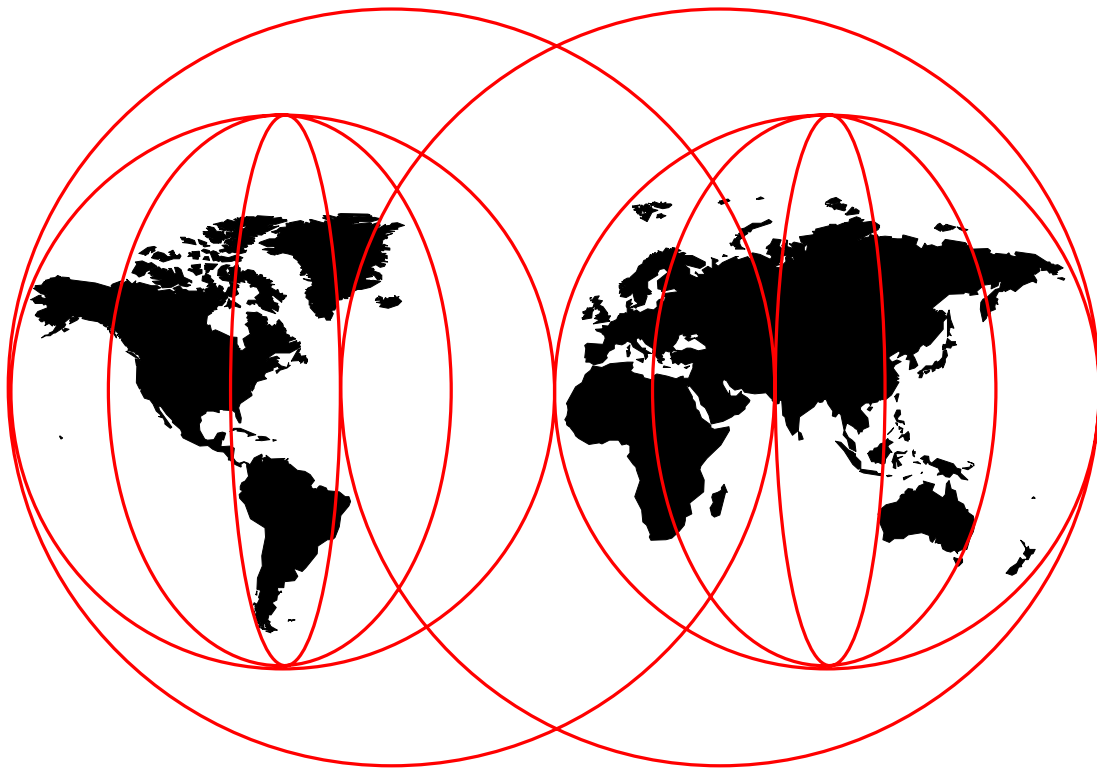


# **Business Process Model Implementation with CICS Business Transaction Services**

*Ulrich Claassen, Kay Johnson, Clement Yiu*



**International Technical Support Organization**

<http://www.redbooks.ibm.com>





International Technical Support Organization SG24-5464-00

**Business Process Model Implementation with  
CICS Business Transaction Services**

August 1999

**Take Note!**

Before using this information and the product it supports, be sure to read the general information in Appendix E, "Special Notices" on page 291.

**First Edition (August 1999)**

This edition applies to Version 1 Release 3 of CICS Transaction Server for OS/390, Program Number 5655-147, for use with the OS/390 Version 2 Release 5.

Comments may be addressed to:  
IBM Corporation, International Technical Support Organization  
Dept. QXXE Building 80-E2  
650 Harry Road  
San Jose, California 95120-6099

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

© **Copyright International Business Machines Corporation 1999. All rights reserved.**

Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

---

# Contents

<b>Figures</b> .....	ix
<b>Tables</b> .....	xiii
<b>Preface</b> .....	xv
The Team That Wrote This Redbook .....	xv
Comments Welcome .....	xvi

---

<b>Part 1. CICS BTS Overview</b> .....	1
<b>Chapter 1. Introduction</b> .....	3
<b>Chapter 2. Concepts and Components</b> .....	7
2.1 CICS BTS Terminology .....	7
2.2 Components .....	10
2.2.1 Application Programming Interface .....	10
2.2.2 System Programming Interface .....	12
2.2.3 CICS BTS Files .....	12
2.3 A Business Transaction with and without CICS BTS .....	14
2.4 Process and Activity Execution .....	16
2.5 User Syncpoints .....	17
2.6 Activity Execution .....	17
2.6.1 Synchronous Execution .....	18
2.6.2 Asynchronous Execution .....	19
2.7 Process Flow .....	20
2.8 Rules to Access a Process or Activity .....	22
2.9 Activity Modes Appearing During Process Execution .....	23
2.10 Using Containers to Communicate between Programs .....	24
2.11 Events .....	26
2.11.1 Atomic Events .....	27
2.11.2 Composite Events .....	28
2.11.3 Deleting Events .....	28
2.11.4 Handling Events .....	29
2.12 Name Scope Summary .....	31
2.13 CBAM Transaction .....	32
2.14 Audit Logging .....	34
<b>Chapter 3. Base API: SALES Application</b> .....	37
3.1 Overview .....	37
3.2 Transactions and Programs .....	38
3.3 What This Application Introduces .....	38
3.3.1 Coverage of API .....	38
3.4 The Initiating Transaction (MENU) .....	39
3.4.1 BTS Resources Used .....	39
3.4.2 High Level Logic .....	39
3.5 The Root Activity .....	41
3.5.1 System Event DFHINITIAL .....	41
3.5.2 Composite Event DEL-INV-COMPLETE .....	47
3.5.3 Timer Event SEND-PAY-REMINDER .....	49

3.5.4	Completion Event REM-COMPLETE	51
3.5.5	Input Event DELETE-TIMER	52
3.5.6	Activities (Children)	54

---

**Part 2. More Complex Application Systems** . . . . . 57

<b>Chapter 4. HOLIDAY Application</b>	59
4.1 Compensation	59
4.2 HOLIDAY Application Overview	59
4.3 Transactions, Programs, Containers, and Control File	61
4.3.1 Transactions and Programs	61
4.3.2 Container Contents	61
4.3.3 The Control File	63
4.3.4 Coverage of API	64
4.4 The Initiating Transaction	65
4.4.1 BTS Resources Used	65
4.4.2 The High-Level Logic	66
4.5 The Root Activity	68
4.5.1 BTS Resources Used	69
4.5.2 System Event DFHINITIAL	70
4.5.3 Composite Event BOOK-HOLIDAY - No Retry Container	74
4.5.4 Composite Event BOOK-HOLIDAY - With Retry Container	81
4.5.5 Timer Event NO-RESPONSE	82
4.5.6 Composite Event CANCEL-HOLIDAY	86
4.5.7 Other Functionality	87
4.6 Booking Child Activities	90
4.6.1 BTS Resources Used	90
4.6.2 System Event DFHINITIAL	91
4.7 Compensating Child Activities	93
4.7.1 BTS Resources Used	93
4.7.2 System Event DFHINITIAL	94
4.8 TD Output, Audit Log, and CBAM Output	95
4.8.1 The TD Queue Output	96
4.8.2 Audit Log for the HOLIDAY Application	97
4.8.3 CBAM Displays	100
<b>Chapter 5. MORTGAGE and FINANCE Application</b>	103
5.1 Communication between Processes	103
5.1.1 Starting Partner Process	103
5.1.2 Communication between Two Active Processes	104
5.2 The SUSPEND and RESUME Commands	104
5.3 The MORTGAGE and FINANCE Application	104
5.4 Transactions, Programs, Containers, and Control File	106
5.4.1 Transactions and Programs	106
5.4.2 Container Contents	106
5.5 The Initial Transactions	107
5.5.1 BTS Resources Used in the FINANCE Root Activity	108
5.5.2 BTS Resources Used in the MORTGAGE Root Activity	109
5.6 A Communicator Program	110
5.7 Forcing a Change to the Interest Rate	116

---

**Part 3. Designing and Developing Your Application** . . . . . 127

<b>Chapter 6. Designing Your Application</b>	129
6.1 Business Process	129
6.2 Business Process Analysis	132
6.2.1 Analysis of Requirements	132
6.2.2 Example of a Business Process	133
6.3 Application Design Extracted from a Business Process Model	136
6.3.1 Process NEWEMPLOYEE, Root Activity, Event DFHINITIAL	137
6.3.2 Process NEWEMPLOYEE, Activity NEWJD, Event DFHINITIAL	138
6.3.3 Process NEWEMPLOYEE, Activity NEWEMPL, Event DFHINITIAL	139
6.3.4 Process NEWEMPLOYEE, Activity NEWJD, Event JD-REL-COMPLETE	139
6.3.5 Process NEWEMPLOYEE, Activity NEWEMPL, Event EMPLREL-COMPLETE	140
6.3.6 Process NEWEMPLOYEE, Activity NEWJD, Event ROLE-CS-CONNECT	141
6.3.7 Process NEWEMPLOYEE, Activity NEWEMPL, Event CONNECT-EMP-CS	142
6.3.8 Process NEWEMPLOYEE, Root Activity, Event JD-EMPL-COMPLETE	143
6.3.9 Process NEWEMPLOYEE, Root Activity, Input Event NEWEMP or Timer Event NOREQ	144
<b>Chapter 7. Developing Your Application</b>	147
7.1 Program Development	147
7.2 Pseudo-Code of the Company Organization Employee Application	147
7.2.1 Pseudo-Code of the Main Program EMPLMAIN	149
7.2.2 Pseudo-Code of the Root Activity Program of the NEWEMPLOYEE Process	150
7.2.3 Pseudo-Code of the Activity NEWEMPL	156
7.2.4 Pseudo-Code of the Activity NEWJD	161
7.3 Hints and Tips	168
7.3.1 Activities	168
7.3.2 Root Activity	168
7.3.3 User Syncpoints	168
7.3.4 Administration Add-Ons to Your Application	169
7.3.5 A Useful Tool	169
<b>Chapter 8. Reuse of Elements</b>	171
8.1 Reuse of Existing Applications	171
8.1.1 Reuse of 3270 Transactions	171
8.1.2 Reuse of Non-Terminal Transactions and Programs	171
8.2 Reuse of CICS BTS Elements	172
8.3 Reuse Management	172
8.3.1 Process	172
8.3.2 Activities	172
8.4 Mapping Messages to Context	173
<b>Chapter 9. Programming Paradigms</b>	175
9.1 Traditional Transaction Model and Business Transaction Model	175
9.1.1 Conversational and Pseudo-Conversational	175
9.1.2 Multi-Tier Model	177
9.1.3 Business Transaction Model	180
9.2 Sharing Data in and across Transactions	180
9.3 Task Synchronization	182

9.3.1 Non-CICS BTS Commands	182
9.3.2 CICS BTS Tools	183
9.4 Recovery Considerations	184
9.5 Dealing with Exception Conditions	184
9.6 Access to System Information	185
9.7 Program Structure	185
9.7.1 Parent and Child Model	186
9.7.2 Client/Server Model	188

---

## Part 4. Universal Process Driver . . . . . 193

<b>Chapter 10. Universal Process Driver Model</b>	195
10.1 Database Process Control Catalogue	195
10.2 Process Definition	197
10.3 Universal Process Driver Global Description	197
10.4 Example of a Process Description in Process Control Catalogue	199
10.5 Conclusion	203

---

## Part 5. System Considerations . . . . . 205

<b>Chapter 11. CICS Business Transaction Services Setup</b>	207
11.1 Process Resource Definition	207
11.2 Local Request Queue Data Set	208
11.3 CICS BTS Repository Data Set	209
11.4 Distributed Routing Program	211
11.5 Audit Log	211
11.5.1 Sample JCL to Update The Logger Policy for The Audit Log	211
11.5.2 Printing The Audit Log Contents	213

<b>Chapter 12. Problem Determination in BTS Applications</b>	215
12.1 Application Design Errors	215
12.2 Restarting Stuck Processes	216
12.2.1 Using Activity Timers	216
12.2.2 Using Process Timers	216
12.2.3 Using Status Containers	216
12.2.4 Using A Utility Program	217
12.3 Activity Abends	217

<b>Chapter 13. CICS Business Transaction Services Utilities</b>	219
13.1 Audit Log	219
13.1.1 Specifying the Level of Audit Logging	219
13.1.2 Sample JCL to Print the Audit Log Contents	221
13.1.3 Example Output from the DFHATUP Utility	222
13.2 Repository Data Set	227
13.2.1 The Repository Utility Program, DFHBARUP	227
13.2.2 Sample JCL to Print the Repository	227
13.2.3 Example Output from the DFHBARUP Utility	229
13.3 Operator Commands	233
13.3.1 CEMT INQUIRE PROCESSTYPE	233
13.3.2 CEMT INQUIRE TASK	235
13.3.3 CEMT SET PROCESSTYPE	236



<b>Appendix A. Company Organization Application Development Log</b> . . . .	239
A.1 Resource Definitions . . . . .	239
A.2 Program Development Problem Log . . . . .	252
<b>Appendix B. CICS BTS Commands</b> . . . . .	265
B.1 Application Programming Interface (API) . . . . .	265
B.2 System Programming Interface (SPI) . . . . .	266
<b>Appendix C. UDP Process Control Catalogue</b> . . . . .	267
<b>Appendix D. Glossary</b> . . . . .	281
<b>Appendix E. Special Notices</b> . . . . .	291
<b>Appendix F. Related Publications</b> . . . . .	295
F.1 International Technical Support Organization Publications . . . . .	295
F.2 Redbooks on CD-ROMs . . . . .	295
F.3 Other Publications . . . . .	295
<b>How to Get ITSO Redbooks</b> . . . . .	297
IBM Redbook Fax Order Form . . . . .	298
<b>List of Abbreviations</b> . . . . .	299
<b>Index</b> . . . . .	301
<b>ITSO Redbook Evaluation</b> . . . . .	303



---

# Figures

1.	Classic CICS and CICS Business Transaction Services	3
2.	Symbol Used for Activities	8
3.	Symbols Used for Containers	9
4.	Symbols Used for Events	9
5.	Symbol Used for Timers	9
6.	Symbol Used for Deletion of an Object	9
7.	CICS BTS Components	14
8.	A Business Transaction without CICS BTS	15
9.	A Business Transaction with CICS BTS	16
10.	Synchronous Execution of Activities	19
11.	Asynchronous Execution of Activities	20
12.	A Simple Process Flow	21
13.	CICS BTS Activity Modes	24
14.	Availability of Containers	25
15.	Event Handling	26
16.	Event Handling with Atomic Events	30
17.	Event Handling with Composite Events	31
18.	The CBAM List of Current Process Types	32
19.	Current Processes of a Selected Type	32
20.	Details of Process SALES333111	33
21.	Details of the Root Activity	33
22.	Resource Display (Container) of an Activity	33
23.	Events Known by the Root Activity	34
24.	Timer Event Details	34
25.	Pseudo-Code to Run a CICS BTS Process	39
26.	Process SALES Activation with DFHINITIAL	42
27.	Pseudo-Code for the SALES Process Root Activity	43
28.	Process SALES, Activation with DEL-INV-COMPLETE	47
29.	Process SALES, Pseudo-Code of Activation with DEL-INV-COMPLETE	48
30.	Process SALES, Activation with SEND-PAY-REMINDER	49
31.	Process SALES, Pseudo-Code of Activation with SEND-PAY-REMINDER	50
32.	Process SALES, Activation with REMINDER-COMPLETE	51
33.	Process SALES, Pseudo-Code of Activation with REMINDER-COMPLETE	52
34.	Process SALES, Activation with DELETE-TIMER	53
35.	Process SALES, Pseudo-Code of Activation with DELETE-TIMER	54
36.	Categorized Child Activity	55
37.	CICS BTS, Pseudo-Code of an Activity	55
38.	Contents of Containers	62
39.	Panels Used to Update the Control File	64
40.	Pseudo-Code for the Initiating Transaction	67
41.	Flow of Program when Attached by DFHINITIAL	70
42.	Pseudo-Code when Attached by DFHINITIAL	71
43.	Flow of Program when Reattached by BOOK-HOLIDAY - No Retry Container	74
44.	Pseudo-Code when Attached by BOOK-HOLIDAY - No Retry Container	76
45.	Flow of Program when Reattached by BOOK-HOLIDAY - With Retry Container	82
46.	Flow of Program when Reattached by NO-RESPONSE	83
47.	Pseudo-Code when Attached by NO-RESPONSE	84
48.	Flow of Program when Reattached by CANCEL-HOLIDAY	87

