,MSGCLASS=H,MSGLEVEL=(1,1)
//* This JCL invokes the REXX script:
//* "Helper Script to recover Event Publisher Agents"
//* Finds all necessary IMS logs to recover a given
//* Event Publisher. It was developed by
//* manuel.mueller@de.ibm.com as part of the RedBook
//* "Publishing IMS and DB2 Data using WebSphere
//* Information Integrator: Configuration and
//* Monitoring Guide".
//*
//* TSOBATCH EXEC PGM=IRXJCL,
//           PARM='LISTLOG mainframe stdin'
//           program platform confmode
//           -mainframe -STDIN
//           -PC       -FILE
//           System print
//SYSPRINT DD SYSOUT=*  
//SYSEXEC DD DSN=MANUEL.REXX.EXEC,DISP=SHR REXX program library
//INFILE DD DSN=MANUEL.REXX.EXEC(DBRCLIST),DISP=SHR
//OUTFILE DD DSN=MANUEL.REXX.OUT,DISP=(MOD,CATLG,CATLG)
//SYSTSPRT DD SYSOUT=*  
//SYSTSERR DD SYSOUT=*  
//SYSTSIN DD *

#* Give log types as abbreviations:
# AP   Archived Primary log
# AS   Archived Secondary log
# ASP  Archived Primary System log
# ASS  Archived Secondary System log
# OP   Primary Online log
# OS   Secondary Online log
#
# For Mainframes, give DDname to write Errors to.
STDERR=SYSTSERR
#
#* First SSID (online IMS in this case) */
SSID=IMSK
RESTART=05.256 16:53:23.7
LOGTYPE=AP
LOGTYPE=OP
```

246  Publishing IMS and DB2 Data using WebSphere Information Integrator: Configuration and Monitoring Guide
# Second SSID
SSID=CACBACH1
RESTART=05.256 17:31:00.0
#
# Second SSID
SSID=CACBACH2
RESTART=05.256 17:30:31.7
#
# Second SSID
SSID=CACBACH3
RESTART=05.256 17:31:00.0
#
# Fake SSID, used to show all logs after RESTART
# Specify an arbitrary time approx. 30 minutes
# prior to the earliest restart point.
SSID=LIST ME
RESTART=05.240 18:00:00.0
#
#
#
# Name of the file that holds DBRC output
DBRCLIST='MANUEL.REXX.EXEC(DBRCLIST)'
DBRCLIST=INFILE
#
# Name of the file or DD statement to write output to
NEEDLOG=OUTFILE
#
# Width of the output (timeline)
COLUMNS=78

/*

Linux/UNIX platform

Execute the runlinux.sh shell script as shown in Example B-3.

Example: B-3  runlinux.sh script

REXPGM=listlog.cmd
CONFIG=listlog.conf
OUTDIR=output
OUTFIL=output.log
ERRFIL=error.log
LOGLST=recover_these_logs.list

# Prepare output directory
mkdir $OUTDIR
rm $OUTDIR/$LOGLST
rm $OUTDIR/$OUTFIL
rm $OUTDIR/$ERRFIL

# Run REXX program
cat $CONFIG | |
    regina $REXPGM PC STDOUT | |
    2> $OUTDIR/$ERRFIL | |
    1> $OUTDIR/$OUTFIL

# Display output
The configuration file shown in Example B-1 on page 244 corresponds to the event publishing scenario shown in Figure B-2 and is the basis of the inputs and outputs described here. Figure B-2 shows four related CCAs; one IMS online region (IMS810K) and three batch jobs (CACBACH1, CACBACH2 and CACBACH3) all updating an IMS database in serial fashion. The partial output of the DBRC LIST.LOG ALL output that is provided as input to the REXX based tool is shown in Example B-4 — the entire output can be downloaded from:

ftp://www.redbooks.ibm.com/redbooks/SG247132
EARLIEST CHECKPOINT = 05.207 09:15:11.4

DSN=IMS810K.RLDSP.IMSK.D05207.T0915080.V00       UNIT=3390
START = 05.207 09:15:08.0   FIRST DS LSN= 0000000000000001
STOP  = 05.207 09:21:53.1    LAST DS LSN= 0000000000000096
FILE SEQ=0001    #VOLUMES=0001

VOLSER=TOTMIN STOPTIME = 05.207 09:21:53.1
CKPTCT=2    CHKPT ID = 05.207 09:21:52.9
LOCK SEQUENCE#= 000000000000

LOGALL
START = 05.207 09:15:08.0    *
EARLIEST ALLOC TIME = 00.000 00:00:00.0
DBDS ALLOC=0

105.256 20:20:32.6       LISTING OF RECON      PAGE 0003
0--------------------------------------------------------------------------
PRISLD                  RECORD SIZE= 304
START = 05.207 09:15:08.0    *
STOP  = 05.207 09:21:53.1    #DSN=1
GSGNAME=**NULL**
FIRST RECORD ID= 0000000000000001 PRILOG TOKEN= 0

DSN=IMS810K.RLDSP.IMSK.D05207.T0915080.V00       UNIT=3390
START = 05.207 09:15:08.0   FIRST DS LSN= 0000000000000001
STOP  = 05.207 09:21:53.1    LAST DS LSN= 0000000000000096
FILE SEQ=0001    #VOLUMES=0001
CHECKPOINT TYPES=88: SIMPLE=Y SNAPQ=N DUMPQ=N PURGE=N FREEZE=Y

VOLSER=TOTMIN STOPTIME = 05.207 09:21:53.1
CKPTCT=2    CHKPT ID = 05.207 09:21:52.9
LOCK SEQUENCE#= 000000000000

105.256 20:20:32.6       LISTING OF RECON      PAGE 0004
0--------------------------------------------------------------------------
PRISLD                  RECORD SIZE= 304
START = 05.207 09:23:46.5    *
STOP  = 05.207 18:11:06.5    #DSN=1
GSGNAME=**NULL**
FIRST RECORD ID= 0000000000000001 PRILOG TOKEN= 0
EARLIEST CHECKPOINT = 05.207 09:23:48.8

DSN=IMS810K.RLDSP.IMSK.D05207.T0923465.V01       UNIT=3390
START = 05.207 09:23:46.5   FIRST DS LSN= 0000000000000001
STOP  = 05.207 18:11:06.5    LAST DS LSN= 00000000000000A2
FILE SEQ=0001    #VOLUMES=0001

VOLSER=TOTMIN STOPTIME = 05.207 18:11:06.5
CKPTCT=2    CHKPT ID = 05.207 18:11:06.4
LOCK SEQUENCE#= 000000000000

LOGALL
START = 05.207 09:23:46.5    *
EARLIEST ALLOC TIME = 00.000 00:00:00.0
DBDS ALLOC=0

105.256 20:20:32.6       LISTING OF RECON      PAGE 0005
0--------------------------------------------------------------------------
PRISLD                  RECORD SIZE= 304
START = 05.207 09:23:46.5    *
STOP  = 05.207 18:11:06.5    #DSN=1
GSGNAME=**NULL**
FIRST RECORD ID= 0000000000000001       PRILOG TOKEN= 0

DSN=IMS810K.SLDSP.IMSK.D05207.T0923465.V01       UNIT=3390
START = 05.207 09:23:46.5       FIRST DS LSN= 00000000000000001
STOP  = 05.207 18:11:06.5       LAST DS LSN= 00000000000000000A2
FILE SEQ=0001       #VOLUMES=0001
CHECKPOINT TYPES=88: SIMPLE=Y SNAPQ=N DUMPQ=N PURGE=N FREEZE=Y

VOLSER=TOTMIN STOPTIME = 05.207 18:11:06.5
CKPTCT=2       CHKPT ID = 05.207 18:11:06.4
LOCK SEQUENCE#= 0000000000000

105.256 20:20:32.6           LISTING OF RECON           PAGE 0006

---------------------------------------------------------------------
PRILOG                                      RECORD SIZE=     304
START = 05.209 00:07:21.3       *       SSID=IMSK     VERSION=8.1
STOP  = 05.209 00:15:28.0       #DSN=1
GSGNAME=**NULL**
FIRST RECORD ID= 00000000000000A3       PRILOG TOKEN= 0
EARLIEST CHECKPOINT = 05.207 09:23:48.8

DSN=IMS810K.RLDSP.IMSK.D05209.T0007213.V00       UNIT=3390
START = 05.209 00:07:21.3       FIRST DS LSN= 0000000000000000A3
STOP  = 05.209 00:15:28.0       LAST DS LSN= 0000000000000131
FILE SEQ=0001       #VOLUMES=0001

VOLSER=TOTMIN STOPTIME = 05.209 00:15:28.0
CKPTCT=2       CHKPT ID = 05.209 00:15:27.8
LOCK SEQUENCE#= 0000000000000

LOGALL
START = 05.209 00:07:21.3       *
EARLIEST ALLOC TIME = 00.000 00:00:00.0
DBDS ALLOC=0

105.256 20:20:32.6           LISTING OF RECON           PAGE 0007

---------------------------------------------------------------------
PRISLD                                      RECORD SIZE=     304
START = 05.209 00:07:21.3       *       SSID=IMSK     VERSION=8.1
STOP  = 05.209 00:15:28.0       #DSN=1
GSGNAME=**NULL**
FIRST RECORD ID= 00000000000000A3       PRILOG TOKEN= 0

DSN=IMS810K.SLDSP.IMSK.D05209.T0007213.V00       UNIT=3390
START = 05.209 00:07:21.3       FIRST DS LSN= 0000000000000000A3
STOP  = 05.209 00:15:28.0       LAST DS LSN= 0000000000000131
FILE SEQ=0001       #VOLUMES=0001
CHECKPOINT TYPES=88: SIMPLE=Y SNAPQ=N DUMPQ=N PURGE=N FREEZE=Y

VOLSER=TOTMIN STOPTIME = 05.209 00:15:28.0
CKPTCT=2       CHKPT ID = 05.209 00:15:27.8
LOCK SEQUENCE#= 0000000000000

105.256 20:20:32.6           LISTING OF RECON           PAGE 0008

---------------------------------------------------------------------
PRILOG                                      RECORD SIZE=     304
START = 05.209 00:16:41.5       *       SSID=IMSK     VERSION=8.1
STOP  = 05.209 00:40:52.3       #DSN=1
GSGNAME=**NULL**
FIRST RECORD ID= 0000000000000132       PRILOG TOKEN= 0
EARLIEST CHECKPOINT = 05.207 09:23:48.8

DSN=IMS810K.RLDSP.IMSK.D05209.T0016415.V00       UNIT=3390
START = 05.209 00:16:41.5             FIRST DS LSN= 0000000000000132
STOP  = 05.209 00:40:52.3             LAST DS LSN= 00000000000001CE
FILE SEQ=0001  #VOLUMES=0001

VOLSER=TOTMIN STOP TIME = 05.209 00:40:52.3
CKPTCT=2    CHKPT ID = 05.209 00:40:52.0
LOCK SEQUENCE#= 000000000000

LOGALL
START  = 05.209 00:16:41.5             *
EARLIEST ALLOC TIME = 00.000 00:00:00.0
DBDS ALLOC=0

105.256 20:20:32.6 LISTING OF RECON PAGE 0009
0-----------------------------------------------------------------------------
PRISLD           RECORD SIZE=    304
START = 05.209 00:16:41.5              *   SSID=IMSK    VERSION=8.1
STOP  = 05.209 00:40:52.3              #DSN=1
GSGNAME=**NULL**
FIRST RECORD ID= 0000000000000132     PRILOG TOKEN= 0

DSN=IMS810K.SLDSP.IMSK.005209.T0016415.V00 UNIT=3390
START = 05.209 00:16:41.5             FIRST DS LSN= 0000000000000132