49.	Flow of Program when There Is a Need to COMPENSATE	88
50.	Pseudo-Code for TIDY-UP-CHILDREN	89
51.	Pseudo-Code for Booking an Element	92
52.	Pseudo-Code for Cancelling a Booking	95
53.	Holiday Audit Output	96
54.	Working Storage and Pseudo-code for the WHICH Output	97
55.	Holiday Audit Printout	98
56.	Holiday CBAM Process Screen Dump	100
57.	Holiday CBAM Activity Screen Dump	100
58.	Holiday CBAM Activity DFHROOT Screen Dump	101
59.	Holiday CBAM Activity DFHROOT Containers Screen Dump	101
60.	Holiday CBAM Activity DFHROOT Events Screen Dump	102
61.	Holiday CBAM Activity DFHROOT Timers Screen Dump	102
62.	Contents of Containers	107
63.	Flow of Programs for the Start of the Mortgage and Finance Application	107
64.	Flow of Activities for Making a Payment	111
65.	Pseudo-Code for the MAKEPAY Activity	112
66.	Pseudo-Code for the ACQUIREP Activity	115
67.	Pseudo-Code for the Program to Suspend and Resume a Process	117
68.	Robert Anthony's Organizational Pyramid	130
69.	Illustration of a Business Process and Its Subsequent Elements	132
70.	Part of the Business Process Model (1)	134
71.	Part of the Business Process Model (2)	135
72.	Transactions Needed to Add a New Employee	136
73.	Process NEWEMPLOYEE, Event DFHINITIAL	137
74.	Process NEWEMPLOYEE, Activity NEWJD, Event DFHINITIAL	138
75.	Process NEWEMPLOYEE, Activity NEWEMPL, Event DFHINITIAL	139
76.	Process NEWEMPLOYEE, Event JD-REL-COMPLETE	140
77.	Process NEWEMPLOYEE, Event EMPLREL-COMPLETE	141
78.	Process NEWEMPLOYEE, Event ROLE-CS-CONNECT	142
79.	Process NEWEMPLOYEE, Event CONNECT-EMP-CS	143
80.	Process NEWEMPLOYEE, Event JD-EMPL-COMPLETE	144
81.	Process NEWEMPLOYEE, Input Event NEWEMP	145
82.	Pseudo-Code for the EMPLMAIN Program	149
83.	Pseudo-Code for the EMPLROOT Program	151
84.	Pseudo-Code for the EMPLACTM Program	157
85.	Pseudo-Code for the JOBEACTM Program	162
86.	Program Structure in a Single-Program Transaction	176
87.	Program Structure in a Pseudo-conversational Transaction	177
88.	Two Tier Model with Separation of the Presentation Logic from the Business Logic	178
89.	Control, Presentation, Table Driven, and Business Logic Model	179
90.	Pseudo-code of a Root Activity	187
91.	Pseudo-code of the Mainline of a Client Program	189
92.	Pseudo-code of the Mainline of a Server Program	190
93.	Process Control Catalogue for a Universal Process Driver	197
94.	Universal Process Driver Logic with Event DFHINITIAL	198
95.	Universal Process Driver Logic with an Event Other Than DFHINITIAL	199
96.	CEDA View of a Process Type - Sales Sample Application	207
97.	JCL to Create a Local Request Queue Data Set	208
98.	JCL to Create a CICS BTS Repository Data Set	209
99.	File Control Resource Definition for the CICS BTS Repository Data Set	210
100.	JCL to Update the Logger Policy for the Audit Log	212
101.	Journal Model Resource Definition for the Audit Log	213

102. Sample JCL to Print the Audit Log Contents . . . . .	221
103. Example Control Statements to Format All the Records for the SALES1234567890 . . . . .	222
104. Sample Audit Log Printout . . . . .	223
105. Sample JCL to Print the Repository for the ORDER Activity of the CUSTSALES1999.13872977829728.QA Process . . . . .	228
106. Example Control Statements to Format All Records on the SALEREP Repository . . . . .	229
107. Sample Repository Printout . . . . .	230
108. CEMT INQUIRE PROCESSTYPE Screen . . . . .	233
109. The Expanded Display of an Individual Entry . . . . .	234
110. CEMT INQUIRE TASK Screen . . . . .	235
111. The Expanded Display of an Individual Entry . . . . .	236
112. Sample JCL to Define Repository Dataset . . . . .	239
113. Processtype Resource Definition . . . . .	240
114. Sample JCL to Define Audit Log . . . . .	240
115. Journal Model Resource Definition for the Audit Log . . . . .	241
116. File Control Resource Definition for the CICS BTS Repository . . . . .	243
117. Sample JCL to Define Resource for the Organization Application . . . . .	244
118. Resource Definition for the Organization Application . . . . .	245
119. Company Organization Employment Entry Map Definition . . . . .	248
120. Company Organization Employment Entry Map Diagram . . . . .	249
121. Sample JCL to Compile the Employment Entry Map . . . . .	250
122. CBAM Screen Dump . . . . .	252
123. CBAM Processtype Screen Dump . . . . .	252
124. CBAM Process Screen Dump . . . . .	252
125. CBAM Activity Screen Dump . . . . .	253
126. CBAM Process Screen Dump . . . . .	253
127. CBAM Activity Screen Dump . . . . .	253
128. CBAM Event Screen Dump . . . . .	254
129. CBAM Activity Screen Dump . . . . .	254
130. AEZJ Explanation . . . . .	254
131. CBAM Process Screen Dump . . . . .	255
132. CBAM Activity Screen Dump . . . . .	255
133. CBAM Event Screen Dump . . . . .	255
134. CBAM Activity Screen Dump . . . . .	256
135. CBAM Event Screen Dump . . . . .	256
136. CBAM Activity Screen Dump . . . . .	256
137. CBAM Event Screen Dump . . . . .	257
138. CBAM Activity Screen Dump . . . . .	257
139. CBAM Event Screen Dump . . . . .	257
140. CBAM Process Screen Dump . . . . .	258
141. CBAM Activity Screen Dump . . . . .	258
142. CBAM Container Screen Dump . . . . .	258
143. CBAM Event Screen Dump . . . . .	258
144. SUBEVENT Abend . . . . .	259
145. CBAM Process Screen Dump . . . . .	259
146. CBAM Activity Screen Dump . . . . .	259
147. CBAM Container Screen Dump . . . . .	260
148. CBAM Event Screen Dump . . . . .	260
149. CBAM Process Screen Dump . . . . .	260
150. CBAM Process Screen Dump . . . . .	261
151. CBAM Activity Screen Dump . . . . .	261
152. CBAM Event Screen Dump . . . . .	261

153. CBAM Activity Screen Dump . . . . .	262
154. CBAM Event Screen Dump . . . . .	262
155. Define ACTIVITY Abend . . . . .	262
156. CBAM Process Screen Dump . . . . .	263
157. CBAM Event Screen Dump . . . . .	263
158. CBAM Process Screen Dump . . . . .	264
159. CICS BTS Application Programming Interface Commands, Part 1 . . . . .	265
160. CICS BTS Application Programming Interface Commands, Part 2 . . . . .	266
161. CICS BTS System Programming Interface Commands . . . . .	266
162. UDP Process Control Catalogue, Part 1 of 20 . . . . .	267
163. UDP Process Control Catalogue, Part 2 of 20 . . . . .	267
164. UDP Process Control Catalogue, Part 3 of 20 . . . . .	268
165. UDP Process Control Catalogue, Part 4 of 20 . . . . .	268
166. UDP Process Control Catalogue, Part 5 of 20 . . . . .	269
167. UDP Process Control Catalogue, Part 6 of 20 . . . . .	269
168. UDP Process Control Catalogue, Part 7 of 20 . . . . .	270
169. UDP Process Control Catalogue, Part 8 of 20 . . . . .	270
170. UDP Process Control Catalogue, Part 9 of 20 . . . . .	271
171. UDP Process Control Catalogue, Part 10 of 20 . . . . .	271
172. UDP Process Control Catalogue, Part 11 of 20 . . . . .	272
173. UDP Process Control Catalogue, Part 12 of 20 . . . . .	273
174. UDP Process Control Catalogue, Part 13 of 20 . . . . .	273
175. UDP Process Control Catalogue, Part 14 of 20 . . . . .	274
176. UDP Process Control Catalogue, Part 15 of 20 . . . . .	275
177. UDP Process Control Catalogue, Part 16 of 20 . . . . .	276
178. UDP Process Control Catalogue, Part 17 of 20 . . . . .	277
179. UDP Process Control Catalogue, Part 18 of 20 . . . . .	278
180. UDP Process Control Catalogue, Part 19 of 20 . . . . .	279
181. UDP Process Control Catalogue, Part 20 of 20 . . . . .	280

---

## Tables

1.	CICS BTS Activity Execution . . . . .	17
2.	Commands Used to Delete Events . . . . .	29
3.	The Sales Application Programs . . . . .	38
4.	The Holiday Application Programs . . . . .	61
5.	The Finance Application Programs Discussed in This Book . . . . .	106
6.	The Mortgage Application Programs Discussed in This Book . . . . .	106
7.	The Programs Applicable to Both Applications . . . . .	106
8.	Short and Long Event Names . . . . .	148
9.	Message Mapping Table . . . . .	174
10.	Subroutine Name Which Creates the Event in the Pseudo-code . . . . .	188
11.	ProcessTable . . . . .	199
12.	ActivityTable . . . . .	200
13.	EventTable . . . . .	201
14.	SubEventTable . . . . .	201
15.	ProcessControlTable, Part 1 . . . . .	202
16.	ProcessControlTable, Part 2 . . . . .	203
17.	Activity, Transaction, Program, and Event Names in the Application . . . . .	251





---

## Preface

This redbook is primarily intended for CICS customers who plan to implement the CICS Business Transaction Services (BTS) application development paradigm. This extension to the CICS API and the accompanying support services are provided by CICS Transaction Server (CICS TS) for OS/390 Version 1 Release 3. CICS BTS helps you to implement your business transaction in CICS and allows the system to control the execution times and steps that are needed to complete it.

We start by introducing base CICS BTS concepts and terminology and show use of the new API commands and CICS BTS entities by three applications. The sample applications implement simple as well as more complex concepts, such as compensation and interprocess communication.

We show how top-down business entity relationship analysis can lead to definition of CICS BTS processes and activities in a way suitable for your organization, and we develop another application based on this analysis. We also suggest an initial design of a universal process driver that can help you automate the design and development of CICS BTS applications.

Finally, we provide guidance on how to customize your CICS TS system for support of CICS BTS and how to use the new utilities and perform problem determination.

---

## 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.

**Ulrich Claassen** is working at the Buchungszentrale der westf.-lipp. Sparkassen GmbH (BWS) in Muenster, Germany. He is responsible for the company's CICS environment architecture and developed a CICS dialogue control system and a process management system. He has 15 years of CICS application design and programming and project management. Ulrich usually manages major architectural projects for his company.

**Kay Johnson** is a System Tester at the Hursley Laboratory in England. She has been writing applications and testing CICS Business Transaction Services, Intersystem Communication, and Restart/Recovery for CICS TS 1.3. Having graduated from Plymouth University with first class honors in mathematics and computing, Kay joined IBM in September 1997. Prior to this, Kay worked as a system analyst and application developer.

**Clement Yiu** is a project manager in Application Development and Maintenance business unit at the IBM Global Services, Australia. He has worked with CICS application development since 1983. His other areas of expertise are DB2 and VisualAge products.

The project was designed and managed by:

**Eugene Deborin**,  
International Technical Support Organization, San Jose Center.

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

**Robert Haimowitz,**  
International Technical Support Organization, Raleigh Center

**Ian Mitchell,**  
IBM UK, Hursley Laboratories

Thanks also to the following people for the time they spent reviewing this book and for their suggestions for improvement:

**Jean Paul Caron,**  
IBM France

**Karl-Heinz Marquardt,**  
IBM Germany

**Markus Muetschard,**  
International Technical Support Organization, San Jose Center

**Nigel Williams,**  
IBM France, Parallel Solutions Support Center

**Bob Yelavich,**  
Consultant

---

## Comments Welcome

### Your comments are important to us!

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

- Fax the evaluation form found in "ITSO Redbook Evaluation" on page 303 to the fax number shown on the form.
- Use the online evaluation form found at <http://www.redbooks.ibm.com/>
- Send your comments in an Internet note to [redbook@us.ibm.com](mailto:redbook@us.ibm.com)

---

## Part 1. CICS BTS Overview



# Chapter 1. Introduction

CICS applications have traditionally been designed to consume a minimal amount of system resources. Typically, they were developed to execute pseudo-conversationally. Sometimes they were programmed conversationally if there was a specific reason for that. In order to support the work routine comfortably, it was necessary to implement control code into each program. This was the only way to supervise dependencies over several conversation steps. The actual problem of running many activities that are dependent on each other could be mastered only by extensive control logic in all transactions involved.

CICS now offers additional programming tools to help the customer better manage his business by implementing business processes under system control. All essential control and supervision needed to run a business process using CICS transactions have been implemented by the introduction of the new CICS Business Transactions Services (CICS BTS).

CICS BTS provides a new way of programming for CICS. It is implemented as a set of new EXEC CICS commands that extend the current CICS application programming interface (API) to make it easier to model, control, and execute complex business transactions. An instance of a running business transaction is called a *process*. A process is a collection of one or more BTS *activities*. An activity is a basic unit of BTS execution and it is mapped to a traditional CICS transaction (see Figure 1).

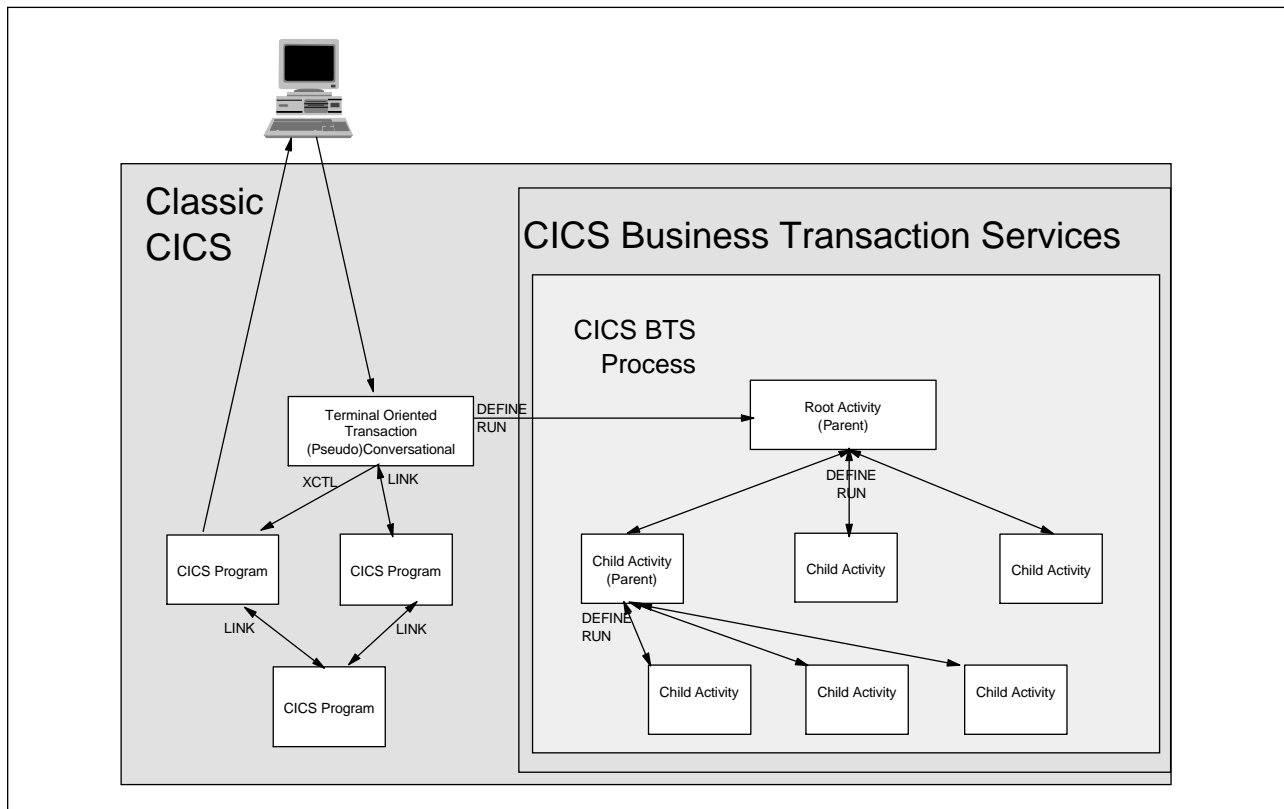


Figure 1. Classic CICS and CICS Business Transaction Services

Figure 1 illustrates how CICS BTS complements the traditional CICS environment. A process is organized hierarchically. Data exchange is done through data-containers, named areas of storage, associated with a particular process or activity, and maintained by BTS. CICS BTS signals progress in a process using BTS *events*. An event informs an activity that an action is required or has completed. Events are given names to make them recognizable by CICS BTS.

A business transaction can be defined as an information technology implementation of business entities that have been defined as a consequence of business process analysis.

From a business process, the business activity is defined. The result is the CICS BTS process. It will be necessary to analyze the work routine to establish the activities that make up the process. A business process can last for seconds; however, it can also last for hours, days, or even weeks. During the lifespan of a business transaction (process), the following could occur:

- Nothing happens until the last event has occurred (for example, the end of a pre-determined time interval is reached).
- Many independent tasks are executed in varying time spans.
- A cascade of independent and dependent tasks are completed.

In order to support this, CICS makes available a set of new commands within the existing API and new services that facilitate implementation of entire business transactions under system control. In the past, you designed CICS transactions that interacted with the system and corresponded to an element of the business process that you implemented. Thus, you have implemented a business transaction using a collection of traditional CICS transactions as a technical vehicle. Most likely, these CICS transactions were scattered over a set of different applications. Moreover, you had to include a significant amount of control logic in the program design. Some tasks needed to complete a business transaction required some sort of manual intervention; so, the system support for the business transaction was interrupted.

Now, it is possible to completely ascertain the technical design of a new application based on what actually happens in practice. It can be applied above all to new applications, but existing applications can also profit from this new technology. Existing programs can be used within business processes. This is possible using the new methods CICS makes available.

With a CICS BTS business transaction, you merely define CICS BTS activity. This can then serve as a wrapper for the existing program, which can be called with existing CICS techniques.

CICS BTS supports reuse of existing 3270 applications as well as reuse of programs. It is possible to invoke complete conversational and pseudo-conversational dialogues and treat them as CICS BTS activities within one CICS BTS business transaction using the 3270 bridge.

CICS BTS can help you to implement more system support for your business transactions. You can clearly separate system control logic from business logic, which in turn promotes reuse of business logic parts in several different applications. CICS BTS also can help you to start a transition from running classic

3270 applications to a multi-tier environment using new technologies without redefining your existing business logic.