STOP  = 05.209 00:40:52.3             LAST DS LSN= 00000000000001CE
FILE SEQ=0001  #VOLUMES=0001
CHECKPOINT TYPES=88: SIMPLE=Y SNAPQ=N DUMPQ=N PURGE=N FREEZE=Y

VOLSER=TOTMIN STOP TIME = 05.209 00:40:52.3
CKPTCT=2    CHKPT ID = 05.209 00:40:52.0
LOCK SEQUENCE#= 000000000000

105.256 20:20:32.6 LISTING OF RECON PAGE 0010
0-----------------------------------------------------------------------------
PRILOG           RECORD SIZE=    304
START = 05.209 09:00:35.6              *   SSID=IMSK    VERSION=8.1
STOP  = 05.209 10:42:30.3              #DSN=1
GSGNAME=**NULL**
FIRST RECORD ID= 0000000000000001     PRILOG TOKEN= 0
EARLIEST CHECKPOINT = 05.209 09:00:38.1

DSN=IMS810K.RLDSP.IMSK.005209.T0900356.V02 UNIT=3390
START = 05.209 09:00:35.6             FIRST DS LSN= 0000000000000001
STOP  = 05.209 10:42:30.3             LAST DS LSN= 0000000000000170
FILE SEQ=0001  #VOLUMES=0001

VOLSER=TOTMIN STOP TIME = 05.209 10:42:30.3
CKPTCT=2    CHKPT ID = 05.209 10:42:30.2
LOCK SEQUENCE#= 000000000000

LOGALL
START  = 05.209 09:00:35.6             *
EARLIEST ALLOC TIME = 09.000 09:00:00.0
DBDS ALLOC=1                        -DBD-    -DDN-    -ALLOC-
DI21PART DI21PARO  1

105.256 20:20:32.6 LISTING OF RECON PAGE 0011
0-----------------------------------------------------------------------------
PRISLD           RECORD SIZE=    304
START = 05.209 09:00:35.6              *   SSID=IMSK    VERSION=8.1
STOP  = 05.209 10:42:30.3              #DSN=1
GSGNAME=**NULL**
FIRST RECORD ID= 0000000000000001     PRILOG TOKEN= 0
<table>
<thead>
<tr>
<th>Listing of Recon</th>
<th>Page 0012</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSN=IMS810K.SLDSP.IMSK.D05209.T0900356.V02</td>
<td>UNIT=3390</td>
</tr>
<tr>
<td><strong>START</strong> = 05.209 09:00:35.6</td>
<td>FIRST DS LSN= 0000000000000001</td>
</tr>
<tr>
<td><strong>STOP</strong> = 05.209 10:42:30.3</td>
<td>LAST DS LSN= 00000000000000170</td>
</tr>
<tr>
<td>FILE SEQ=0001</td>
<td>#VOLUMES=0001</td>
</tr>
<tr>
<td>CHECKPOINT TYPES=88: SIMPLE=Y SNAPQ=N DUMPQ=N PURGE=N FREEZE=Y</td>
<td></td>
</tr>
<tr>
<td>VOLSER=TOTMIN STOPTIME = 05.209 10:42:30.3</td>
<td></td>
</tr>
<tr>
<td>CKPTCT=2</td>
<td>CHKPT ID = 05.209 10:42:30.2</td>
</tr>
<tr>
<td>LOCK SEQUENCE# = 000000000000</td>
<td></td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>Listing of Recon</th>
<th>Page 0013</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSN=IMS810K.SLDSP.IMSK.D05209.T1054557.V01</td>
<td>UNIT=3390</td>
</tr>
<tr>
<td><strong>START</strong> = 05.209 10:54:55.7</td>
<td>* SSID=IMSK VERSION=8.1</td>
</tr>
<tr>
<td><strong>STOP</strong> = 05.209 18:59:03.8</td>
<td>#DSN=1</td>
</tr>
<tr>
<td>GSGNAME=<strong>NULL</strong></td>
<td></td>
</tr>
<tr>
<td>FIRST RECORD ID= 0000000000000171</td>
<td>PRILOG TOKEN= 0</td>
</tr>
<tr>
<td>EARLIEST CHECKPOINT = 05.209 09:00:38.1</td>
<td></td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>Listing of Recon</th>
<th>Page 0014</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSN=IMS810K.SLDSP.IMSK.D05209.T1901032.V01</td>
<td>UNIT=3390</td>
</tr>
<tr>
<td><strong>START</strong> = 05.209 19:01:03.2</td>
<td>* SSID=IMSK VERSION=8.1</td>
</tr>
<tr>
<td><strong>STOP</strong> = 05.209 19:01:03.2</td>
<td>#DSN=1</td>
</tr>
<tr>
<td>GSGNAME=<strong>NULL</strong></td>
<td></td>
</tr>
<tr>
<td>FIRST RECORD ID= 00000000000003A9</td>
<td>PRILOG TOKEN= 0</td>
</tr>
<tr>
<td>EARLIEST CHECKPOINT = 05.209 09:00:38.1</td>
<td></td>
</tr>
</tbody>
</table>

---

252 Publishing IMS and DB2 Data using WebSphere Information Integrator: Configuration and Monitoring Guide
B.3.2 Sample output explanation

Example B-5 shows a sample graphic output of the REXX based tool, while Example B-7 shows the output log of the invocation of this tool.

Example: B-5  Sample graphic output of the REXX based tool

### Print 'Graphic' Log Timeline ###

IMSK     AP : <=1================>
IMSK     ASP: <=2================>
IMSK      OP : <=3================X=15======>
IMSK      * :   !~~~~~~~~~~~~~~~~~~~~~~~~~~~~
CACBACH1 AP :     <=5==>
CACBACH1  * :                  !~~~~~~~~~~~~~
CACBACH2 AP :            <=7=>
CACBACH2  * :              !~~~~~~~~~~~~~~~~~
CACBACH3  * :                  !~~~~~~~~~~~~~
CACBACH3 AP :                      <=16==>

+ - + - + - + - + - + - + - + - + - + - + - + - + - + - + - + - + - + - + - + - + - 

List of Logs and DSNs:
Symbols:

- `<` Start of log dataset.
- `=` Log dataset is open.
- `>` Stop of log dataset.
- `X` Stop and start of log dataset (log switch).
- `!` Restart point of this SSID is here.
- `~` Restart point of this SSID is before this.
- `?` Error while constructing graphical timeline.

Log Types:

- **AP** Archived Primary log
- **AS** Archived Secondary log
- **ASP** Archived Primary System log
- **ASS** Archived Secondary System log
- **OP** Primary Online log
- **OS** Secondary Online log

The graphic output shown in Example B-5 is divided into two main parts:

- Part one is a graphical timeline that displays the start and endpoints of relevant logs.

  It helps you understand the log files that were created by the related CCAs in the event publishing environment, and the sequence in which they were created. It is possible for the logs of the related CCAs to have time overlaps.

  Included in this graphical representation is the restart point which makes it much easier to identify the logs that need to be provided as input to the recovery CCA job. The graphical timeline includes all logs that were finished after the earliest of all given restart points, that is, all logs that were either running at that point or were started after the earliest restart point.

**Note:** The timeline is not proportional. The timeline gives the exact sequence of events, but it does not quantify the time that passed between them. For example, Example B-5 shows that log 5 was started when log 3 was already in use, but it does not tell you how much time passed between the start of log 3 and the start of log 5.

Example B-5 shows different sections for each of the four SSIDs (IMSK, CACBACH1, CACBACH2 and CACBACH3) that were specified in the configuration file Example B-1 on page 244 — these sections appear in Example B-5 in the same order as specified in the configuration file.

Each section is dedicated to one SSID. It contains one row for each log file type. Every row lists the individual log files that may have to be provided to the recovery CCA job. If the configuration file provides a restart point for this SSID, it will be shown on an additional line. The graphical output begins with the first log that was running or started after the restart time specified in the configuration file.
If there are more logs than can fit on one line, one or more new output segments are started to continue the timeline for all SSID sections.

- Part two holds detailed information about the log datasets.
  This includes the data set names (DSN), type of data set, and start times of the logs. This is implemented for easier readability. All logs in the graphical timeline of part one are labelled with a number, which is listed in this part.

The configuration file in Example B-1 on page 244 has the following settings for SSID IMSK:

- SSID=IMSK
- RESTART=05.256 16:53:23.7
- LOGTYPE=AP
- LOGTYPE=OP

**Note:** As mentioned earlier, the LOGTYPE parameter is ignored.

Example B-5 on page 253 indicates the following:

1. IMSK was running all the time, and there was always at least one log.
2. The restart point of IMSK occurred while it was logging to log 3\(^1\).
3. CACBACH1 created log 5 while IMSK was logging to log 3.
4. CACBACH2 created log 7 after CACBACH1 finished log 5.
5. The restart point of CACBACH1 occurred while it was logging to log 7.
6. Then CACBACH2 finished log 7.
7. The restart points of both CACBACH1 and CACBACH3 occurred at the same time.
8. IMSK switched from log 3 to log 15.
9. CACBACH3 created log 16 (its restart point had already occurred) and finished the log.
10. IMSK finished log 15.
11. Log 15 was not archived since there is no AP entry for this time span.

The graphical output in Example B-3 therefore contains all log files that IMSK used at 16:53:23.7 on day 05.256 and created after that time, because they contain entries that were logged after the restart point.

The graphical timeline for the IMSK SSID is shown in Example B-6. The “!” in the last line reflects the restart point indicating the point at which IMSK CCA entered recovery mode.

**Example: B-6   IMSK SSID graphic timeline**

```
IMSK     AP : <=1================>
IMSK     ASP: <=2================>
IMSK     OP : <=3================X=15======>
IMSK " !~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

---

\(^1\) Logs 1 and 2 are archive copies of log 3. Therefore, they were not created at the same time as log 3. However, they cover the same time span and contain the same data.
All logs that appear partially or completely at the same time as a "~" must be considered for the recovery process. In Example B-6, these are logs 1, 2, 3 and 15. These logs can contain IMS changes that were not processed by the event publisher because the CCA entered recovery mode. Log files 1, 2 and 3 are identical and contain the same IMS changes, and therefore only one of them must be provided as input to the recovery CCA 'L' mode (log file recovery). We chose log file number 3 (IMS810K.OLP00) because online logs are usually more easily accessed than archived logs. The 'X' corresponds to the end of log 3 and the beginning of the next log (number 15) IMS810K.OLP01 at the same point in time — IMSK performed a log switch.

Based on the output shown in Example B-6, the following log data sets are required for recovery of the IMSK SSID:

- Log data set number 3 IMS810K.OLP00
- Log data set number 15 IMS810K.OLP01

Using the same approach as used with SSID IMSK, the following applies to CACBACH1, CACBACH2 and CACBACH3:

- For SSID CACBACH1, the restart point appears after the end of the log file NALUR.LOG1. This means that log number 5 NALUR.LOG1 should not be included in the recovery CCA job.
- For SSID CACBACH2 entered recovery mode while log file 7 NALUR.LOG2 was being written to. Therefore, log file 7 NALUR.LOG2 must be provided as input to the recovery CCA job.
- For SSID CACBACH3, the restart time appears before log file 16 NALUR.LOG3 is created. This agent was manually put into recovery mode before it was started indicating that none of the changes in this log were processed. Therefore, log file 16 NALUR.LOG3 must be provided as input to the recovery CCA job.

Example: B-7  REXX based tool output log

```plaintext
Configuration is read from stdin
Platform (for input and output) is PC
SSID 0: IMSK
Restart Time for IMSK: 05.256 16:53:23.7
SSID 1: CACBACH1
Restart Time for CACBACH1: 05.256 17:31:00.0
SSID 2: CACBACH2
Restart Time for CACBACH2: 05.256 17:30:31.7
SSID 3: CACBACH3
Restart Time for CACBACH3: 05.256 17:31:00.0
DBRC output in: 'NEALE.OUTPUT(LIST5)'
Output timeline and list of logs to: recover_these_logs.list
Output width (columns): 78
Not a monitored SSID
Not a monitored SSID
Not a monitored SSID
Not a monitored SSID
Not a monitored SSID
Not a monitored SSID
Not a monitored SSID
Not a monitored SSID
Not a monitored SSID
Not a monitored SSID
Not a monitored SSID
Not a monitored SSID
Not a monitored SSID
Not a monitored SSID
Not a monitored SSID
Not a monitored SSID
Not a monitored SSID
Not a monitored SSID
Not a monitored SSID
Not a monitored SSID
Not a monitored SSID
Not a monitored SSID
Not a monitored SSID
Not a monitored SSID
Not a monitored SSID
Not a monitored SSID
Not a monitored SSID
```
Not a monitored SSID
Not a monitored SSID
Not a monitored SSID
Not a monitored SSID
Not a monitored SSID
Not a monitored SSID
Not a monitored SSID
Not a monitored SSID
Not a monitored SSID
Not a monitored SSID
Not a monitored SSID
Not a monitored SSID
Not a monitored SSID
Not a monitored SSID
Not a monitored SSID
Not a monitored SSID
This log is after the restart time:
  Restart: 05.256 16:53:23.7
  Type: AP
  SSID: IMSK -
  Stop: 00.000 00:00:00.0
  DSN: IMS810K.RLDSP.IMSK.D05256.T1627259.V18
  Stop: 05.256 18:11:42.5
This log is after the restart time:
  Restart: 05.256 16:53:23.7
  Type: ASP
  SSID: IMSK -
  Stop: 00.000 00:00:00.0
  DSN: IMS810K.SLDSP.IMSK.D05256.T1627259.V18
  Stop: 05.256 18:11:42.5
This log is after the restart time:
  Restart: 05.256 17:31:00.0
  Type: AP
  SSID: CACBACH1-
  Stop: 05.256 17:27:16.0
  DSN: NALUR.LOG1
  Stop: 05.256 17:27:16.0
This log is after the restart time:
  Restart: 05.256 17:30:31.7
  Type: AP
  SSID: CACBACH2-
  Stop: 05.256 17:30:32.1
  DSN: NALUR.LOG2
  Stop: 05.256 17:30:32.1
This log is after the restart time:
  Restart: 05.256 17:31:00.0
  Type: AP
  SSID: CACBACH3-
  Stop: 05.256 20:10:34.5
  DSN: NALUR.LOG3
  Stop: 05.256 20:10:34.5
Primary Online Log found
OLD SSID: IMSK -
  DNSSTOP: STOP = 05.252 17:24:11.3 LAST DS LSN= 000000000000385D
  DNSSTOP: STOP = 05.255 20:00:48.7 LAST DS LSN= 000000000000437D
  DNSSTOP: STOP = 05.255 20:03:54.5 LAST DS LSN= 0000000000004CB3
Example: B-8  REXX tool source code

/* REXX */
/* ++++ REXX ++++ Version 2005-12-16 09:45 ++++ */
version=26

/* Helper Script to recover Event Publisher Agents */
* Finds all necessary IMS logs to recover a given
* Event Publisher. It was developed by
* manuel.mueller@de.ibm.com as part of the RedBook
* "Publishing IMS and DB2 Data using WebSphere
* Information Integrator: Configuration and
* Monitoring Guide".
* Call:
*       program platform confmode
*       file23 mainframe STDIN
* Needs output of:
*   - Active Agent Status Job
*   - DBRC LIST.LOG ALL
* Wants Config as STDIN
* If you want to list logs before the earliest restart
* point of any of the given agents, give an additional
* agent (whose name does not occur in DBRC output) and
* the time when you want the list to begin. E.g.
*   SSID=LIST ME
*   RESTART=00.001 18:00:00.0

Assumes time stamps to be unique. E.g. if Log Start
and Restart Point are equal up to a tenth of a
second, they will appear in the same column in the
graphical output.
*/

/* Change Log V 26, begin work at 2005-12-16 09:45:
* Try different approaches to correct mainframe input and output, as suggested by Henry Kiesslich and Denis Gäbler
* Different output of printdsn details: and olddsn details: to distinguish them in the output.
* Correct "default" earliest restart point. Formerly was "zz.zzz zz:zz:zz.z" but in EBCDIC numbers have a higher HEX value than letters
* Do a STRIP(value,t) to strip trailing spaces for all config variables as the EXECIO seems to always give the full line length.
* Introduce default DBRC-list name "DBRCLIST", either a file of that name or a DD statement that allocates a data set/member with the output of DBRC LIIST.LOG ALL
* Todo:
  * Config parameter to set platform.
  * Test Mainframe input/output.
  * Adhere to REXX convention of using stem.0 as the counter of this stem's records.
  * Adhere to config parameter: type of log (Log types regarded for this SSID.) When reading DBRC output, filter logs accordingly.
  * Make sure that start and stop in same row in timeline come from same DBRC entry. pointid. Running number (when reading DBRC) pointlinkid. "This start is linked to this stop." Sort pointid only, use referenced values.
  * Use JavaDoc (or similar) in rewriting comments to make them understandable for novices.
  * Make error: write to STDERR Observe maxcolumns when writing error messages.
  * Ongoing logs should be the last point in timeline. (Now the ongoing restart precedes by one column.) Possibly use a special character to mark this.
  * Accept different time stamp formats for Restart points (config) Log start and stop (DBRC output) Adapt code where time stamps are parsed (configure, DBRC) compared (after:)
* special times (all occurrences of timeongoing
* and timelast)
*
* Do not split DSN numbers on timeline 'line' break.
*
* Handle empty lines in config.
* (Now, an empty lines marks the end of input.)
*
* Possibly read more than one input line at a time.
 */

PARSE ARG platform confmode
    /* confmode: STDIN (default), FILE */
    /* platform: mainframe (default), PC */

/* SAY "Command arguments say:",
  " Platform="platform,  
  " Config="confmode*/

/*****************  Main Program work flow  ******************/

/*CALL openfiles*/    /* Open input/output data sets */
CALL initialize      /* Set defaults and constants. */
CALL configure       /* Read configuration. */
CALL readloglist     /* Parse DBRC output. */
CALL includerestart  /* Include restart points in array of events. */
CALL sortpoints      /* Sort events by time. */
CALL timeline        /* Construct graphical timeline. */
CALL printlogdsn     /* Print log datasets. */
CALL printhowto      /* Print key to the symbols. */
/*CALL closefiles*/     /* Close input/output data sets */

SAY "Program completed."
EXIT 0        /* Exit program normally. */
initialize:

/** Set constants and some variable defaults used throughout this program. */
/** Constants: */

initialize:

fontSize = "00.000 00:00:00.0"
/* Use highest possible HEX value to ensure that any restart point that the configuration gives will be the new earliest restart point.
Format: "zz.zzz zz:zz:zz.z" */
timelast = '\"ffff\"x.\"fffffff\"x "\"fffffff\"x: "\"fffffff\"x. "\"fffffff\"x"x

/* Original character set: */
charblank = ''
charlogstart = 'A'
charopenlog = '='
charlogstop = 'O'
charlogswitch = 'S'
charrestart = 'R'
charrestartopen = '~'
charerror = 'E'

/* Alternate character set (more graphical): */
charblank = ''
charlogstart = '<'
charopenlog = '='
charlogstop = '>
charlogswitch = 'X'
charrestart = '!' charrestartopen = '~'
charerror = '?'
timelineheaderlength = 14
/* Length of string in front of actual timeline.
Usually (V 18): "SSID---- typ:-" */

/* Variables: */

ssididx = 0 /* Number of tracked subsystems */
ssid. = '' /* Array to hold tracked subsystems */
restart. = '' /* Array to hold restart point of Sbstms */
logssid = '' /* SSID of current set of DSNs */
currentssididx = 0 /* Array counter of SSID set (config) */
earliestrestart = timelast /* Earliest restart point of all agents. */
configure: /** Read configuration file and set some variables accordingly. */
/* ??? ToDo: what if confmode <> stdin */
SAY "Begin configuration."