With this book, we want to explain the new programming model that CICS BTS provides. Therefore, we use some sample applications, discuss several design aspects, and give an overview of the installation requirements.

Within your business, you may talk about business processes and activities. These processes and activities are likely to be the result of the analysis of the actual business practices within your organization.

CICS BTS introduces the concept of the CICS BTS process and CICS BTS activity. To avoid confusion, particularly in our discussions of business analysis, we specifically refer to CICS BTS process and CICS BTS activity to distinguish from business processes and activities within an organization.





---

## Chapter 2. Concepts and Components

CICS BTS comprises a set of new CICS commands and services that allow easier implementation of entire business transactions under system control. The distinction between a business transaction and a CICS transaction is that a business transaction normally is of a longer lived nature, such as hours, days, even weeks. A business transaction lives, for example, from the time some kind of business contract is agreed upon until the final payment for delivered goods is booked and completed. A CICS transaction, on the other hand, normally should take seconds or fractions of a second to complete.

The concept of expanding the CICS services is based on the need to portray complex business processes so that the program code only implements individual business steps or business tasks to be supported. That is, each element of a business process, the business process step, is analyzed and designed independently by looking at the ways the element interacts with other elements and outside inputs. With the new capability of CICS, it is possible to implement the tasks following the analysis of the business process and each individual business step. The outside inputs can be accepted from non-CICS BTS transactions as input events. This event then can be processed by the business logic in an activity. The single elements of a CICS BTS process are implemented as activities. These comprise the solution of a relatively small business task or tasks. If more than one business task is to be implemented by one activity, the decision which task has to be executed in a given activation of the activity can be done by the activity analyzing its data-container. This concept provides function hiding within each activity.

With this new view of CICS programming comes a set of new concepts, which we describe in this chapter.

---

### 2.1 CICS BTS Terminology

In this section we introduce the CICS BTS terminology.

- INITIAL REQUEST:

A CICS transaction that starts a CICS BTS process.

- PROCESS:

A collection of one or more CICS BTS activities. It has a unique 36-character name by which it can be referenced and invoked. Typically, a process is an instance of a business transaction.

A process can be categorized by the process type. A process is controlled by a program that represents the root activity. A process is not always active, but it stays alive in CICS until it is entirely completed, that is, until all its activities have completed.

A process is represented as a block of storage containing information relevant to its execution. It is also associated with at least one additional block of information called an activity instance. When not executing, a process and its activity instances reside in a repository.

The CICS BTS repository data set is a VSAM KSDS data set that holds state information about the CICS BTS processes as well as application data

(container contents) that is passed among activities or different activations of the same activity. You can define only one data set to hold information for all of your CICS BTS process types, or you can define a separate data set for each process type, or any combination of these two options.

A process can be distributed over several CICS regions by dynamic routing capabilities provided by CICS TS 1.3 with or without CICSplex SM.

- **PROCESS TYPE:**

Defines a CICS BTS business transaction managed by CICS. The CICS definition is done through resource definition online (RDO) or through CICSplex SM Business Application Services (BAS) and assigns resources (repository, audit log). Using several process types, you can categorize your application. This is useful, for example, for browsing purposes or accounting.

- **ACTIVITY:**

The basic unit of CICS BTS execution. Typically, it represents one of the actions of a CICS BTS business transaction. Activities implement the business logic. An activity is executed by a normal CICS transaction responding to CICS BTS events. To do this, it uses the CICS BTS API. A running activity can be identified by its unique 52 character ACTIVITYID assigned by CICS.

An activity that starts another activity is called a *parent activity*. An activity that is started by another is called a *child activity*. An activity gets activated (several times if necessary) by CICS BTS. Every activation is represented by a new CICS transaction. When an activity completes, the defined completion event is signaled to its parent.

We illustrate activities in all figures found in this book as shown in Figure 2.

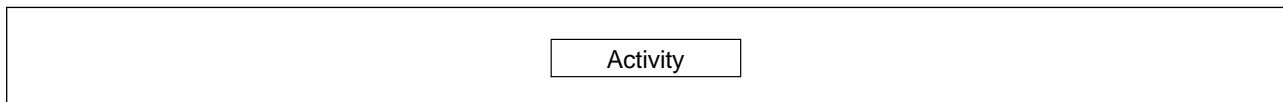


Figure 2. Symbol Used for Activities

- **ROOT ACTIVITY:**

The activity at the top of the activity tree (it has no parent activity). The root activity normally is the control program for a business transaction that represents the start and the end of the process. It initiates and controls a set of child activities.

A root activity always has a CICS-assigned name DFHROOT.

- **DATA-CONTAINER:**

A named area of storage (up to 16 characters), associated with a particular process or activity, maintained by CICS BTS. Each process or activity can have any number of containers. It is a kind of COMMAREA with no 32 KB limit, which is saved in the repository.

We illustrate a container in all figures found in this book as shown in Figure 3 on page 9.

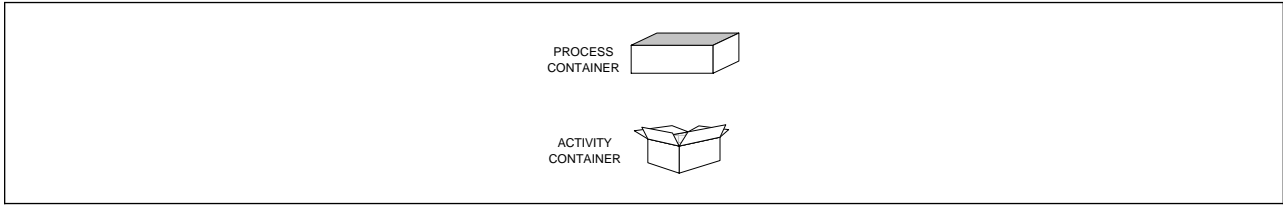


Figure 3. Symbols Used for Containers

- EVENT:

A CICS BTS event is a means by which CICS BTS signals the progress in a process. It informs an activity that an action is required or has completed. Each event gets a unique name within a process and is used to affect the flow of a process.

We illustrate an event in all figures found in this book as shown in Figure 4.

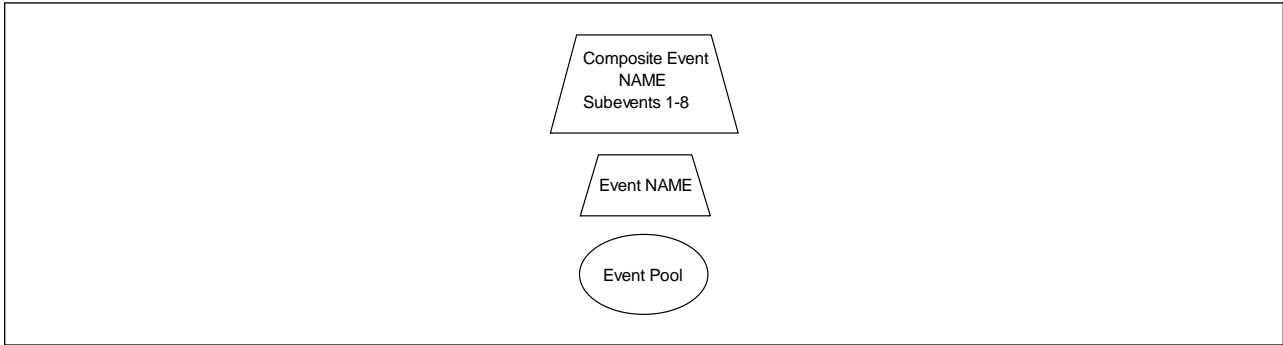


Figure 4. Symbols Used for Events

- TIMER:

A CICS BTS object that expires when the system time becomes greater than a specified date and time or after a specified period has elapsed. Each timer has an event associated with it. The event occurs (*fires*) when the timer expires.

We illustrate a timer in all figures found in this book as shown in Figure 5.

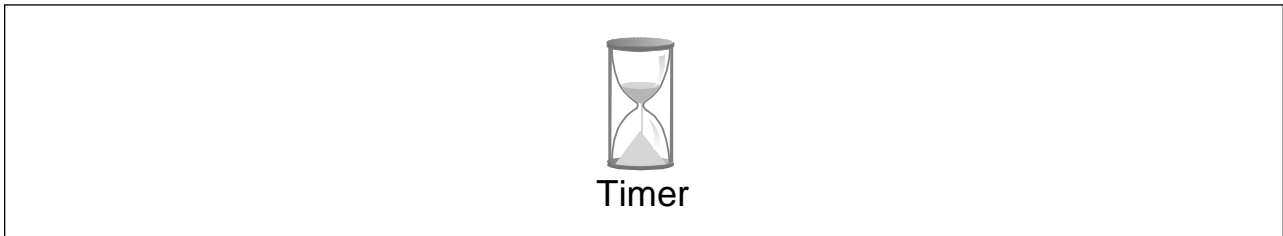


Figure 5. Symbol Used for Timers

We illustrate a deletion of an element in all figures found in this book as shown in Figure 6.

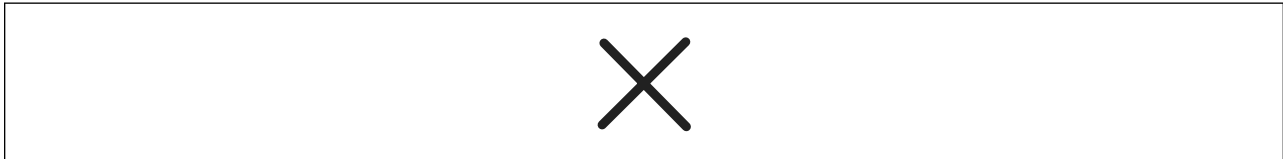


Figure 6. Symbol Used for Deletion of an Object

It is possible to issue a DELETE command against the following CICS BTS elements:

- Activity
- Container
- Event
- Timer

---

## 2.2 Components

CICS BTS consists of several components. From a developer's point of view, the most essential is the supplemented API.

### 2.2.1 Application Programming Interface

In this section we provide a list of new API commands and mention existing commands that have new CICS BTS-related attributes. The commands are logically grouped together according to their function.

- Process Handling
  - CHECK ACQPROCESS** Check the completion status of a process
  - DEFINE PROCESS** Define a CICS BTS process
  - ENDBROWSE PROCESS** End a browse of processes of a specified type
  - GETNEXT PROCESS** Browse all processes of a specified type
  - INQUIRE PROCESS** Retrieve the attributes of a process
  - LINK ACQPROCESS** Execute a CICS BTS process synchronously
  - RESET ACQPROCESS** Reset a process to its initial state
  - STARTBROWSE PROCESS** Start a browse of all processes of a specified type
- Activity Handling
  - ACQUIRE** Acquire access to an activity from outside the process that contains it
  - ASSIGN** Retrieve details of the activity on behalf of which the transaction is executing
  - CHECK ACTIVITY** Check the completion status of an activity
  - DEFINE ACTIVITY** Define a CICS BTS activity
  - DELETE ACTIVITY** Delete a child activity
  - ENDBROWSE ACTIVITY** End a browse of the child activities of an activity, or of the descendant activities of a process
  - GETNEXT ACTIVITY** Browse the child activities of an activity, or the descendant activities of a process
  - INQUIRE ACTIVITYID** Retrieve the attributes of an activity
  - LINK ACTIVITY** Execute a CICS BTS activity synchronously

<b>RESET ACTIVITY</b>	Reset an activity to its initial state
<b>RETURN</b>	The ENDACTIVITY option of the RETURN command indicates to CICS BTS that the current activity is about to complete, and that it will not be reactivated subsequently
<b>STARTBROWSE ACTIVITY</b>	Start a browse of the child activities of an activity, or of the descendant activities of a process
• Commands That Apply to Process and Activity Handling	
<b>CANCEL</b>	Cancel a process or an activity
<b>RESUME</b>	Resume a suspended activity or a process
<b>RUN</b>	Execute a CICS BTS process or activity synchronously or asynchronously
<b>SUSPEND</b>	Suspend a process or an activity
• Container Handling	
<b>DELETE CONTAINER</b>	Delete a named data-container
<b>ENDBROWSE CONTAINER</b>	End a browse of the data-containers associated with a process or an activity
<b>GET CONTAINER</b>	Retrieve data from a named data-container
<b>GETNEXT CONTAINER</b>	Browse the data-containers associated with a process or an activity
<b>INQUIRE CONTAINER</b>	Retrieve the attributes of a data-container
<b>PUT CONTAINER</b>	Save data in a named data-container
<b>STARTBROWSE CONTAINER</b>	Start a browse of the data-containers associated with a process or an activity
• Event Handling	
<b>ADD SUBEVENT</b>	Add a sub-event to a composite event
<b>DEFINE COMPOSITE EVENT</b>	Define a composite event
<b>DEFINE INPUT EVENT</b>	Define an input event
<b>DELETE EVENT</b>	Delete an event
<b>ENDBROWSE EVENT</b>	End a browse of the events known to an activity
<b>GETNEXT EVENT</b>	Browse the events known to an activity
<b>INQUIRE EVENT</b>	Retrieve the attributes of an event
<b>REMOVE SUBEVENT</b>	Remove a sub-event from a composite event
<b>RETRIEVE REATTACH EVENT</b>	Retrieve the name of an event that caused the current activity to be reattached
<b>RETRIEVE SUBEVENT</b>	Retrieve the name of the next sub-event in a composite event's sub-event queue
<b>STARTBROWSE EVENT</b>	Start a browse of events known to an activity
<b>TEST EVENT</b>	Test whether an event has fired

- Timer Handling
 

<b>CHECK TIMER</b>	Check the status of a timer
<b>DEFINE TIMER</b>	Define a timer
<b>DELETE TIMER</b>	Delete a timer
<b>FORCE TIMER</b>	Force early expiry of a timer
<b>INQUIRE TIMER</b>	Retrieve the attributes of a timer

For further details refer to Appendix B, “CICS BTS Commands” on page 265 and to the *CICS Business Transaction Services, SC34-5268* manual.

## 2.2.2 System Programming Interface

The following system programming interface (SPI) commands have been added or additional attributes are supported:

<b>CREATE PROCESSTYPE</b>	Define a PROCESSTYPE RDO object in the local CICS region
<b>DISCARD PROCESSTYPE</b>	Remove a PROCESSTYPE definition
<b>INQUIRE PROCESSTYPE</b>	Retrieve the attributes of a PROCESSTYPE
<b>INQUIRE TASK</b>	Retrieve the name of the CICS BTS process of which this task is a part, the process type of a CICS BTS process of which this task is a part, the name of the activity on behalf of which this task is executing, and the CICS-assigned identifier of this activity.
<b>SET PROCESSTYPE</b>	Change the attributes of a process type

For further details, refer to the *CICS Business Transaction Services, SC34-5268* manual.

## 2.2.3 CICS BTS Files

To ensure that processes can exist for several days and can even survive restarts, CICS BTS uses some new data sets. See Figure 7 on page 14 to get an overview.

- Repository

The CICS BTS repository data set holds state information about the CICS BTS processes as well as application data (container) that can be passed among activities or different instances of the same activity. One data set can be defined to hold information for all CICS BTS process types, or a separate data set for each process type can be used, or any combination of these two options can be used.

Notice that function shipping cannot be used to write to this data set. It is, therefore, recommended to define the repository data set in a system managed storage (SMS) storage class for record level sharing (RLS) data sets. If you are planning to execute CICS BTS activities that belong to the same process, and you want to distribute these activities to several CICS regions, you must access the repository in RLS mode. This is true even if you run all of the CICS regions in the same MVS image. Refer to 11.3, “CICS BTS Repository Data Set” on page 209 for a sample JCL to define a repository data set.

- Audit Log

The audit log for CICS BTS processes is functionally similar to the trace file for CICS transactions. It is optional per process and collects all audit data for a given process. However, the audit log entries for the same process can come from different CICS regions when a workload of processes is balanced in a CICSplex.

The audit log is a CICS user journal that is defined through a JOURNALMODEL resource definition with the TYPE(MVS) attribute. The records are written to an MVS system logger log stream, the name of which has been specified through the STREAMNAME attribute. If this log stream has been defined as a coupling facility log stream, the audit log that is associated with it can contain entries written from different CICS regions running on any MVS image within the Parallel Sysplex. Refer to 11.5, “Audit Log” on page 211 for more information about the audit log.

- Local Request Queue

This data set is needed even if you do not use CICS BTS. For CICS TS 1.3, CICS BTS is the only user of the local request queue. It must be defined as a recoverable VSAM KSDS, one data set for each CICS TS 1.3 region. It holds a copy of pending requests destined for this specific CICS region, thus, providing recoverability for such requests as CICS BTS timer events. Refer to 11.2, “Local Request Queue Data Set” on page 208 for a sample JCL to define a local request queue data set.

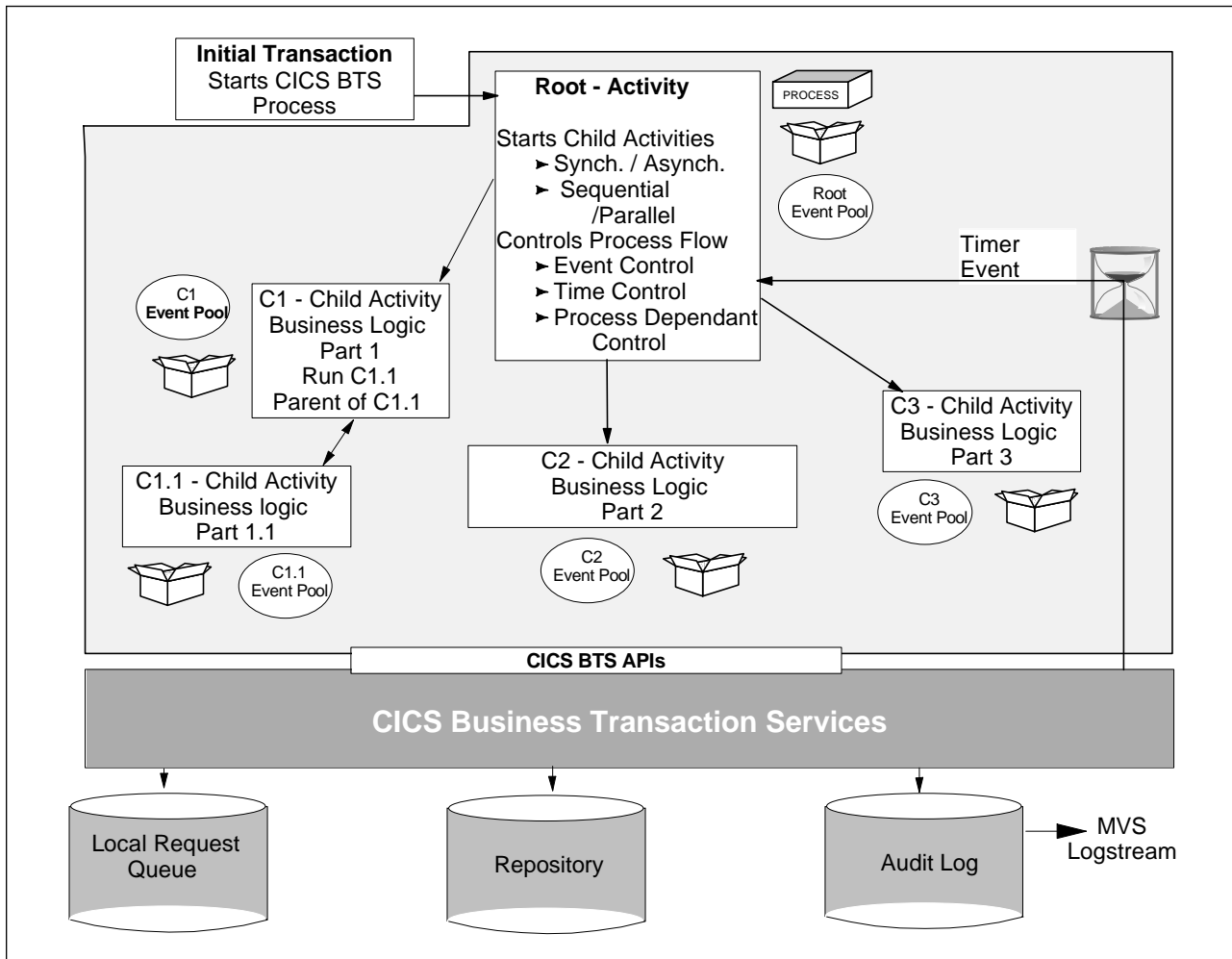


Figure 7. CICS BTS Components

## 2.3 A Business Transaction with and without CICS BTS

An activity that controls child activities can be compared with a terminal-oriented pseudo-conversational transaction where the transaction becomes activated with every terminal input. All data to execute the next program step has to be prepared. In a CICS BTS environment, for each activity, this role is assumed by its child activities. When a child activity has finished its work, the parent activity gets reactivated by the child's fired event. Figure 8 on page 15 and Figure 9 on page 16 illustrate the difference between a classic CICS application and a CICS BTS application solving the same task.



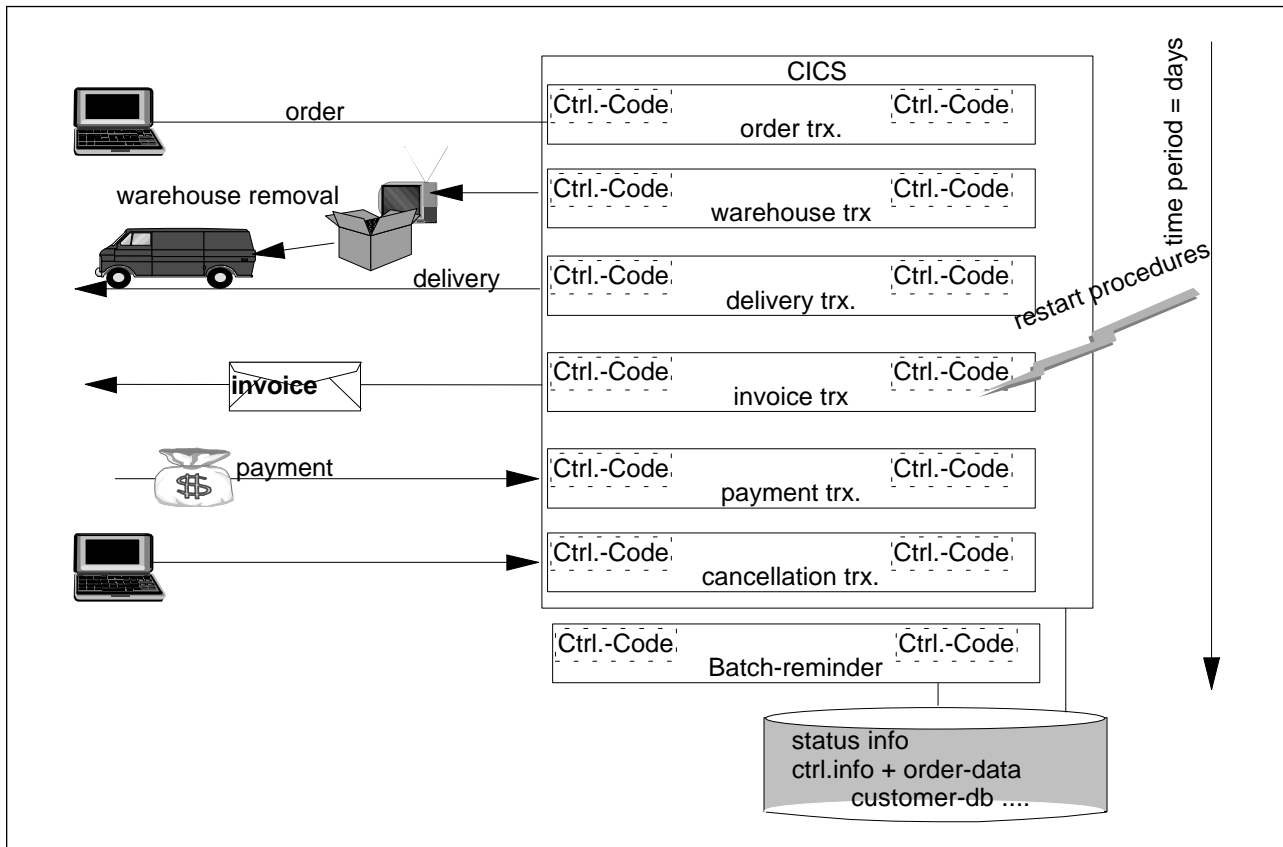


Figure 8. A Business Transaction without CICS BTS

Using classic CICS transactions, the example in Figure 8 shows that seven transactions are necessary to execute this business transaction. Each transaction is, in terms of CICS, independent from the one before and the next one. To synchronize the execution of the business steps needed to execute this business transaction, additional control code in each transaction is necessary. Status data has to be stored. A restart procedure must be developed separately if necessary.

The characteristics of this business transaction can be summarized as follows:

- Business transaction control logic is present in each CICS transaction.
- Code reuse is difficult or even impossible.
- Manual progress control may be necessary.
- The restart procedure is complicated.
- Status and business data are intermixed.

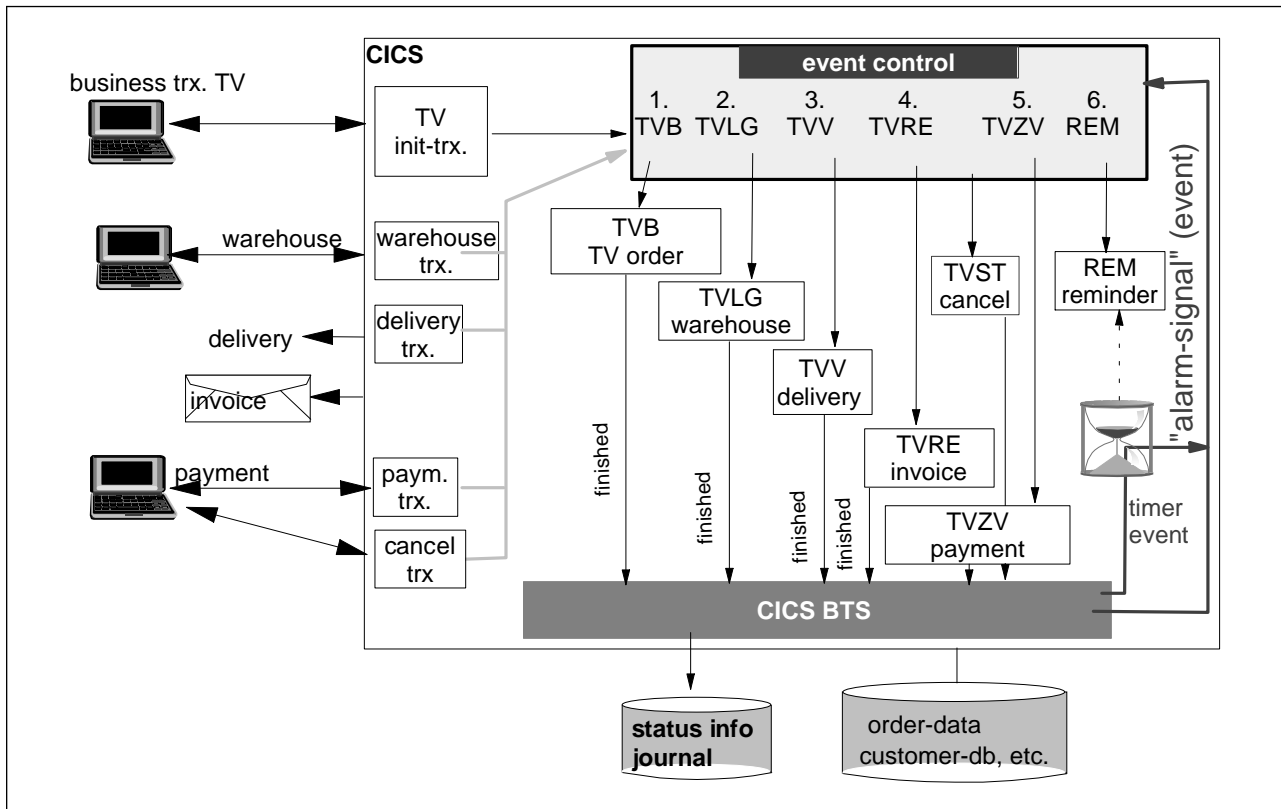


Figure 9. A Business Transaction with CICS BTS

Figure 9 shows the same business transaction as shown in Figure 8 on page 15 developed as a CICS BTS process. The business transaction control logic is implemented in the root activity. Within the child activities, there should only be business logic. All tasks of this business transaction should be developed as activities and be controlled by the CICS BTS event manager; therefore, no additional control logic needs to be included with the business logic.

The advantages can be summarized as follows:

- Business control logic is separated from the business logic.
- Reuse of code is possible.
- Progress control is done by CICS BTS.
- Automated restart is provided by CICS and CICS BTS.
- Control and status data are maintained by CICS BTS.

## 2.4 Process and Activity Execution

A process can be started by any traditional CICS transaction through a DEFINE PROCESS command followed by one of these new commands:

- RUN ACQPROCESS SYNCHRONOUS
- RUN ACQPROCESS ASYNCHRONOUS
- LINK ACQPROCESS

With DEFINE PROCESS, a new instance of a process is prepared. CICS BTS sets the mode of this instance to INITIAL. Nothing else happens until the defining program executes another command to start the process. With a RUN ACQPROCESS or LINK ACQPROCESS command, the process is executed.

Once a process has been started, an activity within it can be executed through a DEFINE ACTIVITY command followed by one of the following new commands:

- RUN ACTIVITY SYNCHRONOUS
- RUN ACTIVITY ASYNCHRONOUS
- LINK ACTIVITY

With DEFINE ACTIVITY, a new activity is defined to CICS BTS. It is used to add a child activity to the current activity. With a RUN ACTIVITY or LINK ACTIVITY command, the activity is executed.

## 2.5 User Syncpoints

A program that is running as an activation of a BTS process or activity cannot issue user syncpoints through EXEC CICS SYNCPOINT command. Only programs that are running outside any CICS BTS process may issue user syncpoints, for example:

- A top-level transaction that defines and runs a BTS process
- A program executing outside the BTS environment that acquires a process or activity by means of an ACQUIRE command

## 2.6 Activity Execution

Within a process, an activity can be executed synchronously or asynchronously with the requestor (parent activity).

Table 1 shows the different behaviors with the commands starting the execution of an activity within a process.

<b>Behavior</b>	<b>LINK ACTIVITY</b>	<b>RUN SYNC</b>	<b>RUN ASYNC</b>
Separate unit of work	No	Yes	Yes
Separate transaction ID	No	Yes	Yes
Child's result is backed out when parent rolls back	Yes	Yes	No
Parent's result is backed out when child rolls back	Yes	No	No
Failure isolation	No	Yes	Yes
CHECK completion after RUN command	Yes	Yes	No
Same task number	Yes	No	No

## 2.6.1 Synchronous Execution

Synchronous execution of an activity can be started by a LINK ACTIVITY or by a RUN ACTIVITY SYNCHRONOUS command (Figure 10 on page 19).

When the activity is activated with a LINK ACTIVITY command, the calling activity is suspended. CICS then executes the child activity. With a RETURN ENDACTIVITY command issued by the child activity, the parent activity is resumed. The child activity is invoked in the same unit of work (UOW) the parent activity is executed in. Therefore, all changes done by the child activity will be backed out if the parent activity fails after issuing the child activity. The child activity is executed using the transaction attributes (transaction ID and user ID) the parent activity uses.

When the activity is started using a RUN SYNCHRONOUS command, the calling activity and the transaction associated with it are suspended. The child activity is executed in a new UOW under its own transaction ID. In CICS BTS environment, this change is referred to as a *context-switch*. When the child activity issues a RETURN ENDACTIVITY command, the parent activity is resumed.

To discover whether the child activity was successful or not, the parent activity must issue a CHECK ACTIVITY command following either a LINK ACTIVITY command or a RUN ACTIVITY SYNCHRONOUS command. The CHECK ACTIVITY command deletes the fired completion event from the reattachment queue.

When the parent activity issues a RETURN ENDACTIVITY command, it completes normally, if there are no user events in the activity's event pool. If the ENDACTIVITY option is not specified on an EXEC CICS RETURN command and there are user events in the event pool but no events on the reattachment queue, the activity becomes dormant until a reattachment event occurs.

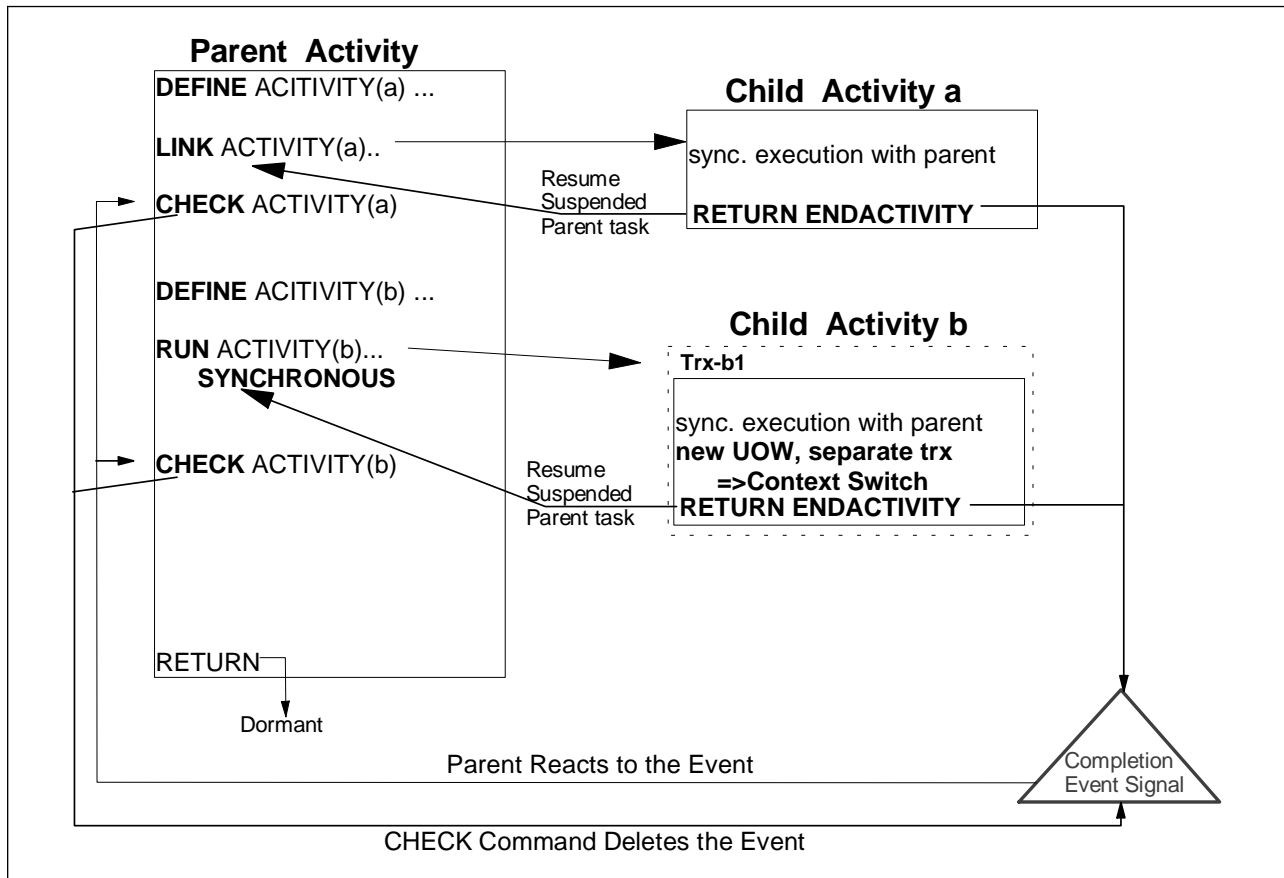


Figure 10. Synchronous Execution of Activities

With a LINK command, the child activity is executed as part of the parent activity's UOW. With a RUN command, a separate UOW is created, which means a context-switch takes place. A context-switch can be defined as an activation of a process or activity in a separate UOW from the requestor.