DO UNTIL TYPE = ''
/* Read the next line of config until line is returned. */
configline = getconfigline()
/* SAY "Config:" configline (Debug) */

PARSE VAR configline type '=' value
    /* Remove trailing blanks (especially on mainframe) */
value = STRIP(value, t)
/* SAY "T:" type "V:" value */

SELECT
    WHEN TYPE='STDERR' THEN DO
        errorout = value
        SAY "STDERR DD:" errorout
    END
    WHEN TYPE='SSID' THEN DO
        ssid.ssididx = value
        SAY "SSID ssididx":" ssid.ssididx
        ssididx = ssididx + 1
WHEN type='RESTART'            THEN DO
    i = ssididx - 1
    restart.i = value
    SAY "Restart Time for SSID" ssid.i":",
        restart.i "(earliestrestart)"
    check = after(earliestrestart, restart.i)
    IF check = 1 THEN
        /* This is yet the earliest restart point. */
        earliestrestart = restart.i
    END
WHEN type='DBRCLIST'           THEN DO
    dbrclist = value
    SAY "DBRC output in:" dbrclist
END
WHEN type='NEEDLOG'            THEN DO
    needlog  = value
    SAY "Output timeline and list of logs to:" needlog
END
WHEN type='COLUMNS'            THEN DO
    maxcolumns  = value
    SAY "Output width (columns):" maxcolumns
END
OTHERWISE
    NOP
END

IF DBRCLIST = '' THEN DO
    SAY 'No DBRC LIST given.'
    EXIT 12
END

SAY "Configuration completed."
return

openfiles:    PROCEDURE EXPOSE platform dbrclist needlog errorout
    /** Open data sets for input/output. */

    SELECT
        WHEN platform = "PC" THEN DO
            /* PC does not need this. */
        END

        WHEN platform = "mainframe" THEN DO
            /* DBRC output as input */
            "EXECIO 0 DISKR" dbrclist "(OPEN"
            SAY "Exec returned" RC
            /* Timeline as output */
            "EXECIO 0 DISKW" needlog "(OPEN"
            SAY "Exec returned" RC
            /* Error as output */
            "EXECIO 0 DISKW" errorout "(OPEN"
            SAY "Exec returned" RC
        END

        OTHERWISE DO
SAY "Write: Unknown platform" platform,
   "(only PC or mainframe)."
RETURN 2
END
END
return 0

closefiles: PROCEDURE EXPOSE platform dbrclist needlog errorout
/** Open data sets for input/output. */
SELECT
  WHEN platform = "PC" THEN DO
   /* PC does not need this. */
  END
WHEN platform = "mainframe" THEN DO
   /* DBRC output as input */
   "EXECIO 0 DISKR" dbrclist "(FINIS"
   /* Timeline as output */
   "EXECIO 0 DISKW" needlog "(FINIS"
   /* Error as output */
   "EXECIO 0 DISKW" errorout "(FINIS"
  END
OTHERWISE DO
  SAY "Write: Unknown platform" platform,
      "(only PC or mainframe)."
  RETURN 2
END
END
return 0

printhowto: /* Print a legend of symbols to output file */
SAY "Print howto."
line = ""
IF writelogline(needlog, line)<0 THEN CALL error "writing log list"

line = " Symbols:"
IF writelogline(needlog, line)<0 THEN CALL error "writing log list"
line = ""
IF writelogline(needlog, line)<0 THEN CALL error "writing log list"
line = " charlogstart " Start of log dataset."
IF writelogline(needlog, line)<0 THEN CALL error "writing log list"
line = " charopenlog " Log dataset is open."
IF writelogline(needlog, line)<0 THEN CALL error "writing log list"
line = " charlogstop " Stop of log dataset."
IF writelogline(needlog, line)<0 THEN CALL error "writing log list"
line = " charlogswitch " Stop and Start of log dataset (log switch)."
IF writelogline(needlog, line)<0 THEN CALL error "writing log list"
line = " charrestart " Restart point of this SSID is here."
IF writelogline(needlog, line)<0 THEN CALL error "writing log list"
line = " charrestartopen " Restart point of this SSID is before this."
IF writelogline(needlog, line)<0 THEN CALL error "writing log list"
line = charerror " Error while constructing graphical timeline."
IF writelogline(needlog, line)<>0 THEN CALL error "writing log list"
line = ""
IF writelogline(needlog, line)<>0 THEN CALL error "writing log list"
line = " Log Types:"
IF writelogline(needlog, line)<>0 THEN CALL error "writing log list"
line = ""
IF writelogline(needlog, line)<>0 THEN CALL error "writing log list"
line = "AP  Archived Primary log"
IF writelogline(needlog, line)<>0 THEN CALL error "writing log list"
line = "AS  Archived Secondary log"
IF writelogline(needlog, line)<>0 THEN CALL error "writing log list"
line = "ASP Archived Primary System log"
IF writelogline(needlog, line)<>0 THEN CALL error "writing log list"
line = "ASS Archived Secondary System log"
IF writelogline(needlog, line)<>0 THEN CALL error "writing log list"
line = "OP  Primary Online log"
IF writelogline(needlog, line)<>0 THEN CALL error "writing log list"
line = "OS  Secondary Online log"
IF writelogline(needlog, line)<>0 THEN CALL error "writing log list"

SAY "Print howto completed."
RETURN

/****************  Read DBRC output  *******************/

readloglist: /** Interpret all lines in DBRC output. */
status        = 'findlog'

SAY "Parsing DBRC output..."
DO WHILE getdbrcline() = 0
  /* Read the next line of DBRC output. */
  SELECT
    WHEN status = 'findlog' THEN CALL findlog
    WHEN status = 'finddsn' THEN CALL finddsn
    WHEN status = 'findolddsn' THEN CALL findolddsn
    OTHERWISE CALL error "readloglist: Status error."
END

END /* WHILE LINES() */

RETURN 0

findlog: /** Find the beginning of a Log entry in DBRC output. */
IF LEFT(WORD(line, 1), 6) = "0-----" THEN
    /* Marker for new log found. */
    CALL discernlogs
RETURN status

discernlogs: /** Process log details only if it is of a monitored type. */
    IF getdbrcline() < 0 THEN
        CALL error "discernlogs: Unexpected end of DBRC file."
    END
    SELECT
        WHEN WORD(line, 1) = "PRILOG" THEN DO
            /* SAY "Primary Log (archived) found" */
            logtype = "AP"
            CALL prilogdetails
        END
        WHEN WORD(line, 1) = "SECLOG" THEN DO
            /* SAY "Secondary Log (archived) found: not needed." */
            logtype = "AS"
            RETURN
        END
        WHEN WORD(line, 1) = "PRISLD" THEN DO
            /* SAY "Primary System Log (archived) found." */
            logtype = "ASP"
        END
        WHEN WORD(line, 1) = "SECSLD" THEN DO
            /* SAY "Secondary System Log (archived) found: not needed" */
            logtype = "ASS"
            RETURN
        END
        WHEN WORD(line, 1) = "PRIOLD" THEN DO
            SAY "Primary Online Log found"
            logtype = "OP"
            CALL priolddetails
        END
        WHEN WORD(line, 1) = "SECOLD" THEN DO
            /* SAY "Primary Online Log found: not needed" */
            logtype = "OS"
            RETURN
        END
        OTHERWISE DO
            /* Not a log entry we can/want to handle */
            logtype = "* unknown *"
            status = 'findlog'
            RETURN
        END
    END
RETURN

prilogdetails: /** Primary Log found, extract details about it. */
    IF getdbrcline() < 0 THEN
        SAY "Unexpected end of DBRC file."
        logssid = SUBSTR(WORD(line, 6), 6, 8)
        /* dummy = WORD(line, 6)
        PARSE VAR dummy dummy '=' logssid */
        IF inssids() > 0 THEN DO
            logstart = WORD(line, 3) WORD(line, 4)
            /* SAY "Log start:" logstart */
IF getdbrcline() <> 0 THEN
    CALL error "prilogdetails:",
    "Unexpected end of DBRC file."
logstop = WORD(line, 3) WORD(line, 4)
status = 'finddsn'
END
ELSE DO
    /* SAY "Not a monitored SSID:" logssid */
    status = 'findlog'
END
RETURN