## 2.6.2 Asynchronous Execution

With a RUN ACTIVITY ASYNCHRONOUS command, the activity is executed asynchronously in a separate UOW (see Figure 11 on page 20). This means that the parent and child activities execute completely independently from each other. The RUN ACTIVITY ASYNCHRONOUS command does not directly start the child activity. Execution of the child activity in an asynchronously running transaction is deferred until the calling transaction executes a RETURN command.

When the child activity ends, its completion event is fired. CICS BTS reactivates the corresponding parent activity, presenting the child's completion event when the RETRIEVE REATTACH command is executed. The parent activity must then investigate how the child activity completed by issuing the CHECK ACTIVITY command.

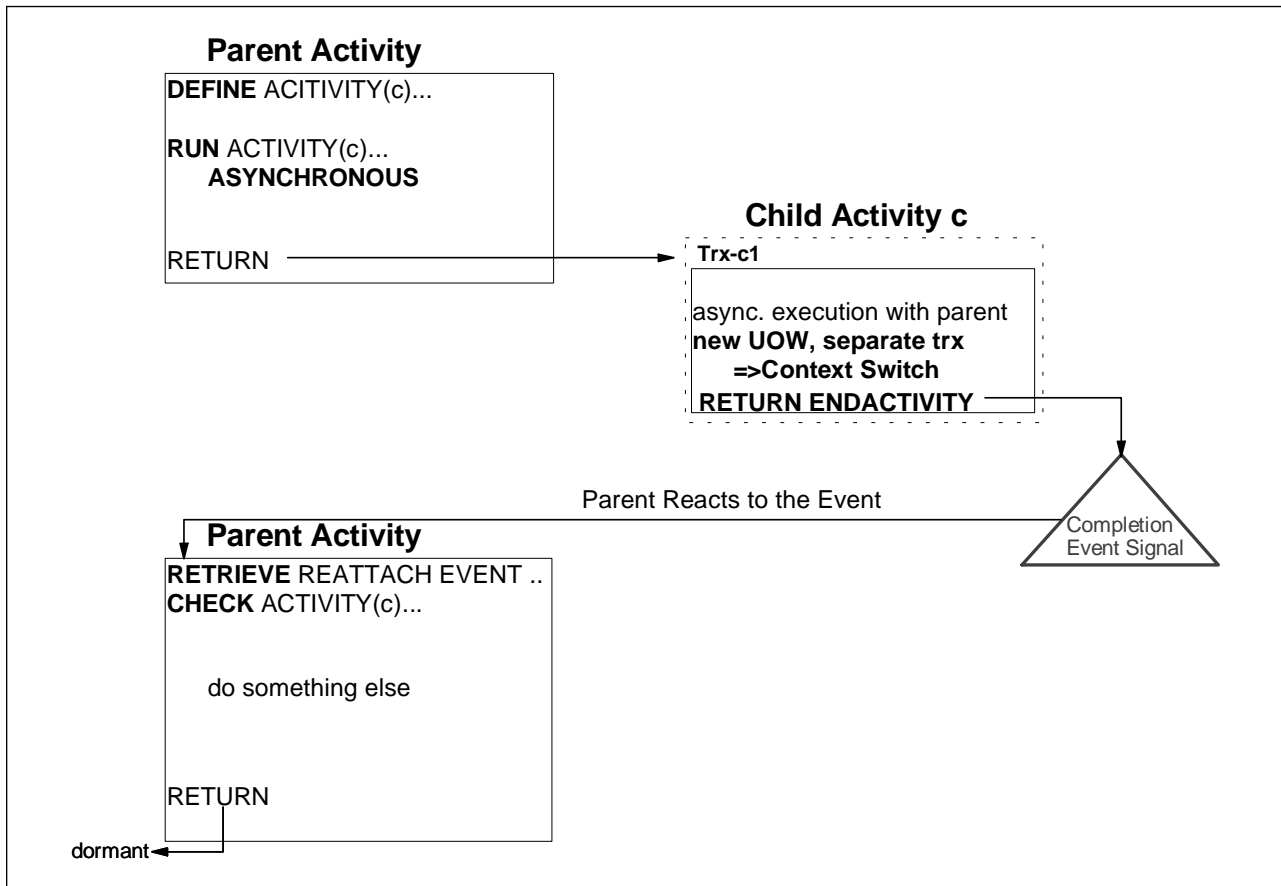


Figure 11. Asynchronous Execution of Activities

## 2.7 Process Flow

A process is always created by a CICS program within a transaction. This program gives the process its name. This name is used to identify the process on subsequent commands that act upon it. The process name is up to 36 characters long. This name has to be unique within the process type to which the process belongs.

Activities use an activity name up to 16 characters long. This name needs to be unique within the set of child activities defined by its parent. A process's first activity always has the CICS-assigned name DFHROOT.

Since the execution units of a process are its activities, and each activation of an activity needs an event, the first activation is triggered by the system event DFHINITIAL. This event is supplied by CICS BTS after the first RUN or LINK command is issued. Each activation caused by a RUN command is executed by a new CICS transaction in a separate UOW. When the execution is caused by a LINK command, the process is executed within the requestor's UOW.

An activation is simply a transaction executing the program or programs associated with an activity. An activation ends when the top-level program issues the EXEC CICS RETURN command and control is returned to CICS. In general, the work of an activity is carried out in a sequence of activations, so when an activation ends, it might not mean the completion of the activity.

When an activity is complete, the termination of the last activation will cause CICS to signal the completion to the activity's parent by firing the child's completion event.

When no more processing steps are necessary (this means that the current activation is the last) and the activity is not to be reactivated for further events, it issues an EXEC CICS RETURN ENDACTIVITY command.

The activity that was activated with the RUN/LINK ACQPROCESS commands is called the root activity. As mentioned before, this activity is always started with the event DFHINITIAL. Usually the root activity prepares the control of the process-flow. To do this, it defines timers, events, and starts activities or even another process. If any user input (user-related activity) is necessary, the application has to be designed to provide this information to the transaction. This can be done by:

- Executing the root activity with an input event.
- Writing to the process' or root activity's container. To do this, it is necessary to acquire the process beforehand. After a RUN or LINK command is issued, the process can access the input data in a container.
- Writing to an activity's container and specifying an input-event with the RUN or LINK command.

Figure 12 gives a brief overview of the process flow in a simple process, such as the sample SALES process.

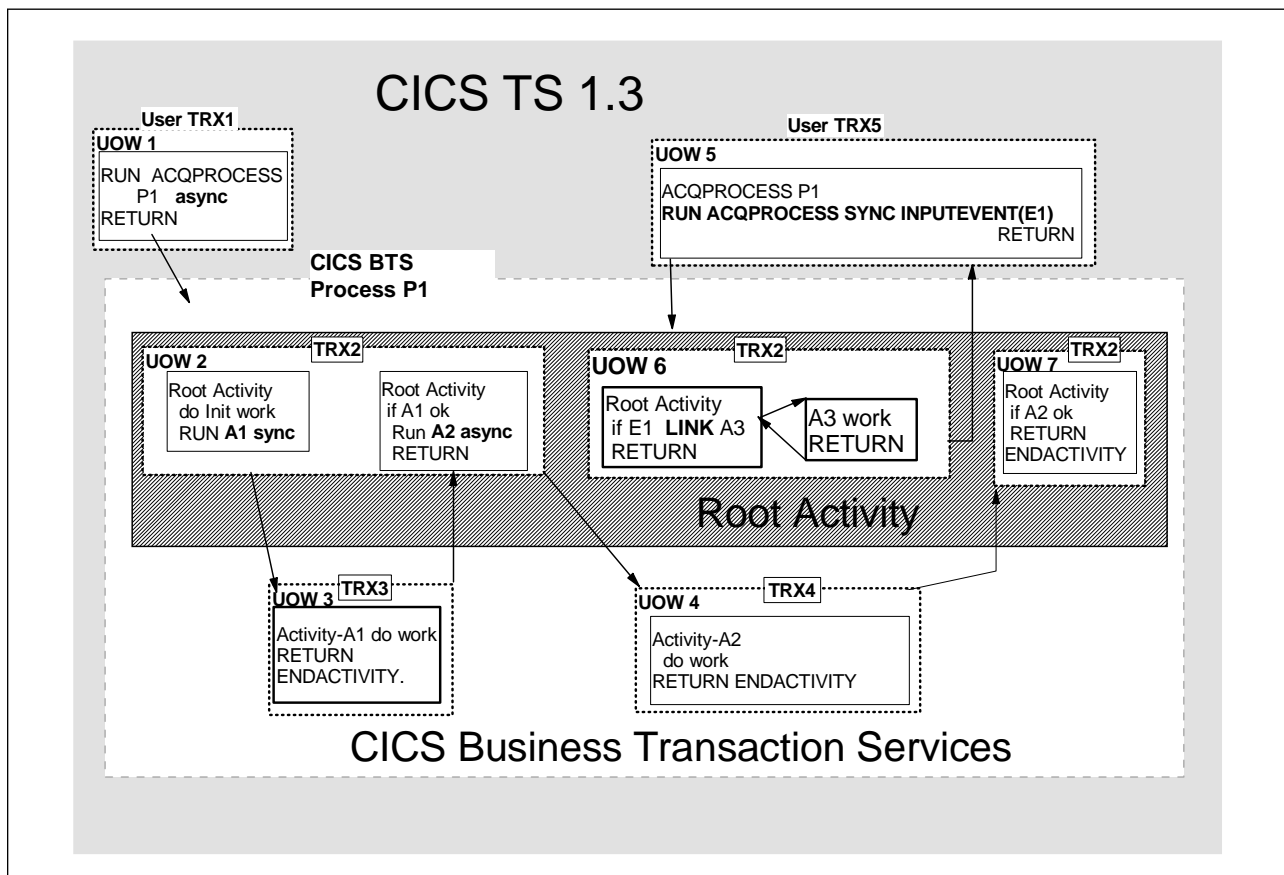


Figure 12. A Simple Process Flow

This is the flow sequence of the SALES process:

1. User transaction TRX1 (UOW1) is a classic CICS transaction. It defines the process P1 and starts the process with the RUN ACQPROCESS ASYNC command. The process is executed asynchronously in a new UOW (UOW2). The execution starts when TRX1 commits (by RETURN or SYNCPOINT, without ROLLBACK). At this point, the process is activated.
2. CICS BTS gives control to the root activity with a system event DFHINITIAL. The root activity runs as transaction TRX2. The root activity creates and then activates activity A1, executed in UOW3 with the transaction ID TRX3. This activity is started synchronously with a RUN command. Therefore, transaction TRX2 and its UOW (UOW2) are suspended.
3. Activity A1 does its work. When it has finished, a RETURN ENDACTIVITY command is issued. This causes CICS BTS to mark the event A1 fired. On completion of this event, TRX2 becomes resumed.
4. If A1 has completed OK, the root activity chooses to run activity A2 asynchronously. The root activity after issuing the RETURN command becomes dormant. CICS BTS starts activity A2 using UOW4 with a transaction-id TRX4 as required by the root activity's DEFINE ACTIVITY command. Activity A2 is started with the system event DFHINITIAL.
5. In the meantime, a non-CICS BTS transaction TRX5 has acquired and activated process P1 with the input event E1. Executing the RUN SYNC command the root activity is executed synchronously. The root activity starts with the transaction ID TRX2 in UOW6. It is reattached with the event E1. It evaluates the event and defines and links to activity A3. On the last RETURN command in UOW6, control is given back to UOW5. The units of work UOW5 and UOW6 become committed when TRX5 executes an EXEC CICS RETURN. The root activity's status becomes dormant again.
6. Activity A2 runs until a RETURN command is executed. When transaction TRX4 commits, CICS BTS fires event A2. When this event fires, the activity's parent (the root activity) is activated.
7. UOW7 executes the root activity when it reacts to the fired event A2. This means the status of the root activity becomes active again. With a RETURN ENDACTIVITY command, the UOW7 becomes committed, and process P1 completes, since it has no outstanding unfired events.

---

## 2.8 Rules to Access a Process or Activity

When a program needs access to a process or an activity, it has to obey the following rules:

1. A program can acquire only one activity within the same UOW. The activity remains acquired until the next syncpoint. This means, for example, that a program:
  - Cannot issue both a DEFINE PROCESS and an ACQUIRE PROCESS command within the same UOW.
  - Cannot issue both an ACQUIRE PROCESS and an ACQUIRE ACTIVITYID command within the same UOW. That is, it can acquire either a descendant activity or a root activity but not both.
2. If a program is executing as an activation of an activity, it cannot:



- Acquire an activity in the same process as itself; that is, it cannot issue a `ACQUIRE PROCESS` for the current process.
  - Use a `LINK` command to activate the activity that it has acquired.
3. An acquired activity's process is accessible in the same way as the activity itself can access it. Thus, if the acquired activity is a descendant activity:
- Its process' containers may be read but not updated.
  - The process may not be the subject of any command (`RUN`, `LINK`, `SUSPEND`, `RESET`) that directly manipulates the process or its root activity.

Conversely, if the acquired activity is a root activity:

- Its process' containers may be both read and updated.
  - The process may be the subject of commands such as `RUN`, `LINK`, `SUSPEND`, `RESET`, or `RESUME`. The `ACQPROCESS` keyword on the command identifies the subject process as the one the program issuing the command acquired in the current UOW.
4. Within a process, direct connections are established only between a parent and its child. Thus, a `DELETE ACTIVITY` command can only be issued by the parent activity. A `DELETE CONTAINER` and `DELETE EVENT` command can be issued by the activity itself or by its parent. A `DELETE TIMER` command can only be issued by the activity that defined the timer.

---

## 2.9 Activity Modes Appearing During Process Execution

During the lifecycle of a process, all activities used are in a certain mode at any one time. Figure 13 on page 24 shows the dependencies.

To see how the child finished, the parent has to execute a `CHECK ACTIVITY` command. The `CVDA` value will reflect one of the following activity statuses:

- **ACTIVE:**

An instance of an activity is running.

- **CANCELLING:**

`CICS BTS` is waiting to cancel an activity. Possibly an activity cannot be cancelled because a UOW associated with the activity's child holds retained locks.

- **COMPLETE:**

The activity has completed either successfully or unsuccessfully. The value returned on the `COMPSTATUS` option of a `CHECK ACTIVITY` command tells you how it completed.

- **DORMANT:**

The activity is waiting for an event to fire and, thus, cause its next activation.

- **INITIAL:**

No `RUN` or `LINK` command has yet been issued against the activity, or the activity has been reset to its initial state by means of a `RESET ACTIVITY` command.



containers, while P2 and P3 use the ACQUIRE ACTIVITYID command to access the containers of the child activities A2 and A4.

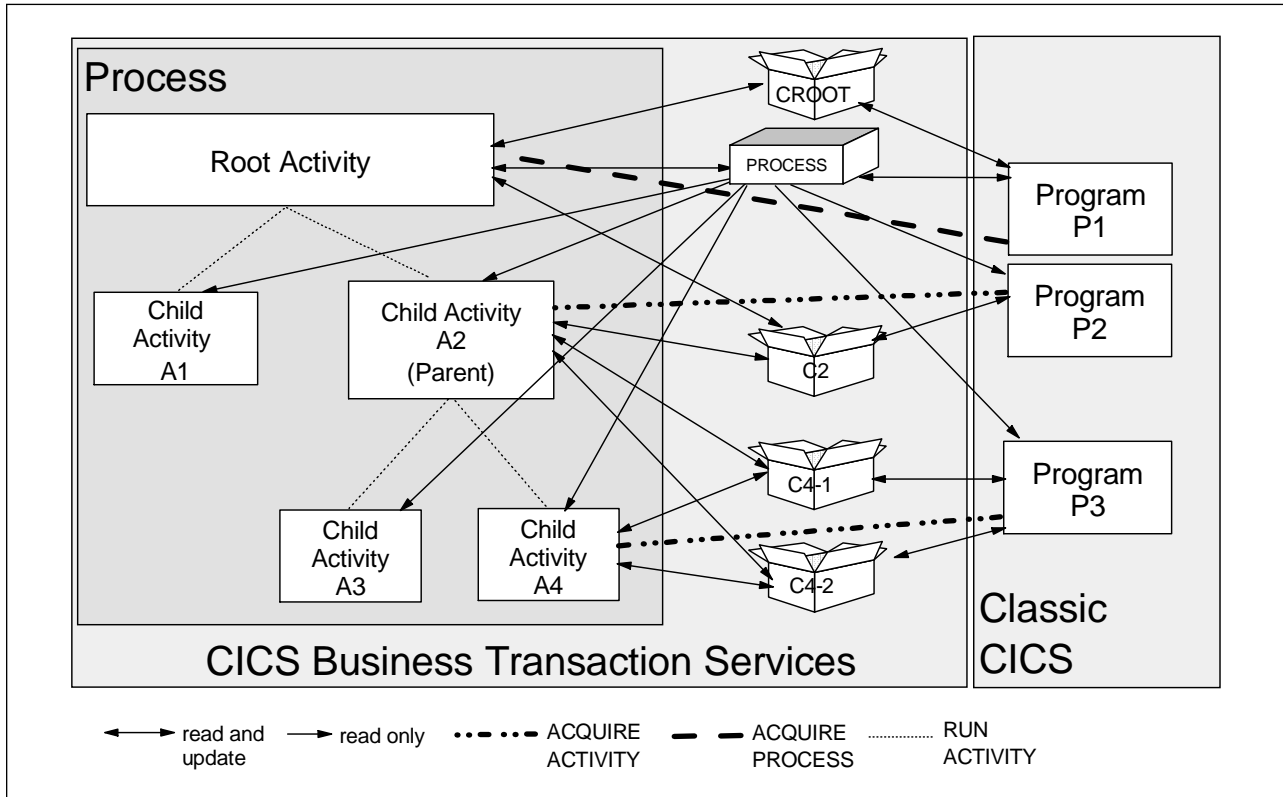


Figure 14. Availability of Containers

We can see from Figure 14 that:

- The root activity has access to the process container, its own activity container (CROOT), and to the activity container C2. It cannot access containers C4-1 or C4-2 as these do not belong to the root activity or its immediate descendants.
- Program P1 (after ACQUIRE PROCESS) has access to the process container and to the CROOT container.
- Only the root activity and program P1 that acquired the process have write access to the process container. All other activities have only read access.
- Activity A2 has access to containers C2, C4-1, and C4-2.
- Program P2 (after ACQUIRE ACTIVITY) has access to container C2. Notice that this program does not have the same access to A4's containers as activity A2, which is parent of A4.
- Activity A4 and program P3 (after ACQUIRE ACTIVITY) have access to containers C4-1 and C4-2.