finddsn:
/** Cause loop (through readloglist) until either the next DSN
or new log marker is found. */
IF LEFT(WORD(line, 1), 3) = "DSN" THEN
    CALL printdsndetails
    IF LEFT(WORD(line, 1), 6) = "0-----" THEN
        CALL discernlogs
    return
printdsndetails:
/** Extract details of DSN (start, stop, etc.) from DBRC output. */
parse var line with typedsn '=' logdsn typeunit '=' unit
/* SAY "-   Unit:" unit "  DSN:" logdsn */
/* SAY "-   " WORD(line,1) */
IF getdbrcline() <> 0 THEN
    CALL error "printdsndetails: Unexpected end of DBRC file."
dsnstart = WORD(line, 3) WORD(line, 4)
/* SAY "-   DSN start:" dsnstart */
ELSE DO
    SAY "Unexpected end of DBRC file."
    dsnstop = WORD(line, 3) WORD(line, 4)
    /* SAY "-   DSN stop:" dsnstop */
ENDIF
IF after(dsnstop, earliestrestart) >= 0 THEN DO
    SAY "This log is after the restart time:
    SAY "  Restart:" restart.currentssididx
    SAY "  Type:" logtype
    SAY "  SSID:" logssid"-
    SAY "  Stop:" logstop
    SAY "  DSN: " logdsn
    SAY "  Stop:" dsnstop
    CALL storedsn
END
RETURN

priolddetails: /* Primary Online Log found, extract details about it. */
IF getdbrcline() <> 0 THEN
    SAY "Unexpected end of DBRC file."
logssid = SUBSTR(WORD(line, 1),6, 8)
SAY "OLD SSID:" logssid"-
IF inssids() > 0 THEN DO
/* SAY "This SSID is monitored." logstart */
status = 'findolddsn'
END
ELSE DO
/* SAY "Not a monitored SSID:" logssid */
status = 'findlog'
END
RETURN

findolddsn:
/* Cause loop (through readloglist) until either the next DSN
or new log marker is found. */
/* Needed in addition to finddsn: because Online Logs are
listed in DBRC output in a different format. */
IF LEFT(WORD(line, 2), 3) = "DSN" THEN
CALL olddsndetails
IF WORD(line, 1) = "SECOLD" THEN DO
/* SAY "Beginning of list of Secondary Online Log",
"Data Sets - not needed." */
status = 'findlog'
RETURN
END
IF LEFT(WORD(line, 1), 6) = "0-----" THEN
CALL discernlogs
RETURN

olddsndetails:
/* Extract details of DSN (start, stop, etc.) from DBRC output. */
/* Needed in addition to printdsndetails: because Online Logs are
listed in DBRC output in a different format. */
PARSE VAR line WITH typeddname '=' ddname typedsn '=' logdsn
/* SAY "-   DDName:" ddname "  DSN:" logdsn */
IF getdbrcline() <> 0 THEN
CALL error "olddsndetails: Unexpected end of DBRC file."
dsnstart = WORD(line, 3) WORD(line, 4)
/* SAY "- DSN start:" dsnstart */
IF getdbrcline() <> 0 THEN
CALL error "olddsndetails: Unexpected end of DBRC file."
SAY "DNSSTOP:" line
dsnstop = WORD(line, 3) WORD(line, 4)
/* SAY "- DSN stop: " dsnstop */
IF after(dsnstop, earliestrestart) >= 0 THEN DO
SAY "This online log is after the restart time:"
SAY " Type:" logtype
SAY " SSID:" logssid"
SAY " Stop:" logstop
SAY " DSN:" logdsn
SAY " Stop:" dsnstop
END
RETURN
storedsn:
/** Put information about one log DSN, extracted
 from DBRC output, to array of events. */

/* First event, start of log: */
pointtime.pointidx = dsnstart
pointssid.pointidx = logssid
pointlogtype.pointidx = logtype
pointdsn.pointidx = logdsn
pointoper.pointidx = 'A'
pointidx = pointidx + 1

/* Second event, stop of log: */
pointtime.pointidx = dsnstop
pointssid.pointidx = logssid
pointlogtype.pointidx = logtype
pointdsn.pointidx = logdsn
pointoper.pointidx = 'O'
pointidx = pointidx + 1
RETURN

printlogdsn:
/** Print log details (index number in timeline,
 DSN, start, stop, etc.) to output file. */
SAY "Print log DSNames."

/* Output header, including format information. */
line = "List of Logs and DSNs:" IF writelogline(needlog, line)<>0 THEN CALL error "writing log list"

line = "#### SSID     Typ  start" /* "             stop" */ IF writelogline(needlog, line) <> 0 THEN CALL error "writing log list"

line = "DSN" IF writelogline(needlog, line) <> 0 THEN CALL error "writing log list"
line = "" IF writelogline(needlog, line) <> 0 THEN CALL error "writing log list"

/* Output all start events. (Which should cover all logs.) */
DO count = 1 TO pointidx
   IF pointoper.count = 'A' THEN DO
      line = rightalign(count,4) pointssid.count,
            pointlogtype.count pointtime.count
      IF writelogline(needlog, line) <> 0 THEN CALL error "writing log list"
      line = "pointdsn.count"
      IF writelogline(needlog, line) <> 0 THEN CALL error "writing log list"
   END
END
SAY "Print DSNames completed."
RETURN

/****************  Checks & Comparisons  ******************/

rightalign: PROCEDURE
/** Right align string to minlength if possible,
  but do not truncate it. */
ARG string, minlength
/* Return too long string rather than truncate it. */
actuallength = LENGTH(string)
IF actuallength > minlength THEN
  RETURN string
/* Insert ' ' to string after position 0.
  "Truncate" ' ' to min - actual
  (and padd with second ' '). */
string = INSERT(' ', string, 0, minlength - actuallength, ' ')
RETURN string

inssids: PROCEDURE EXPOSE ssid. ssididx logssid currentssididx
  currentssididx = 0
  DO UNTIL currentssididx = ssididx
    IF logssid = ssid.currentssididx THEN DO
      /* SAY "SSID matches" */
      RETURN currentssididx + 1
    END
    currentssididx = currentssididx + 1
  END
  currentssididx = currentssididx + 1
RETURN 0

after: PROCEDURE EXPOSE timeongoing
/** determine whether time1 occurs after time2. */
/* Compare two times
  * Return -1 if time2 is after    time1
  * Return  0 if time1 is equal to time2
  * Return  1 if time1 is after    time2
  * Return  1 if time1 is 0 (log still online)
    and time 2 is not 0 (not equal)
  */
ARG time1, time2
/* Is 'log_stop' after 'restart'? */
IF time1 = time2 THEN DO /* Points in time are equal. */
  RETURN  0 /* Even if both are null/ongoing. */
END
IF time1 = timeongoing THEN DO /* Point1 is still ongoing. */
  /* But Point2 isn't. */
  /* SAY "Time 1 is ongoing:" time1 */
  RETURN  1
END
IF time2 = timeongoing THEN DO /* Point2 is still ongoing. */
  /* But Point1 isn't. */
  /* SAY "Time 2 is ongoing:" time1 */
  RETURN -1
Appendix B. REXX based tool to identify log files for the recovery change capture agent (CCA)

PARSE VAR time1 year1 '.' day1 ' ',
    hour1 ':' minute1 ':' second1 '.' csecond1
PARSE VAR time2 year2 '.' day2 ' ',
    hour2 ':' minute2 ':' second2 '.' csecond2

IF year1 > year2 THEN RETURN 1
IF year1 < year2 THEN RETURN -1
IF day1 > day2 THEN RETURN 1
IF day1 < day2 THEN RETURN -1
IF hour1 > hour2 THEN RETURN 1
IF hour1 < hour2 THEN RETURN -1
IF minute1 > minute2 THEN RETURN 1
IF minute1 < minute2 THEN RETURN -1
IF second1 > second2 THEN RETURN 1
IF second1 < second2 THEN RETURN -1
IF csecond1 > csecond2 THEN RETURN 1
IF csecond1 < csecond2 THEN RETURN -1

CALL error "Function 'after()' reached a point it should not reach."
RETURN 2

/********************  Restart Points  ********************/
includerestart:
/** Store restart points to array of events. */

    count = 0
    DO WHILE count < ssididx
        pointtime.pointidx = restart.count
        pointssid.pointidx = LEFT(ssid.count, 8)
        pointlogtype.pointidx = ' * '
        pointdsn.pointidx = 'Restart Point'
        pointoper.pointidx = 'R'
        pointidx = pointidx + 1
        count = count +1
    END
RETURN

/********************  Sort the Bubbles  ********************/
sortpoints: PROCEDURE EXPOSE pointtime. pointssid. pointlogtype.,
              pointdsn. pointoper. pointidx timeongoing
/** Make sure all events are sorted by time of occurrence. */

points = pointidx - 1

/* ???ToDo: use optimized sorting algorithm. */
sorttime = bubblesort()

/* Print list of sorted Points: (Debug) */
/*
SAY "Sorted Points:"
DO point=1 TO points
SAY pointtime.point pointssid.point pointlogtype.point,
    pointoper.point pointdsn.point
END
SAY "Sorted in" sorttime "seconds"
*/
RETURN

bubblesort:
/** Bubble sort all events by time. */
start=TIME("R")         /* Start the timer */
DO i = points TO 1 BY -1 UNTIL sorted = 1
    sorted = 1          /* Assume the items are sorted */
    DO j=2 TO i
        m = j - 1
        IF isbigger(m,j) = 1 THEN DO
            /* If the items are out of order, swap them */
            dummy = swappoints(m,j)
            sorted = 0  /* Not yet sorted. */
        end
    end
END
stop = time("R") /* Stop the timer */
runtime = stop - start
RETURN runtime

swappoints:
/** Swap the position of two given events within the array. */
ARG one, two
/*SAY "Time before" pointtime.one pointtime.two temp.1 */

temp.1 = pointtime.one
temp.2 = pointssid.one
temp.3 = pointlogtype.one
temp.4 = pointdsn.one
temp.5 = pointoper.one

pointtime.one           = pointtime.two
pointssid.one           = pointssid.two
pointlogtype.one        = pointlogtype.two
pointdsn.one            = pointdsn.two
pointoper.one           = pointoper.two

pointtime.two           = temp.1
pointssid.two           = temp.2
pointlogtype.two        = temp.3
pointdsn.two            = temp.4
pointoper.two = temp.5
RETURN 0

isbigger:
ARG one, two
/** Determine whether one occurs after two
 or, if both are the same (logswitch),
 whether one is stop (earlier) and one
 is start (later).
 * Return 0 if point1 occurs before point2
 * Return 1 if point1 occurs after point2
 * Return 2 on ERROR
 /**
 compare = after(pointtime.one, pointtime.two)