The process containers and the root activity's containers can be created before the RUN ACQPROCESS command is executed. They can also be created by the root activity but not by any child activity.

The containers are maintained in the CICS BTS repository and are restored at system restart.

## 2.11 Events

Events are used within CICS BTS to handle progress signals in a process. Activities must be designed to support the process event logic.

An event informs an activity that an action is required or its child has completed. Each event is named by a DEFINE command. An event's state is shown by a boolean value (FIRED or NOTFIRED). When an activity is reattached after an event has fired, it can discover the event that caused it to be reattached by issuing the RETRIEVE REATTACH EVENT command.

Each activity has an event pool associated with it (see Figure 15). An activity's event pool is initialized when the activity is created and deleted when the activity is deleted. Event related commands, such as DEFINE INPUT EVENT and DEFINE COMPOSITE EVENT operate on the event pool associated with the current activity. The event pool contains the set of events recognized by an activity.

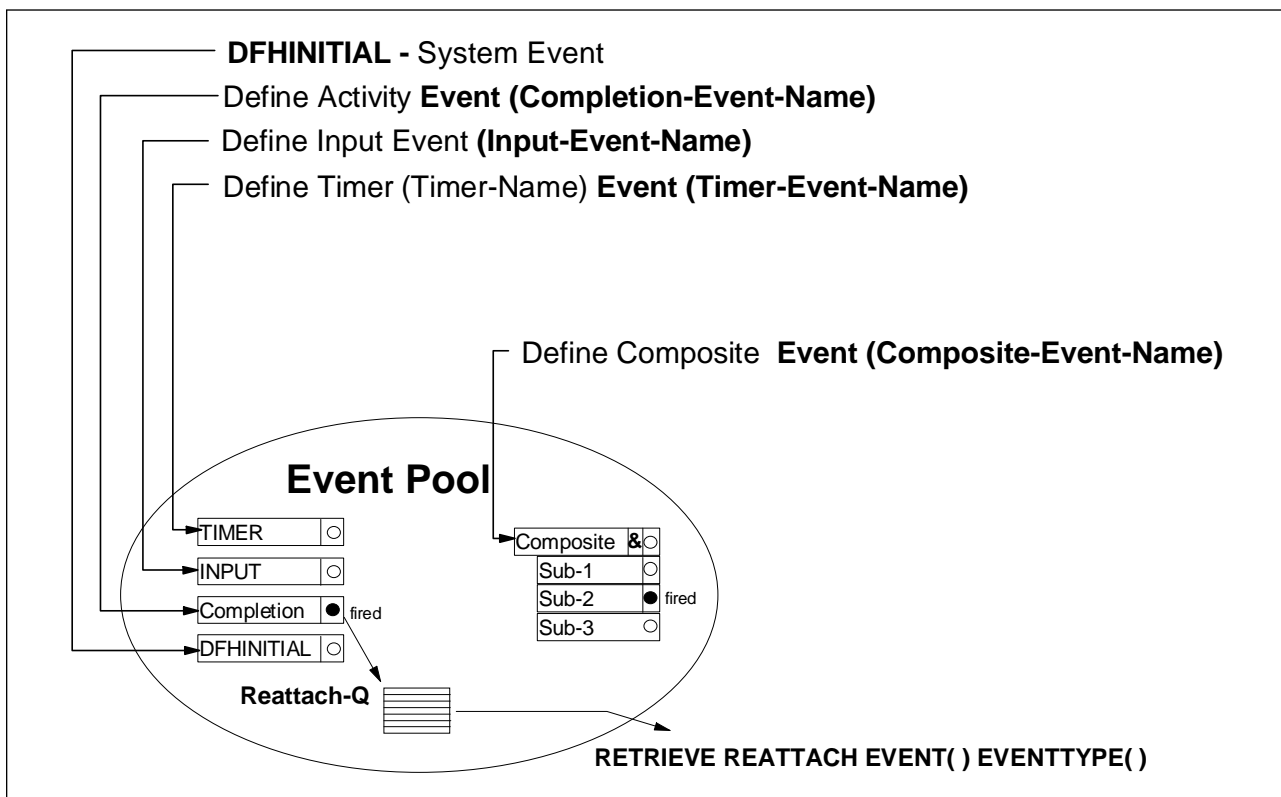


Figure 15. Event Handling

Events are defined in a number of ways

- DFHINITIAL is the system event CICS BTS uses when an activity is executed the first time after it has been defined. It can only be defined by CICS.
- The DEFINE ACTIVITY(A1) EVENT(E1) command creates a completion event for the activity(A1). The event(E1) is set to FIRED in the parent's event pool on completion of the child activity A1.
- With DEFINE INPUT EVENT(E2) command, this event type is added to the event pool. Nothing else happens until a RUN ACTIVITY(A1) INPUTEVENT command is issued by another program. When this happens, the event is fired.

- DEFINE TIMER() EVENT(E3) sets a timer and adds a timer event to the event pool. The activity that defined the timer becomes reattached when the event fires.
- A composite event consists of zero or more subevents. It has a boolean operator AND or OR, which determines when it fires. With the AND operator, the composite event fires when all its subevents have fired. With the OR operator, the composite event fires when any one of its subevents have fired. See 2.11.2, “Composite Events” on page 28 for further details.

When an event's state changes to FIRED, a new entry is added to the reattach queue. Processing this queue CICS BTS reinvokes the activity.

CICS BTS distinguishes between atomic and composite events.

## 2.11.1 Atomic Events

An atomic event is a single occurrence. It may happen either under or outside the control of CICS BTS. Four different types of events are known.

### 2.11.1.1 System Events

The only type of CICS BTS system event is DFHINITIAL, and it is defined by CICS BTS.

The first activation of an activity is always given the event name DFHINITIAL. It cannot be included in composite events.

All other types of events are referred to as user-defined events because they are defined by the CICS BTS application programmer.

### 2.11.1.2 Activity Completion Events

The completion of a child activity causes the activity completion event to fire. The event is named by the EVENT option on the DEFINE ACTIVITY command. If it is not specified, the completion event is given the same name as the activity itself.

### 2.11.1.3 Input Events

Input events tell activities why they are being asked to run. They must have been defined to the activity before the activity can be reattached with an input event. A RUN/LINK ACTIVITY INPUTEVENT command delivers an input event to the activity. The command starting the activity's execution must name the event except when the activity is started the first time in this process. An activity typically defines its input events during its initial phase. However, depending on the result of child activities, this can be done during any activation.

### 2.11.1.4 Timer Events

A timer event is automatically defined with the DEFINE TIMER command. It fires when the timer expires. It is possible to name this event using the EVENT option of the DEFINE TIMER command. If not, the event is given the same name as the timer itself.

## 2.11.2 Composite Events

A composite event is a collection of atomic events. It is formed from zero or more user defined events. These user defined events are referred to as sub-events when included in a composite event. A system event cannot become part of a composite event.

A composite event is defined by using the DEFINE COMPOSITE EVENT() AND|OR command. Sub-events can be defined as parameters of this command or with the ADD SUBEVENT command. Up to eight sub-events can be defined with a single DEFINE command. Further sub-events have to be defined with the ADD SUBEVENT command.

With the parameter AND or OR, the logical use of the composite event is specified. This parameter is used as a boolean operator. If the OR operator is used, the composite event fires when any of the sub-events fires. An AND-defined composite event changes its state to FIRED when all sub-events have fired.

To manage sub-events, CICS BTS uses the sub-event queue. All fired sub-events are placed in this queue. From there, they can be retrieved using the RETRIEVE SUBEVENT command for each entry. CICS BTS creates a sub-event queue entry for each sub-event.

Rules:

- A composite event must use either OR or AND (not both).
- The following cannot be used as sub-events:
  - Composite events
  - Sub-events of other composite events
  - System events
- An empty composite event is always:
  - True (FIRED) when the AND parameter is used.
  - False (NOTFIRED) when the OR parameter is used.

## 2.11.3 Deleting Events

You can delete an event (that is, discard both the event and its name). If the event is a sub-event, the value of the composite event will be that of its predicate after the sub-event has been removed from the predicate's boolean expression.

The command you use to delete an event depends on the type of event to be deleted.

- You cannot delete system events.
- An activity completion event is implicitly deleted when a response from the completed activity has been acknowledged by a CHECK ACTIVITY command issued by the activity's parent, or when a DELETE ACTIVITY command is issued. The CHECK ACTIVITY command returns the completion status of a child activity as a CVDA value. It might be a good idea to save this status for further use, because the event is deleted from the event pool and cannot be rechecked.
- To delete an input event explicitly, use the DELETE EVENT command.

- A timer event is implicitly deleted if its associated timer has expired and a CHECK TIMER command is issued by the activity that owns it, or when a DELETE TIMER command is issued.
- To delete a composite event explicitly, use the DELETE EVENT command. Note that deleting a composite event does not delete the composite's sub-events.
- If an activity program issues a RETURN ENDACTIVITY command, CICS automatically deletes all user events, other than activity completion events, in the activity's event pool. Completion events must always be deleted by means of the CHECK ACTIVITY or DELETE ACTIVITY commands.

Table 2 summarizes the commands that can be used to delete each type of event.

Event type	Deletion commands
System	Cannot be deleted
Activity completion	1. CHECK ACTIVITY (if the activity has completed) 2. DELETE ACTIVITY
Input	1. DELETE EVENT 2. RETURN ENDACTIVITY
Timer	1. CHECK TIMER (if the timer has expired) 2. DELETE TIMER 3. RETURN ENDACTIVITY
Composite	1. DELETE EVENT 2. RETURN ENDACTIVITY

Before it can complete normally, an activity must have deleted all the activity completion events in its event pool. That is, it must have dealt with all its child activities.

## 2.11.4 Handling Events

When an event (except sub-events) fires, the activity is reattached. This is true for composite events and atomic events not used in a composite event. These events are called reattachment events.

A sub-event never causes the reattachment of an activity. Reattachment of the associated activity depends on the parameter (AND or OR) of the composite event the sub-event is added to.

All fired events except sub-events are placed in the reattachment queue of the activity. This means it is possible that more than one event could fire. CICS BTS activates the activity once. The events can be evaluated in a loop. If not, CICS BTS starts the activity again when the activity has executed a RETURN command.

It is possible to deal with several events in one activation. During application design, the fact that SYNCPOINT processing in CICS BTS is only possible with a RETURN command must be considered.

An activity must use the RETRIEVE REATTACH EVENT command to discover the event that caused reattachment. Figure 16 on page 30 illustrates how to react on an atomic event.

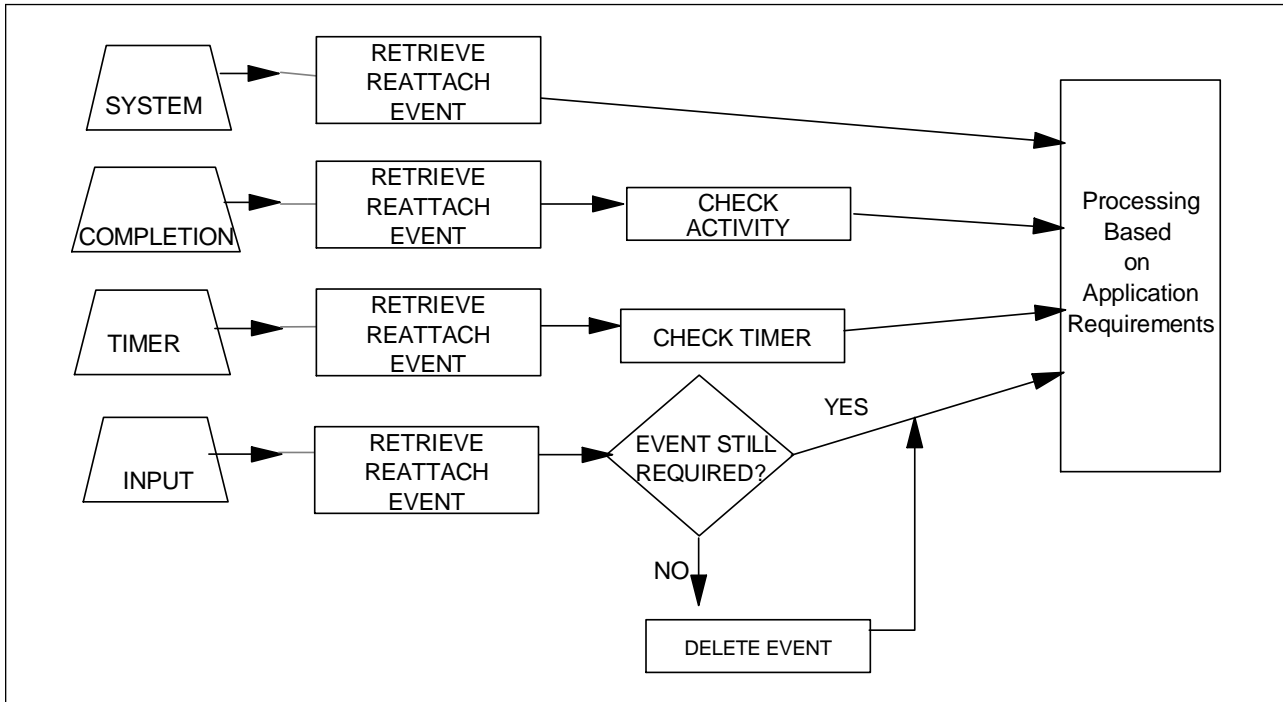


Figure 16. Event Handling with Atomic Events

CICS BTS resets all atomic events to NOTFIRED after execution of the activity that retrieved the event. CICS BTS does not reset composite events to NOTFIRED until the predicate becomes false. The activity can do this by executing the RETRIEVE SUBEVENT command. Retrieving from the sub-event queue causes CICS BTS to re-evaluate the state of the composite event.

Issuing a CHECK ACTIVITY command provides the information to the parent as to whether the child completed normally. CICS BTS then deletes the completion event from the event pool. The CHECK TIMER command can be used in the same way for a timer. This evaluates whether the timer expired normally or was forced. Figure 17 on page 31 illustrates how to deal with composite events in an activity.



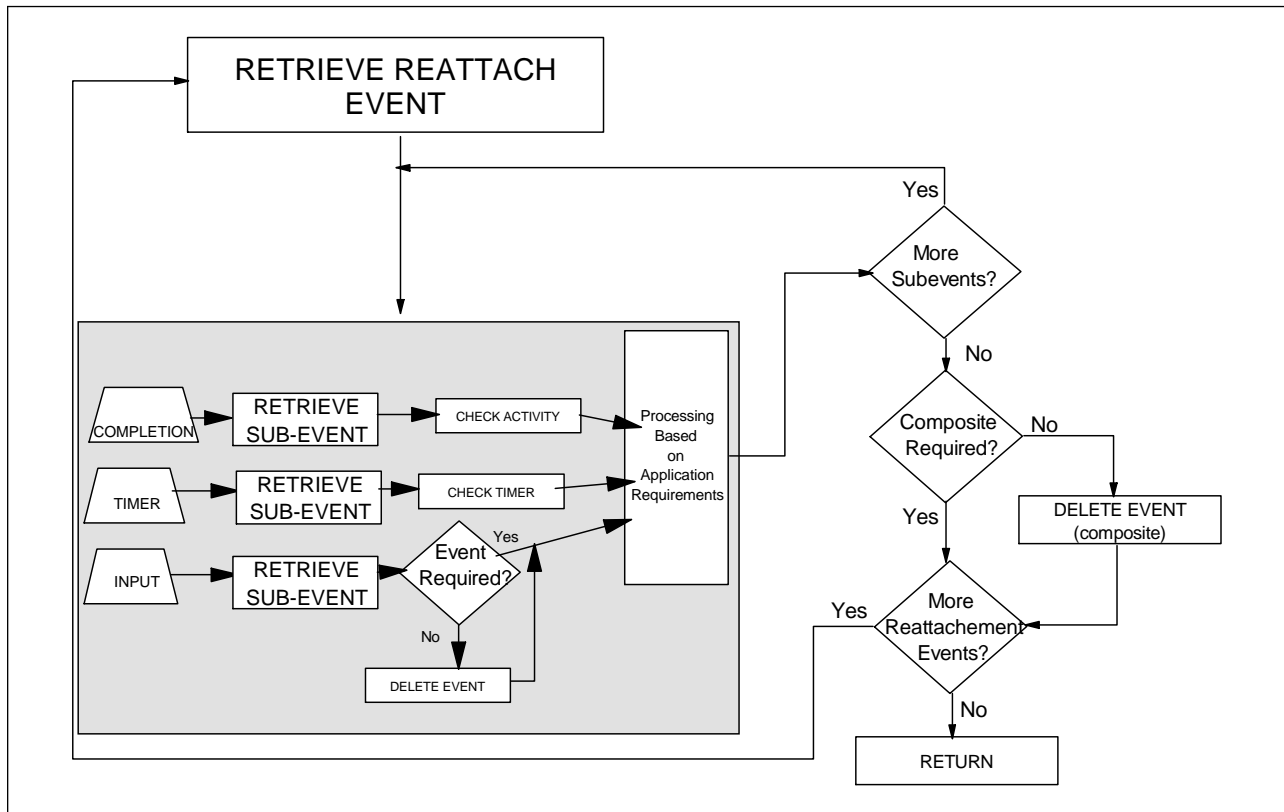


Figure 17. Event Handling with Composite Events

## 2.12 Name Scope Summary

In this section we provide a summary of name scope and conventions for various BTS entities.

- Process names
  - Process names are unique within the process type.
  - They can be reused once the process has completed.
  - Process names are of 1 to 36 character length.
- Activity names
  - Activity names are unique within the parent activity.
  - They are of 1 to 16 character length.
- Activity ID
  - Each activity has an ID which is unique for all time.
  - An activity ID is a 52 character identifier assigned by CICS.
- Event names
  - Event names are unique within the owning activity.
  - They are of 1 to 16 character length.
- Data-containers
  - Data-container names are unique within the owning activity.

- They are of 1 to 16 character length.
- Process containers are accessible by all activities in a process.

## 2.13 CBAM Transaction

Now let us see what observations can be made from the new CICS-supplied CBAM transaction. The screen captures below apply to the CICS-provided SALES sample application.

When you enter CBAM on a CICS screen, you get a list of the process types known to this CICS region (see Figure 18).

CBAM			
Processtype	File	Status	Auditlevel
ORDER	PTYPSALE	Enabled	Full

Figure 18. The CBAM List of Current Process Types

If you place the cursor alongside the Processtype (ORDER in this sample) and press Enter, CBAM displays a list of the currently known processes of the selected type. Figure 19 shows the two processes that are currently in the system including their status. The term "currently in the system" only means that CICS BTS knows about this process and has some state information stored in the repository, but the process has not completed. It does not necessarily mean that there is an activation that you would see with the CEMT INQUIRE TASK command. Once a process has completed, it is removed from the repository, and the only proof that it has run would be an audit log (or some other trace of course).

CBAM		Processtype ORDER		
Process	Mode	Comp	Susp	
SALES333111	Dormant	Incomplete	No	
SALES500666	Dormant	Incomplete	No	

Figure 19. Current Processes of a Selected Type

Placing the cursor beside a process (for example, process SALES333111), and pressing Enter, you get the details of the selected process as shown in Figure 20 on page 33.

CBAM	Process SALES333111	Processtype ORDER		
Activity		Mode	Comp	Susp
DFHROOT		Dormant	Incomplete	No
INVOICE-BUILD		Complete	Normal	No
DELIV-NOTE		Complete	Normal	No
STOCK-CHECK		Complete	Normal	No
CREDIT-CHECK		Complete	Normal	No

Figure 20. Details of Process SALES333111

CBAM displays the activities that have been defined so far in this selected process and the current status and completion information of the activities. The REMINDER activity is not shown yet because it is not yet defined.

Select the root activity, for example, and press Enter to display the details of the root activity (Figure 21).

CBAM	Process SALES333111	Processtype ORDER	
Activity	DFHROOT		
Program	DFH0SAL2		
Transid	SALE		
Userid	CICSUSER		
Containers			
Events			
Timers			

Figure 21. Details of the Root Activity

This screen shows the program that implements the activity, the transaction ID it uses, and the user ID under which the activity runs. It also shows the other resources an activity may have, namely containers, events, and timers.

Each of these resources can now be selected to see whether the respective resource is used and the current status of each resource. Figure 22 shows that there is a 4-byte container in use.

CBAM	Process SALES333111	Processtype ORDER	
Activity	DFHROOT		
Container		Data length	
TEMP-CONTAINER		4	

Figure 22. Resource Display (Container) of an Activity

Figure 23 on page 34 shows the events known by the root activity.

CBAM	Process SALES333111	Processtype ORDER		
Activity DFHROOT				
Event	Type	Fired	Composite	Timer
DELETE-TIMER	Input	No		
DFHINITIAL	System	No		
SEND-PAY-REM	Timer	No		REM-TIMER

Figure 23. Events Known by the Root Activity

We have three events here, all with a status of NOTFIRED. This status can indicate that:

1. The event has not yet occurred for this activity instance.
- or
2. The event has been reset.

We know that the DFHINITIAL event must have occurred once when the process was started. So, here you see that the initial event is reset when the activity goes dormant. You also see that the SEND-PAY-REM event is a timer event, and the timer is called REM-TIMER. REM-TIMER is set to a specific value, and, when it expires, the SEND-PAY-REM event is fired.

Selecting the timer event gives you additional information about the timer as shown in Figure 24.

CBAM	Process SALES333111	Processtype ORDER		
Activity DFHROOT				
Timer	Status	Event	Date	Time
REM-TIMER	Unexpired	SEND-PAY-REM	11131998	133307

Figure 24. Timer Event Details

## 2.14 Audit Logging

With the audit log feature, it is possible to record the progress of CICS BTS business transactions and to diagnose problems in programs that are part of a CICS BTS business transaction. To do so, CICS provides BTS audit points. These can be compared with the trace points you are used to work with in classic CICS. The CICS BTS audit log can be described as follows:

- The audit log is written to a CICS journal, which resides on an MVS log stream.
- The records written to the audit log for a process can extend to days, weeks, or even months.

- A process can be executed in several CICS regions in a sysplex. Therefore, the audit log file can be defined per process type. If defined, it collects the audit entries from all regions executing this process.

To write entries to the audit log, four auditing levels are available. The level to be used is specified with the process definition in DFHCSD. The level can be set to:

- None
- Process-level
- Activity-level
- Full

For a description of each level, see 11.5, “Audit Log” on page 211.



---

## Chapter 3. Base API: SALES Application

CICS TS 1.3 provides a sample CICS BTS *SALES* application.

We use this application to explain the base CICS BTS API. We first give a brief overview of what the application does. Then, we provide information about some of the API commands.

Each event is explained with a pictorial overview and with a pseudo-code listing. Then we give an explanation of the commands used. The commands are explained in the context of use. For further explanations of each command, refer to *CICS Business Transaction Services, SC34-5268*.

Notice that the sample application provided with CICS TS 1.3 is not exactly the same as the application described in the manual. The manual expands the sample application in order to explain some of the various components of CICS BTS.

---

### 3.1 Overview

The sales application implements one 3270 transaction to activate the process and another to provide payment information. The process itself implements a stock and credit check, delivery, invoice, and reminder preparation, as well as supervision. The application is developed in eight actions.

- An initial MENU to request the order
- Root activity
- Credit check
- Stock check
- Delivery
- Invoice
- Reminder (written up to three times)
- 3270-payment transaction

The sales transaction will be started by a terminal user initiating a classic CICS transaction that will display a map into which order data can be entered. After the user enters the data, the process is started synchronously. This means that the 3270 application waits for the EXEC CICS RETURN. CICS commits the (four) UOWs with the EXEC CICS RETURN of the terminal application.

---

## 3.2 Transactions and Programs

Table 3 shows the transaction IDs and programs used in the CICS BTS business transaction.

Activity	Transid	Program	Comments
-	MENU	DFH0SAL0 DFH0SAL1	A traditional CICS transaction to invoke the CICS BTS process
DFHROOT	SALE	DFH0SAL2	BTS root activity that manages the child activities
CREDIT	RED1	DFH0RED1	Check credit activity
STOCK	STOC	DFH0STOC	Check stock activity
DELIVERY	DEL1	DFH0DEL1	Delivery activity
INVOICE	INV1	DFH0INV1	Invoice activity
REMINDER	REM1	DFH0REM1	Reminder activity
-	PAYM	DFH0PAY1	A traditional CICS transaction to provide an input event

---

## 3.3 What This Application Introduces

The sales application introduces a brief insight of the new CICS BTS application programming technique.

### 3.3.1 Coverage of API

The design uses the following API commands:

- ACQUIRE PROCESS(PROCESS-NAME)
- ADD SUBEVENT(SUBEVENT-NAME)
- ASSIGN PROCESS(PROCESS-NAME)
- CHECK ACTIVITY(ACTIVITY-NAME)
- DEFINE ACTIVITY(ACTIVITY-NAME)
- DEFINE COMPOSITE EVENT(COMPOSITE-EVENT-NAME)
- DEFINE INPUT EVENT(EVENT-NAME)
- DEFINE PROCESS(PROCESS-NAME)
- DEFINE TIMER(TIMER-NAME)
- DELETE CONTAINER(CONTAINER-NAME)
- DELETE EVENT(EVENT-NAME)
- DELETE TIMER(TIMER-NAME)
- GET CONTAINER(CONTAINER-NAME)
- PUT CONTAINER(CONTAINER-NAME)
- RESET ACTIVITY(ACTIVITY-NAME)
- RETRIEVE REATTACH EVENT(EVENT-NAME)



- RETRIEVE SUBEVENT(SUB-EVENT-TYPE)
- RETURN ENDACTIVITY
- RUN ACQPROCESS
- RUN ACTIVITY(ACTIVITY-NAME)

### 3.4 The Initiating Transaction (MENU)

Typically, a process is activated by a classic CICS transaction, which is either a terminal or a non-terminal oriented transaction. However, a process can also be started by any other CICS mechanism including activities of other processes.

#### 3.4.1 BTS Resources Used

The initiating transaction will use the following BTS resource:

- Container PROCESS I/O

#### 3.4.2 High Level Logic

The transaction MENU provides a simple map to enter some order-data. In addition, a customer-number and an order-number are required.

When the user presses Enter, the program writes data into the root activity's container after it has defined the process. Then, it starts the process with a RUN ACQPROCESS SYNC command. CICS suspends the execution of this task because of the synchronous execution of the process. When the process has ended, the program reads the results from the container. Figure 25 shows how a process activation can be implemented.

```

*.....
EXEC CICS DEFINE PROCESS(PROCESS-NAME)
                PROCESSTYPE('ORDER')
                TRANSID('SALE')
                PROGRAM('DFHOSAL2')

END-EXEC.
* Save order details in a process container and run the process
* synchronously so that we may return a response to the inputter
EXEC CICS PUT CONTAINER(PROCESS-NAME)
                FROM(PROCESS-CONTAINER)
                FLENGTH(PC-LENGTH)
                ACQPROCESS

END-EXEC.
EXEC CICS RUN ACQPROCESS
                SYNCHRONOUS
                RESP(RESP-AREA) RESP2(RESP2-AREA)

END-EXEC.
* return from process
* check RESP areas for normal completion
* do any necessary error handling here
* if ok then continue
* get process container with update results
EXEC CICS GET CONTAINER(PROCESS-NAME)
                INTO(PROCESS-CONTAINER)
                FLENGTH(PC-LENGTH)
                ACQPROCESS

END-EXEC
*.....

```

Figure 25. Pseudo-Code to Run a CICS BTS Process

The new API commands used in this pseudo-code example work as follows:

- EXEC CICS DEFINE PROCESS(PROCESS-NAME)

This command adds a new process to the CICS BTS system. The field PROCESS-NAME contains the name used for this process. This name can be up to 36 characters long and must be unique within the repository used.

As a process type, we choose the name ORDER, that has been defined in the CSD file through RDO.

The transaction ID used to run this process is SALE, and the process starts with program DFH0SAL2. The transaction must be defined in the CICS region in which the DEFINE PROCESS command is executed.

Notice that the program name is taken from the TRANSID parameter if no program is specified.

- EXEC CICS PUT CONTAINER(PROCESS-NAME)

The data addressed with the field or structure PROCESS-CONTAINER is saved and placed in the container. If the container already exists for this process, it will be overwritten. Otherwise, it will be created. The container name must be unique within this process instance. The length of a container is limited by the format of FLENGTH, which is a fullword binary value.

With the ACQPROCESS parameter, it is implied that the container is a process container.

- EXEC CICS RUN ACQPROCESS SYNCHRONOUS

This command executes the process. The execution is synchronous because of the parameter SYNCHRONOUS. Synchronous means the transaction that executes the RUN command becomes suspended until the UOW the process is executing in issues a RETURN command.

With a RUN command, CICS BTS does a context switch. A context switch means that:

- The process or activity runs in a UOW separate from the requestor.
- The transaction attributes specified with the DEFINE command are used.

The process is started with the system event DFHINITIAL.

When performance is more important to the application, it is recommended to use the LINK command. However, running the process or activity in a separate UOW gives you the ability to isolate failures.

Notice that you cannot use SYNCPOINT commands in a process that was initiated synchronously.

- EXEC CICS GET CONTAINER(PROCESS-NAME) ACQPROCESS

After the process has finished, the requesting program reads the process container with this command. The container is identified by its name and the process or activity that holds the container.

---

## 3.5 The Root Activity

The root activity uses these CICS BTS resources:

- Activity CREDIT (transaction RED1)
- Activity STOCK (transaction STOC)
- Activity DELIVERY (transaction DEL1)
- Activity INVOICE (transaction INV1)
- Activity REMINDER (transaction REM1)
- Container PROCESS I (process container)
- Container CREDIT I (child activity container)
- Container STOCK I (child activity container)
- Container TEMP 0 (activity container)
- Event DFHINITIAL (system event)
- Event DEL-INV-COMPLETE (composite event)
- Event DELETE TIMER (input event)
- Event SEND-PAY-REMINDER (timer event)
- Event REM-COMPLETE (completion event)
- Event CBAM-INSPECT (input event)

### 3.5.1 System Event DFHINITIAL

When the root activity receives a request to sell something, it starts the credit and stock activities. When both return with an OK message, this means the customer has credit, and the articles are available, a composite event concerning delivery and invoice is defined. Figure 26 on page 42 shows the logical work and event flow when the application is initially started (with system event DFHINITIAL). Before returning to CICS, the TEMP-container is written to save data for restart with other events. In Figure 27 on page 43 we introduce a pseudo-code listing of the root activity that comes as a sample application with CICS TS 1.3.

With the RETURN command, this activation of the process completes. Because the process has been started synchronously, the activating transaction MENU gets control to finish its work.