 SELECT
 WHEN compare = 1 THEN /* one is later than two, return "yes" */
 RETURN 1
 WHEN compare = -1 THEN /* two is later than one, return "no" */
 RETURN 0
 WHEN compare = 0 THEN DO
 /* Points in time are equal. Bring "stop" operation first. */
 IF pointoper.one = 'O' THEN
 IF pointoper.two = 'A' THEN DO
 /* stop shall occur before
 (is not bigger than) start:
 return "no" */
 RETURN 0
 END
 IF pointoper.one = 'A' THEN
 IF pointoper.two = 'O' THEN DO
 /* start shall occur after
 (is bigger than) stop:
 return "yes" */
 RETURN 1
 END
 END
 OTHERWISE DO
 CALL error "Function 'isbigger()' for equal times",
 "reached a point it should not reach."
 RETURN 2
 END
 END
 /* SAY "Points are equal and A/O cannot be used to decide." */
 RETURN 0

/******************** Timeline ********************/

timeline:
/* Construct a graphical timeline of events */
/* PROCEDURE EXPOSE pointtime. pointssid. pointdsn. 
  pointoper. pointidx logssid ssididx */
SAY "Construct timeline."

step   = 1
row.   = ''
rowc   = 0
column = 1
rowssid. = '        ' /* SSID of this column */
rowlog. = '   '        /* Column log type */
rowop.  = ''           /* Last operation in this row */
rowcolop. = ''           /* Operation in this row at current column */
rowahead. = 0            /* Number of columns written in one row */
outputrows = assignrows()

/*SAY "Timeline" pointidx ssididx rowassignedidx ssid.1 (Debug)*/
line = "  #### Print 'Graphic' Log Timeline (V "version") ####  "
IF writelogline(needlog,line)<0 THEN CALL error "writing log list"
line = ""
IF writelogline(needlog,line)<0 THEN CALL error "writing log list"

DO WHILE step < pointidx /* For all events. (pointidx - 1) */
  logssid = pointssid.step
  logtype = pointlogtype.step
  rowc = rowassigned.logssid.logtype
  /* SAY "Step" step " Row" rowc logssid logtype,
  " Column" column," Oper" pointoper.step,
  " Type" pointtime.step (Debug) */
  CALL printopcode

  nextstep = step + 1
  IF pointtime.step <> pointtime.nextstep THEN DO
    /* SAY "Next point in Time, next Column." */
    dummy = advancecolumn()
    dummy = advancecolumn()
  END
  step = step + 1
  /* if step = 100 then exit 42 (Debug) */
END

dummy = printtimeline()
SAY "Timeline completed."
RETURN

goahead:
/** Advance lagging rows to make all rows end in the current 
  column. */
/* ???ToDo: only advance to the column needed for the 
  new event. */
max = 0
rowcount = 1
DO WHILE rowcount <= outputrows
  /* SAY "Row" rowc ahead" rowahead.rowcount " max" max */
  IF rowahead.rowcount > max THEN
max = rowahead.rowcount
rowcount = rowcount + 1
END

/* SAY "Advancing" max "columns." */
DO max
    dummy = advancecolumn()
END
rowahead. = 0 /* Reset ahead counters. */
/* Insert additional column in output for better viewing. */
dummy = advancecolumn()
RETURN

printopcode:
    /** Insert operation marker of given event into
    output row. */
    /**
    * A     stArt of log
    * O     stOp of log
    * S     log Switch
    * R     Restart point
    * E     Error
    **/
    oldopcode  = rowop.rowc         /* code of last operation */
    thisopcode = pointoper.step     /* code to print in this column */
    nextopcode = pointoper.step     /* code to use next time
    (Print "X" for logswitch but
    treat it as a log start.) */
    /* SAY "OldOpCode:" oldopcode,
    "ThisOpCode:" thisopcode,
    "NextOpCode:" nextopcode,
    "Op of this row in this column:" rowcolop.rowc
    (Debug) */
    IF rowcolop.rowc <> '' THEN DO
    /* The last operation was not a stOp. */
    /* SAY "OpCodes in row" rowc":" oldopcode,
    thisopcode nextopcode (Debug) */
    SELECT
        WHEN oldopcode = '' THEN NOP
        WHEN oldopcode = 'A' THEN DO /* StArt of Log */
            CALL error "Unexpected operation at",
            "Log Start:" pointoper.step,
            "on same point in time as stArt of log."
            thisopcode = 'E'
            nextopcode = oldopcode
        END
        WHEN oldopcode = 'O' THEN /* StOp of Log */ DO
            /*say thisopcode nextopcode SUBSTR(row.rowc, column, 1)*/
            IF thisopcode = 'A' THEN DO
                /* Log started at same point in time.
                Assume Log Switch. */
                thisopcode = 'S'
                nextopcode = 'A'
            END
            ELSE DO

Appendix B. REXX based tool to identify log files for the recovery change capture agent (CCA)  275
CALL error "Unexpected operation at Log Stop: ",
    pointoper.step,
    "on same point in time as st0p of log."
thisopcode = 'E'
nextopcode = oldopcode

WHEN oldopcode = 'R' THEN /* Restart Point */
    CALL error "Unexpected operation at Restart Point: ",
        pointoper.step,
        "on same point in time as Restart."
    thisopcode = 'E'
    nextopcode = thisopcode
    RETURN
END

ELSE IF rowahead.rowc > 0 THEN /* There was already output written to this
    row in the current column. */
    CALL goahead
SELECT
    WHEN thisopcode = 'A' THEN
        thischar = charlogstart
    WHEN thisopcode = 'O' THEN
        thischar = charlogstop
    WHEN thisopcode = 'S' THEN
        thischar = charlogswitch
    WHEN thisopcode = 'E' THEN
        thischar = charerror
    WHEN thisopcode = 'R' THEN
        thischar = charrestart
    OTHERWISE DO
        CALL error "PrintOpeCode found an unknown code."
        thischar = '?'
    END
END

IF nextopcode = 'A' THEN DO
    thischar = thischar || charopenlog || step || charopenlog
    /* SAY "OpCode: -"thischar"-" (Debug) */
END

rowahead.rowc = rowahead.rowc + length(thischar)
row.rowc = OVERLAY(thischar, row.rowc, column, length(thischar))
rowop.rowc = nextopcode
rowcolop.rowc = rowop.rowc

/* SAY "New row" rowc":" row.rowc (Debug) */
RETURN
assignrows: PROCEDURE EXPOSE ssid, ssididx,
    pointidx pointssid, pointlogtype,
    rowassigned rowassignedidx rowssid, rowlogtype.
/** Store for each SSID + LogType pair the row it
  will appear in output "graphic", group them
  by SSID.
  Return total number of rows */

ssidcount = 0
rowassignedidx = 0
rowassigned = 0
rowssid = ''
rowlogtype = ''

DO WHILE ssidcount < ssididx
    /* SAY "Assign rows for SSID:" ssidcount,
       ssid.ssidcount (Debug) */
    pointcount = 0
    DO WHILE pointcount <= pointidx
        logssid = pointssid.pointcount
        logtype = pointlogtype.pointcount
        IF logssid = ssid.ssidcount THEN DO
            IF rowassigned.logssid.logtype = 0 THEN DO
                rowassignedidx = rowassignedidx + 1
                rowassigned.logssid.logtype = rowassignedidx
                rowssid.rowassignedidx = logssid
                rowlogtype.rowassignedidx = logtype
                /*SAY "  "logtype "" rowassigned.logssid.logtype*/
            END
        END
        pointcount = pointcount + 1
    END
    ssidcount = ssidcount + 1
END
return rowassignedidx

advancecolumn:
/** Advance output rows by one column.
  Print openlog, restartopen, or blank. */
IF column >= maxcolumns - timelineheaderlength THEN
    /* Output width reached */
    dummy = printtimeline()

    count1 = 1
    DO WHILE count1 <= rowassignedidx
        IF SUBSTR(row.count1, column, 1) = '' THEN DO
            SELECT
                WHEN rowop.count1 = "A" THEN
                    row.count1 =,
                    OVERLAY(charopenlog, row.count1, column)
WHEN rowop.count1 = "R" THEN
   row.count1 = OVERLAY(charrestartopen,,
     row.count1, column)
  OTHERWISE
   row.count1 =,
   OVERLAY(charblank, row.count1, column)
END

IF rowahead.count1 > 0 THEN DO
   rowahead.count1 = rowahead.count1 - 1
   /* SAY "Ahead of row" count1,
     "reduced by advancecolumn to",
     rowahead.count1 (Debug) */
END

/* Delete "This column operation" for new column */
rowcolop.count1 = ''
/* SAY "row#, step#, row:" column count1 row.count1
 (Debug) */
count1 = count1 + 1
END

column = column + 1
return 0

printtimeline:
/** Print the next part of the timeline
 that fits the output file's width
 to the output file." */

count = 1
currentssid = rowssid.count

DO WHILE count <= rowassignedidx
   IF currentssid <> rowssid.count THEN DO
      /* New SSID, insert blank line */
      line = ""
      IF writelogline(needlog,line)<>0 THEN
         CALL error "writing log list"
      currentssid = rowssid.count
   END
   line = rowssid.count rowlogtype.count":",
      LEFT(row.count, maxcolumns - timelineheaderlength - 1)
   IF writelogline(needlog,line) <> 0 THEN
      CALL error "writing log list"
   row.count = SUBSTR(row.count,,
      maxcolumns - timelineheaderlength)
   count = count + 1;
   /* Print blank lines and devider. */
   line = ""
   IF writelogline(needlog,line)<>0 THEN CALL error "writing log list"
   IF writelogline(needlog,line)<>0 THEN CALL error "writing log list"
   DO count = 1 TO maxcolumns - 4 BY 4 /* Construct devider. */
      line = line || ' + -'
   END

IF writelogline(needlog,line)<0 THEN CALL error "writing log list"
line = ""
IF writelogline(needlog,line)<0 THEN CALL error "writing log list"
IF writelogline(needlog,line)<0 THEN CALL error "writing log list"
column = 1
RETURN 0

/****************  Input / Output  ******************/
error: /** Output an error message. */
PARSE ARG comment
/* Print an error to output file. */

/* ???ToDo: observe maxcolumns */
/* ???ToDo: Write to STDERR, not file "STDERR:" */
line = "ERROR:" comment
SELECT
  WHEN platform = "PC" THEN DO
    SAY line
    call lineOut 'STDERR:', ,
      "StandardError " || line
  END
  WHEN platform = "mainframe" THEN DO
    OUTLINE.1 = line
    "EXECIO 1 DISKW" errorout "(STEM OUTLINE.
  END
  OTHERWISE DO
    SAY "Write: Unknown platform" platform,
      "(only PC or mainframe)."
  RETURN 2
END
END

IF writelogline(needlog, line)<0 THEN DO
  IF platform = "mainframe" THEN
    SAY "Error while writing ERROR to "errorout "."
  ELSE
    SAY "Error while writing ERROR to STDERR."
  SAY line
RETURN

getconfigline: PROCEDURE EXPOSE confmode platform
/** Read the next line in config file/input. */
/* Ignore comment lines */
goodline = 0
incomment = 0
conflcnt = 0
DO WHILE goodline = 0
  PARSE PULL configline
  /* ???ToDo: prepare for confmode=FILE */
  beginline.1 = SUBSTR(WORD(configline, 1), 1, 1)
  beginline.2 = SUBSTR(WORD(configline, 1), 1, 2)
  endline.2   = RIGHT(configline, 2)
  SELECT
    WHEN incomment > 0 THEN
      IF endline.2 = '*/' THEN
        incomment = incomment - 1
      WHEN beginline.2 = '/*' THEN
        IF endline.2 <> '*/' THEN
          incomment = incomment + 1
        WHEN beginline.2 = '//' THEN
          NOP
        WHEN beginline.1 = '# ' THEN
          NOP
        WHEN beginline.1 = '% ' THEN
          NOP
        OTHERWISE
          goodline = 1
          /* SAY "Config:" configline */
    END
  END
RETURN configline

getdbrcline:        PROCEDURE EXPOSE dbrclist line platform
  /** Read the next line in DBRC LIST.LOG file.
      Returns data in the variable "line". */
  code = getfileline(dbrclist)
  IF SUBSTR(line,38,38) = "LISTING OF RECON                  PAGE",
  THEN DO
    /* Found one of DBRC's "page markers",
      do not return it - read a new line. */
    code = getfileline(dbrclist)
    IF SUBSTR(line, 2, 1) <> '-' THEN DO
      /* This line is not the beginning of
        a lift of a new SSID:
        suppress the first character
        (number) after the "page marker". */
      line = OVERLAY(' ', line, 1)
    END
  END
RETURN code

getfileline:         PROCEDURE EXPOSE platform line
  PARSE ARG filename
  /** Read one line from given file or dataset */
  SELECT
    WHEN platform = "PC" THEN DO
      /* Any more lines? */
      IF LINES(filename, 'N') > 0 THEN DO

/* ???.ToDo: use INTERPRET to hide from mainframe. */
/* Read the next line */
line = LINEIN(filename, , 1)
END
ELSE
  RETURN 1
END

WHEN platform = "mainframe" THEN DO
  "EXECIO 1 DISKR" filename "(STEM inline."
  /* Check EXECIO return code to see if
  end of file was reached. */
  IF RC = 2 THEN
    RETURN 1
  line = inline.1
END

OTHERWISE DO
  SAY "Read: Unknown platform" platform,
  "(only PC or mainframe)."
  RETURN 2
END
RETURN 0

writelogline: PROCEDURE EXPOSE platform
/** Write one line to given file or dataset */
/* TRACE ?r (Debug) */
PARSE ARG filename, line
/* SAY "To File name" filename "write this line:" line (Debug) */

SELECT
  WHEN platform = "PC" THEN DO
    CALL LINEOUT filename, line
  END
  WHEN platform = "mainframe" THEN DO
    OUTLINE.1 = line
    "EXECIO 1 DISKW" filename "(STEM OUTLINE."
    /* SAY "ExecIO returned" RC (Debug) */
  END
  OTHERWISE DO
    SAY "Write: Unknown platform" platform,
    "(only PC or mainframe)."
    RETURN 2
  END
END
RETURN 0
Additional material

This redbook refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this redbook is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

ftp://www.redbooks.ibm.com/redbooks/SG247132

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select the Additional materials and open the directory that corresponds with the redbook form number, SG247132.

Using the Web material

The additional Web material that accompanies this redbook includes the following files:

<table>
<thead>
<tr>
<th>File name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complexscenario</td>
<td>This is a Microsoft PowerPoint® presentation that should be viewed as a screen show. It shows a time sequence of events corresponding to the complex scenario described in Chapter 4, “Recovery mode scenarios” on page 195 of the redbook.</td>
</tr>
<tr>
<td>disclaimer.txt</td>
<td>This is a disclaimer about the code being provided on an “as-is” basis, with no implied warranties, etc.</td>
</tr>
<tr>
<td>Manifest.txt</td>
<td>This is a text file describing the contents of all the files available.</td>
</tr>
</tbody>
</table>
REXXTOOL.ZIP  This is a zip file containing the code, input, and output for the REXX based tool described in Appendix B, “REXX based tool to identify log files for the recovery change capture agent (CCA)” on page 241. The code includes an example of execution on the z/OS and Linux platforms.

Simplescenario  This is a Microsoft PowerPoint presentation that should be viewed as a screen show. It shows a time sequence of events corresponding to the simple scenario described in Chapter 4, “Recovery mode scenarios” on page 195 of the redbook.

Trustworthy.zip  This is a zip file containing code to define the Trustworthy Insurance CIF ODS database, and the MQ Listener and Java code to update the CIF ODS and process the EAS messages as described in Chapter 3, “Trustworthy Insurance Customer Information Facility (CIF) and Event Alert System scenario” on page 85.

System requirements for downloading the Web material

The following system configuration is recommended:

- **Hard disk space:** 350KB
- **Operating System:** Windows

How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip file into this folder.
Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 286. Note that some of the documents referenced here may be available in softcopy only.

- *WebSphere Information Integrator Q Replication: Fast Track Implementation Scenarios*, SG24-6487
- *MQSeries Primer*, REDP-0021-00

Other publications

These publications are also relevant as further information sources:

- *IBM DB2 Information Integrator Release Notes for Classic Event Publisher Version 8.2*
- *IBM DB2 Information Integrator, Replication and Event Publishing Guide and Reference Version 8.2*, SC18-7568-00
- *IBM DB2 Information Integrator, ASNCLP Program Reference for Replication and Event Publishing Version8.2*, SC18-9410-00
- *WebSphere MQ for AIX, Quick Beginnings Version5.3*, GC34-6079-01
- *WebSphere MQ, System Administration Guide*, SC34-6069-01
- *WebSphere MQ, Script (MQSC) Command Reference*, SC34-6055-01
- *WebSphere MQ, Security Version5.3*, SC34-6079-01
- *WebSphere MQ, Intercommunication*, SC34-6059-01
Online resources

These Web sites and URLs are also relevant as further information sources:

- DB2 UDB Version 8 manuals site:

- WebSphere MQ sites:
  http://www.ibm.com/software/integration/mqfamily/library/manualsa/csqqak05/csqqak05e.htm

- WebSphere Information Integration home page
  http://www.ibm.com/software/data/integration/

- Q Replication V8.2 Information Roadmap

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads
ibm.com/support

IBM Global Services
ibm.com/services
Index

A
active CCA 17–18, 143, 197, 200, 242
active libraries 17, 93, 139
active mode 6, 31, 196, 202, 242
administration queue 82, 151
amqsget 105, 107
amqsput 105
auto insurance 86–87, 197, 216

C
CA-IDMS 2, 4, 236
Change Capture Agent 70–71, 241
change capture agent 70
channels 34, 83, 97–98
check agent 46–47, 203–204
CIF xix, 20, 85–86
CIF ODS database 156, 158
CIF solution 114–115
CIF Updater 89, 91
commit interval 77, 82
COMMIT_INTERVAL 82
Complex recovery mode scenario 195–196
collection data set 147
configuration file 30, 35, 199, 211, 242, 244
configuration parameters 6, 22, 244
collection file 57, 61
collection tables 76, 83, 149–150
Copybooks 8, 10, 93, 108
copybooks 8, 109, 121
correlation service 5–6, 93, 145, 197–198
CSA 8, 23, 147, 201, 203
customized control file 64, 67

data set allocation 19, 143
database changes 2–3, 75–76, 87, 200
DBDs 8–9, 249
DBRC 59, 61, 197, 213, 242, 244
DDLT0 198
DFSFLGX0 17–18, 143
duplicated buffers 40, 54
dynamic queues 83

E
EBCDIC 3, 7, 232
end-to-end latency 82
Event Alert System 85–86
Event notification programs 89, 92, 97
event notification programs 89, 91

event publisher xix, 1–2, 73, 75, 85, 87, 200, 203, 232,
234, 256
event publisher administrator 211
event publishing xix, 2, 4, 74–75, 87, 89, 196–197,
231–232, 242, 248
EXIT keyword 16–17, 139

F
FTP 10, 112, 114
full log file recovery 64

G
GDG data sets 8, 20, 93, 145

H
home insurance 86–87

I
IBMQREP.SPILL.MODELQ 83
IMS CCA 38, 46
IMS log record suffix 38, 50
IMS Logger exit 17–18, 201
IMS Online Change 52–53
IMS processing flow 34, 36
IMS segment 7, 108–109
Incremental log file recovery 64–65
incremental log file recovery 46, 64
in-doubt resolution 6, 8
informational messages 7, 32, 76, 82

L
listener 7, 74, 89, 95, 236
LOB 77–78
local administration queue 83
log buffer 5, 33
log buffers 5, 35
log file recovery 46–47, 207–208, 242, 256
log files 5, 7, 213, 241–242
Log Record Suffix 38, 50
log sequence number 50, 59, 164, 172, 203, 227, 244
LSN 59–60, 203, 206, 249

M
maximum connections 25–26
Maximum tasks 24, 26
MESSAGE POOL SIZE 28, 42
Message Queue Listener 7, 89, 236
Metadata catalog 4, 6, 139, 143
metadata catalog 4–5, 93, 108
metadata utility 6, 8
Minimum tasks 24, 26
<table>
<thead>
<tr>
<th>Term</th>
<th>Page Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>model queue definition</td>
<td>83</td>
</tr>
<tr>
<td>monitor interval</td>
<td>82</td>
</tr>
<tr>
<td>MQ listener</td>
<td>159, 186</td>
</tr>
<tr>
<td>O</td>
<td></td>
</tr>
<tr>
<td>ODS</td>
<td>87, 89</td>
</tr>
<tr>
<td>overrun</td>
<td>53</td>
</tr>
<tr>
<td>overruns</td>
<td>63, 69</td>
</tr>
<tr>
<td>P</td>
<td></td>
</tr>
<tr>
<td>persistent messages</td>
<td>106</td>
</tr>
<tr>
<td>Primary Log File</td>
<td>62</td>
</tr>
<tr>
<td>Processing flow</td>
<td>xix, 1, 34, 73, 80</td>
</tr>
<tr>
<td>processing flow</td>
<td>xix, 1, 4, 73, 75</td>
</tr>
<tr>
<td>publication definition</td>
<td>23, 197</td>
</tr>
<tr>
<td>publication service</td>
<td>6–7, 198</td>
</tr>
<tr>
<td>Publishing queue maps</td>
<td>78, 83</td>
</tr>
<tr>
<td>publishing queue maps</td>
<td>76–77, 83, 149</td>
</tr>
<tr>
<td>Q</td>
<td></td>
</tr>
<tr>
<td>Q Apply</td>
<td>77, 79, 97–98</td>
</tr>
<tr>
<td>Q Capture</td>
<td>76–77, 94, 97–98, 234, 239</td>
</tr>
<tr>
<td>Queue manager</td>
<td>27, 82–83, 97–98</td>
</tr>
<tr>
<td>queue manager</td>
<td>27, 82–83, 91, 95</td>
</tr>
<tr>
<td>quiesce</td>
<td>23, 45, 214, 229</td>
</tr>
<tr>
<td>quiescing</td>
<td>214–215</td>
</tr>
<tr>
<td>R</td>
<td></td>
</tr>
<tr>
<td>receive queue</td>
<td>82, 99</td>
</tr>
<tr>
<td>receive queues</td>
<td>89, 95</td>
</tr>
<tr>
<td>Recovery CCA</td>
<td>35, 44, 203–204, 242, 244</td>
</tr>
<tr>
<td>recovery CCA</td>
<td>xix, 35, 44, 196, 202, 241–242</td>
</tr>
<tr>
<td>recovery data sets</td>
<td>57, 67, 197, 215</td>
</tr>
<tr>
<td>recovery dataset</td>
<td>18, 47, 200, 204</td>
</tr>
<tr>
<td>Recovery mode</td>
<td>5, 18, 195–196, 242, 256</td>
</tr>
<tr>
<td>recovery mode</td>
<td>xix, 1, 4, 195–196, 242, 255</td>
</tr>
<tr>
<td>Recovery mode resolution</td>
<td>195–196</td>
</tr>
<tr>
<td>Rejection</td>
<td>41, 45</td>
</tr>
<tr>
<td>rejection</td>
<td>35, 41</td>
</tr>
<tr>
<td>related CCAs</td>
<td>196–197, 244, 248</td>
</tr>
<tr>
<td>remote queue</td>
<td>82, 97–98</td>
</tr>
<tr>
<td>replication</td>
<td>xix–xx, 2, 7, 75–76, 89, 148, 234</td>
</tr>
<tr>
<td>Replication Alert Monitor</td>
<td>84</td>
</tr>
<tr>
<td>Replication Center</td>
<td>80, 84, 149–150</td>
</tr>
<tr>
<td>replication queue maps</td>
<td>83</td>
</tr>
<tr>
<td>repository</td>
<td>9, 14, 110–111</td>
</tr>
<tr>
<td>Restart information</td>
<td>20, 38, 82, 97, 202–203</td>
</tr>
<tr>
<td>restart information</td>
<td>6, 8, 82, 97, 200–201, 244</td>
</tr>
<tr>
<td>restart point</td>
<td>8, 18, 200, 205, 244–245</td>
</tr>
<tr>
<td>restart queue</td>
<td>27, 82–83, 95, 99</td>
</tr>
<tr>
<td>REXX Based tool</td>
<td>220–221, 241–242</td>
</tr>
<tr>
<td>REXX based tool</td>
<td>220–221, 241–242</td>
</tr>
<tr>
<td>Configuration file</td>
<td>227, 248</td>
</tr>
<tr>
<td>Functions</td>
<td>244</td>
</tr>
<tr>
<td>Inputs and outputs</td>
<td>248</td>
</tr>
<tr>
<td>Invocation</td>
<td>253</td>
</tr>
<tr>
<td>overview</td>
<td>241–242</td>
</tr>
<tr>
<td>source code</td>
<td>242</td>
</tr>
<tr>
<td>S</td>
<td></td>
</tr>
<tr>
<td>SAVINGDB</td>
<td>210–211</td>
</tr>
<tr>
<td>Schema</td>
<td>7, 14, 76, 110, 114</td>
</tr>
<tr>
<td>schema</td>
<td>7–8, 76, 88, 94, 232</td>
</tr>
<tr>
<td>search condition</td>
<td>81</td>
</tr>
<tr>
<td>Secondary Log File</td>
<td>62</td>
</tr>
<tr>
<td>segments</td>
<td>4, 7, 86, 114, 215, 255</td>
</tr>
<tr>
<td>send queue</td>
<td>76–77, 99, 147, 234</td>
</tr>
<tr>
<td>send queues</td>
<td>81–82, 88</td>
</tr>
<tr>
<td>Service Info Entry</td>
<td>23, 28, 147</td>
</tr>
<tr>
<td>set agent</td>
<td>46, 49, 203, 205</td>
</tr>
<tr>
<td>shutdown</td>
<td>6, 8, 201, 208</td>
</tr>
<tr>
<td>SIE</td>
<td>22–23, 147, 201</td>
</tr>
<tr>
<td>SIEs</td>
<td>22–23</td>
</tr>
<tr>
<td>Simple recovery mode scenario</td>
<td>209</td>
</tr>
<tr>
<td>SOFTWARE AG Adabas</td>
<td>2, 5</td>
</tr>
<tr>
<td>spill queue</td>
<td>83</td>
</tr>
<tr>
<td>SQLDA</td>
<td>6, 41</td>
</tr>
<tr>
<td>staging libraries</td>
<td>17, 93, 139</td>
</tr>
<tr>
<td>startup</td>
<td>6, 23, 198, 204</td>
</tr>
<tr>
<td>status</td>
<td>33–34, 74, 84, 104–105, 203–204, 234, 244</td>
</tr>
<tr>
<td>stored procedure</td>
<td>7, 28, 74, 79, 159, 165, 236</td>
</tr>
<tr>
<td>synchronize</td>
<td>8, 158</td>
</tr>
<tr>
<td>T</td>
<td></td>
</tr>
<tr>
<td>THROTTLE</td>
<td>48, 51</td>
</tr>
<tr>
<td>Topology</td>
<td>91</td>
</tr>
<tr>
<td>topology</td>
<td>11, 91–92</td>
</tr>
<tr>
<td>trace tables</td>
<td>82</td>
</tr>
<tr>
<td>TRANS</td>
<td>23, 147, 199–200, 232–233</td>
</tr>
<tr>
<td>transaction boundary</td>
<td>81</td>
</tr>
<tr>
<td>transmission queues</td>
<td>83</td>
</tr>
<tr>
<td>Trustworthy Insurance</td>
<td>8, 20, 86–88</td>
</tr>
<tr>
<td>two-phase commit</td>
<td>92</td>
</tr>
<tr>
<td>two-phase protocol</td>
<td>92</td>
</tr>
<tr>
<td>U</td>
<td></td>
</tr>
<tr>
<td>UOR</td>
<td>6, 37, 214</td>
</tr>
<tr>
<td>USE grammar</td>
<td>8–9, 14, 19, 107–110</td>
</tr>
<tr>
<td>Use grammar</td>
<td>8, 14, 108, 110</td>
</tr>
<tr>
<td>user application</td>
<td>74, 76</td>
</tr>
<tr>
<td>UTF</td>
<td>3, 7</td>
</tr>
<tr>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VSAM</td>
<td>2–3, 108, 139</td>
</tr>
<tr>
<td>W</td>
<td></td>
</tr>
<tr>
<td>warm start</td>
<td>6, 217</td>
</tr>
<tr>
<td>WebSphere Business Integration Message Broker</td>
<td>7, 74, 159, 235</td>
</tr>
<tr>
<td>WebSphere DataStage</td>
<td>7, 236</td>
</tr>
<tr>
<td>WebSphere MQ</td>
<td>2–3, 74–75, 87–88, 217, 234, 236</td>
</tr>
<tr>
<td>WIICEP for IMS</td>
<td>1, 4, 91, 96, 209</td>
</tr>
</tbody>
</table>
WIIEP architecture 75
WIIEP monitoring tools 84
WTO 32, 51, 211, 214
WTOR 52, 200–201

X
XM data gram 38–39, 200
XM data space queue 63, 69
XM queue 37, 39
XML format 6, 41, 77–78
XML message 9, 34, 75, 89, 116, 199, 232–233
XML publications 77–78, 153, 156
XML Publishing Toolkit 7, 159, 187, 231, 236

Z
Zoned decimal 138–139
zoned decimal 53, 137–138
This IBM Redbook provides an overview of IBM’s event publishing solution delivered via its WebSphere Information Integrator Classic Event Publisher and WebSphere Information Integrator Event Publisher products, and documents the procedure for implementing a replication and event publishing solution using these products.

The book is aimed at an audience of IT architects and database administrators (DBA) responsible for leveraging the WebSphere Information Integrator event publishing capabilities in a variety of business solutions involving replication and event alert systems. Using an insurance industry acquisition scenario involving auto insurance and home insurance companies, this book documents a step-by-step approach to implementing an integrating Customer Information Facility (CIF) application, and an Event Alert System application using WebSphere Information Integrator Classic Event Publisher and WebSphere Information Integrator Event Publisher.