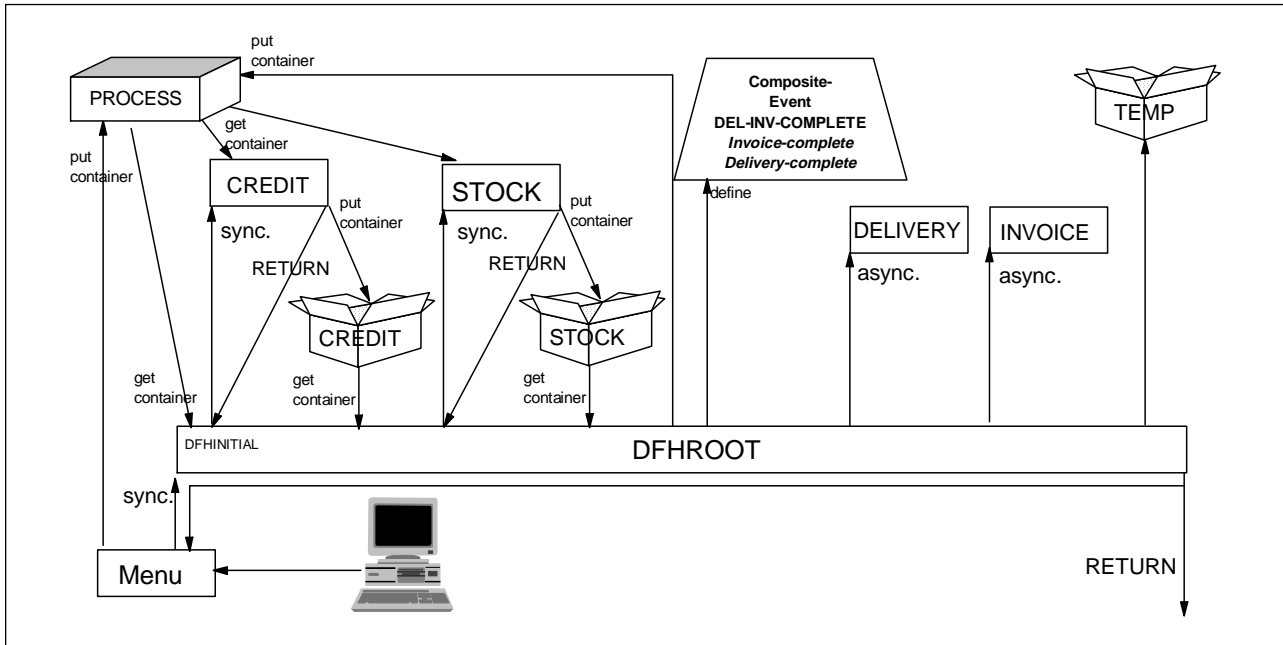


Figure 26. Process SALES Activation with DFHINITIAL

Notice that the UOW of the process finishes only when the MENU transaction executes a SYNCPOINT or RETURN command.

```

PROCEDURE DIVISION.
BEGIN-PROCESS.
  EXEC CICS RETRIEVE REATTACH EVENT(EVENT-NAME)
    RESP(RESP-AREA)
    RESP2(RESP2-AREA)
  END-EXEC.
  IF RESP-AREA NOT = DFHRESP(NORMAL) THEN
    ... do something to report the error
  END-IF.
  EVALUATE TRUE
    WHEN DFH-INITIAL
      PERFORM INITIAL-ACTIVITY
      PERFORM CHECK-CREDIT-STOCK-ACTIVITY
    WHEN .....
    WHEN OTHER
      ...
  END-EVALUATE.
  EXEC CICS RETURN END-EXEC.
.....
INITIAL-ACTIVITY.
  EXEC CICS ASSIGN PROCESS(PROCESS-NAME)
    RESP(RESP-AREA)
    RESP2(RESP2-AREA)
  END-EXEC.
  EXEC CICS GET CONTAINER(PROCESS-NAME)
    INTO(PROCESS-CONTAINER)
    FLENGTH(PC-LENGTH)
    PROCESS END-EXEC.
  END-EXEC.
* two synchronous activities. If both okay define one composite
* then two completion events being the sub-events. Then run
* two asynchronous activities as sub-events.
CHECK-CREDIT-STOCK-ACTIVITY.
* check credit-activity is run with a different userid for security
* reasons. This will cause a context switch. The new userid is specified
* on the DEFINE ACTIVITY statement.
  EXEC CICS DEFINE ACTIVITY('CREDIT-CHECK')
    TRANSID('RED1')
    PROGRAM('DFHORED1')
    USERID('ALCAPONE')
    RESP(RESP-AREA)
    RESP2(RESP2-AREA)
  END-EXEC.
  EXEC CICS RUN ACTIVITY('CREDIT-CHECK')
    SYNCHRONOUS
    RESP(RESP-AREA)
    RESP2(RESP2-AREA)
  END-EXEC.
  EXEC CICS CHECK ACTIVITY('CREDIT-CHECK')
    COMPSTATUS(WS-STATUS)
    ABCODE(ABEND-CODE)
  END-EXEC.
  IF WS-STATUS NOT = DFHVALUE(NORMAL) THEN
* do logic to handle bad completion of task.
  ELSE
    EXEC CICS GET CONTAINER('CREDIT-CONTAINER')
      ACTIVITY('CREDIT-CHECK')
      INTO (CREDIT-CHECK-CONTAINER)
      RESP(RESP-AREA)
      RESP2(RESP2-AREA)
    END-EXEC
  END-IF.

```

Figure 27 (Part 1 of 3). Pseudo-Code for the SALES Process Root Activity

```

*****
* CHECK-STOCK-ACTIVITY.
  EXEC CICS DEFINE ACTIVITY('STOCK-CHECK')
    TRANSID('STOC')
    PROGRAM('DFH0STOC')
    RESP(Resp-AREA)
    RESP2(Resp2-AREA)
  END-EXEC.
  EXEC CICS RUN ACTIVITY('STOCK-CHECK')
    SYNCHRONOUS
    RESP(Resp-AREA)
    RESP2(Resp2-AREA)
  END-EXEC.
  EXEC CICS CHECK ACTIVITY('STOCK-CHECK')
    COMPSTATUS(WS-STATUS)
    ABCODE(ABEND-CODE)
  END-EXEC.
  IF WS-STATUS NOT = DFHVALUE(NORMAL) THEN
* do logic for bad return from stock CHECK ACTIVITY
  ELSE
    EXEC CICS GET CONTAINER('STOCK-CONTAINER')
      ACTIVITY('STOCK-CHECK')
      INTO (STOCK-CHECK-CONTAINER)
      RESP(Resp-AREA)
      RESP2(Resp2-AREA)
    END-EXEC
    IF STOCK-ALL-GONE OR CREDIT-BROKE THEN
      PERFORM ORDER-NOT-ACCEPTED
    ELSE
      PERFORM ORDER-ACCEPTED-PROC
      PERFORM DEL-NOTE-AND-INVOICE-ACTIVITY
    END-IF
  END-IF.
* Stock and credit limits okay run delivery note and
* invoice activities as a COMPOSITE EVENT.
  DEL-NOTE-AND-INVOICE-ACTIVITY.
    EXEC CICS DEFINE ACTIVITY('DELIV-NOTE')
      EVENT('DEL-COMPLETE')
      TRANSID('DEL1')
      PROGRAM('DFH0DEL1')
      RESP(Resp-AREA)
      RESP2(Resp2-AREA)
    END-EXEC.
    EXEC CICS DEFINE ACTIVITY('INVOICE-BUILD')
      EVENT('INVOICE-COMPLETE')
      TRANSID('INV1')
      PROGRAM('DFH0INV1')
      RESP(Resp-AREA)
      RESP2(Resp2-AREA)
    END-EXEC.

```

Figure 27 (Part 2 of 3). Pseudo-Code for the SALES Process Root Activity

```

* now define a COMPOSITE EVENT for the buildof the
* delivery note and the invoice activities
EXEC CICS DEFINE COMPOSITE EVENT('DEL-INV-COMPLETE') AND
    RESP(RESP-AREA)
    RESP2(RESP2-AREA)
END-EXEC.
EXEC CICS ADD SUBEVENT('DEL-COMPLETE')
    EVENT('DEL-INV-COMPLETE')
    RESP(RESP-AREA)
    RESP2(RESP2-AREA)
END-EXEC.
EXEC CICS ADD SUBEVENT('INVOICE-COMPLETE')
    EVENT('DEL-INV-COMPLETE')
    RESP(RESP-AREA)
    RESP2(RESP2-AREA)
END-EXEC.
* write out a temporary state container before running the activities
PERFORM PUT-TEMP-CONTAINER.
* now run the two activities asynchronously
EXEC CICS RUN ACTIVITY('INVOICE-BUILD')
    ASYNCHRONOUS
    RESP(RESP-AREA)
    RESP2(RESP2-AREA)
END-EXEC.
EXEC CICS RUN ACTIVITY('DELIV-NOTE')
    ASYNCHRONOUS
    RESP(RESP-AREA)
    RESP2(RESP2-AREA)
END-EXEC.
*.....

```

Figure 27 (Part 3 of 3). Pseudo-Code for the SALES Process Root Activity

- EXEC CICS RETRIEVE REATTACH EVENT

The activity started by CICS BTS issues this command to determine the name of an event that causes reattachment.

CICS BTS provides the next event in the current activity's reattachment queue.

CICS BTS resets the status of the event to NOTFIRED if the retrieved event is atomic.

When started by CICS BTS, an activity must at least execute one RETRIEVE REATTACH EVENT command to ensure that an event is processed.

- EXEC CICS ASSIGN PROCESS(PROCESS-NAME)

This command provides information about the current activity. In this case, the program has to know the process name it is executing on behalf of to prepare for the next GET CONTAINER command.

- EXEC CICS DEFINE ACTIVITY('CREDIT-CHECK')

This command defines an activity to the CICS BTS system. The new activity becomes a child of the current activity. If no program name is specified, CICS BTS takes the name of the program to be executed from the transaction definition.

When necessary, you can define a user ID under whose authority this activity is executed when started by a RUN command. If the user ID is specified, CICS performs a surrogate security check to verify that the user ID used with the process is authorized to use the defined user ID.

- EXEC CICS RUN ACTIVITY('CREDIT-CHECK') SYNCHRONOUS

This command executes the activity CREDIT-CHECK synchronously with the root activity. Because of the RUN command the activity will be executed in another UOW using the transaction attributes from the previous DEFINE command.

The result of the activation of the activity cannot be discovered with the RUN command. The RESP fields only provide information about the request to run the activity. Instead you must use a CHECK ACTIVITY command.

- EXEC CICS CHECK ACTIVITY('CREDIT-CHECK')

By issuing this command, the requestor gets information whether the activity was executed successfully or not. It returns the completion status of an activity.

This command can be issued by:

- A parent activity
- A program that has acquired an activity

The COMPSTATUS option returns the completion status of the activity. When the returned value signals INCOMPLETE, the activity needs to be reattached in order to complete its processing.

If this command is issued by a parent activity, and the child has completed, CICS BTS deletes the child's completion event from the parent's event pool.

Therefore, this command is usually executed:

- Immediately after the RUN or LINK command if the activity is executed synchronously
- For an asynchronously executed activity:
  - When the parent is reattached due to the firing of the activity's completion event
  - When the parent is reattached because of the expiry of a timer

- EXEC CICS DEFINE COMPOSITE EVENT('DEL-INV-COMPLETE') AND

In this example, the command specifies a composite event that does not contain any subevents when defined. All subevents added later are interpreted as logically tied by the AND parameter. This means that the composite event fires when all of its subevents have fired.

After execution of this command, the composite event's status is FIRED because it contains no subevents. An empty composite event with the operator AND always has the status FIRED. If the empty composite event has the operator OR, its status will always be NOTFIRED.

- EXEC CICS ADD SUBEVENT('INVOICE-COMPLETE')

With this command, a subevent is added to the composite event referenced with the parameter EVENT. The subevent to be added must be an atomic event, not a system event or another composite event. When used, it cannot be part of another composite event. Additionally, it cannot be an input event if the parameter AND is used for the composite event. The subevent name must exist, it has to be defined before. Its name can be up to 16 characters long.



### 3.5.2 Composite Event DEL-INV-COMPLETE

During its initial execution, the root activity has defined the composite event DEL-INV-COMPLETE. The activity becomes reactivated when this event fires. Figure 28 shows the logical work- and event-flow, and Figure 29 on page 48 shows a pseudo-code listing when the activity is reactivated on firing of this composite event. Because the event is defined with the parameter AND, the event changes to *completed* when both sub-events are completed. In other words, the process progresses when the order is executed, and the invoice is printed.

On this reactivation, the activity evaluates the event that caused the activation. Afterwards, it reads the TEMP-container to have process-relevant data available.

To become informed on what has happened, the activity defines two events.

1. An input-event DELETE-TIMER. This event fires when a payment is made, and the corresponding 3270 transaction executes a RUN ACQPROCESS command (see 3.5.5, “Input Event DELETE-TIMER” on page 52).
2. A timer-event that expires after 28 days if no payment is received. This timer will be set again up to three times (see Figure 30 on page 49).

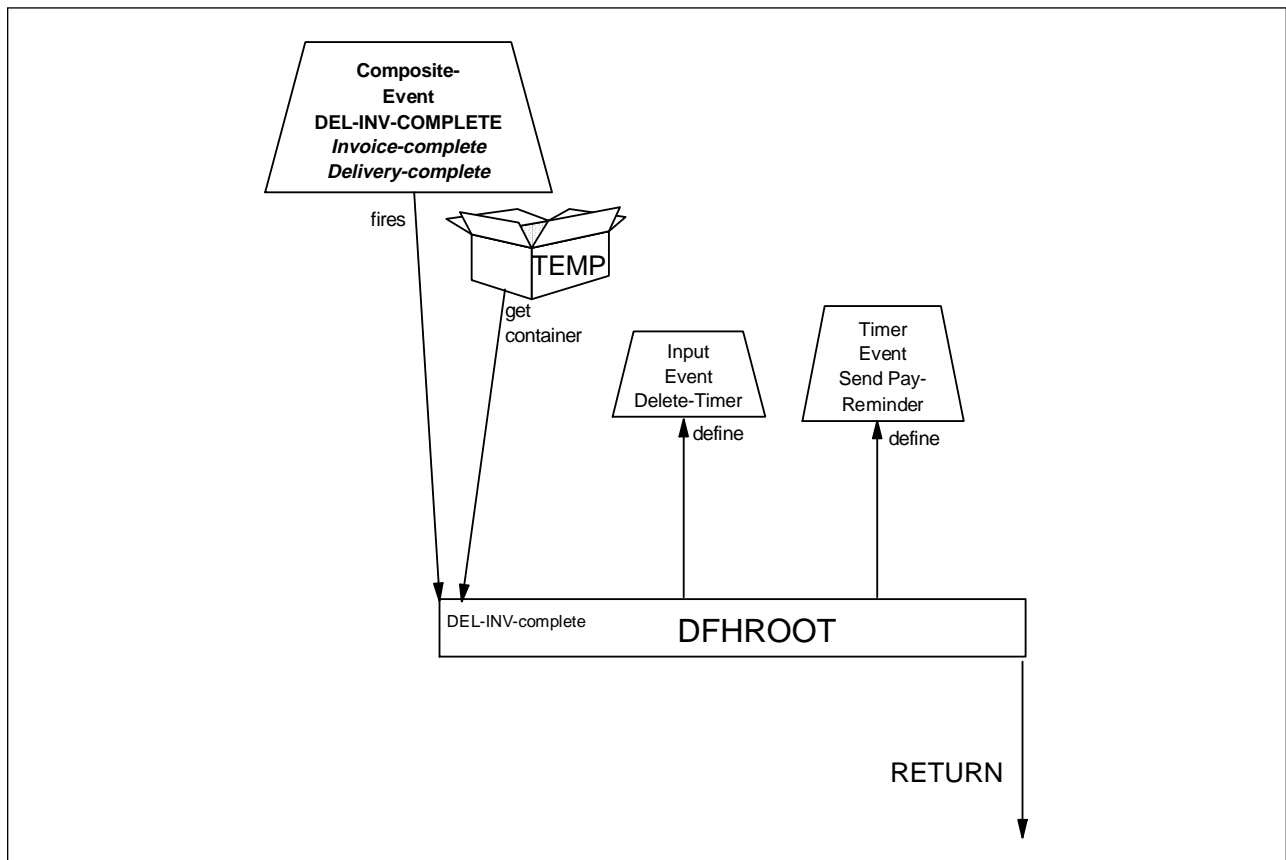


Figure 28. Process SALES, Activation with DEL-INV-COMPLETE

Figure 29 on page 48 shows the pseudo-code of the reactivation of the root activity illustrated in Figure 28.

```

PROCEDURE DIVISION.
BEGIN-PROCESS.
  EXEC CICS RETRIEVE REATTACH EVENT(EVENT-NAME)
    RESP(RESP-AREA)
    RESP2(RESP2-AREA)
  END-EXEC.
  IF RESP-AREA NOT = DFHRESP(NORMAL) THEN
    ... do something to report the error
  END-IF.
  EVALUATE TRUE
    WHEN .....
    WHEN DEL-INV-COMplete
      PERFORM DEL-INV-CHECK
    WHEN OTHER
      ...
  END-EVALUATE.
  EXEC CICS RETURN END-EXEC.
.....
DEL-INV-CHECK.
  EXEC CICS RETRIEVE SUBEVENT(SUB-EVENT-TYPE)
    EVENT(EVENT-NAME)
    EVENTTYPE(EVENT-TYPE)
    RESP(RESP-AREA)
    RESP2(RESP2-AREA)
  END-EXEC.
  IF RESP-AREA NOT = DFHRESP(NORMAL)
    ... do something ...
  END-IF.
  EVALUATE TRUE
    WHEN INVOICE-COMplete
      PERFORM INVOICE-CHECK
    WHEN DELIVERY-COMplete
      PERFORM DELIVERY-CHECK
    WHEN OTHER
      ...
  * unexpected SUBEVENT
    ... do something to report the error
  END-EVALUATE.
  * all COMPOSITE SUB-EVENTS okay DELETE COMPOSITE EVENT
  PERFORM GET-TEMP-CONTAINER.
  IF INV-OKAY = 'Y' AND
    DEL-OKAY = 'Y' THEN
    MOVE PP-OKAY-MESSAGE TO TRACE-MSG
    EXEC CICS DELETE EVENT('DEL-INV-COMplete')
  END-EXEC
  PERFORM SET-TIMER-EVENT
  * define INPUT EVENT until payment routine is complete
  EXEC CICS DEFINE INPUT EVENT('DELETE-TIMER')
  END-EXEC
  ELSE
    MOVE PP-NOT-OKAY-MESSAGE TO TRACE-MSG
  END-IF.
  * .....

```

Figure 29. Process SALES, Pseudo-Code of Activation with DEL-INV-COMplete

- EXEC CICS RETRIEVE SUBEVENT(SUB-EVENT-TYPE)

This command retrieves the name of the next subevent in the subevent queue of the composite event referenced with the EVENT parameter. The parameter SUBEVENT returns the name of the subevent. With EVENTTYPE, you retrieve the type (ACTIVITY, INPUT, or TIMER) of the subevent.

The status of the subevent is reset to NOTFIRED.

Notice that the sample program only uses one RETRIEVE SUBEVENT command. This is correct because the composite event uses the AND parameter. This means both subevents must have finished to cause the composite event to fire. The firing of a subevent just indicates that the activity

has finished. It does not indicate whether it has finished successfully or not. To examine the activity's success, the CHECK ACTIVITY command is used. This command is not shown in the pseudo-code. It is executed in the paragraphs INVOICE-CHECK and DELIVERY-CHECK.

- EXEC CICS DELETE EVENT('DEL-INV-COMPLETE')

CICS BTS does not delete a fired composite event. Because the composite event is not needed anymore, we delete it with this command. To delete it means to remove it from the activity's event pool.

- EXEC CICS DEFINE INPUT EVENT('DELETE-TIMER')

With the input event define command, the activity prepares to become reattached because of the firing of this event. Firing of this event is caused by a program outside of CICS BTS that runs the process or the activity with this input event. It is used to inform the process that something in context with this event (a payment for example) was done.

### 3.5.3 Timer Event SEND-PAY-REMINDER

When the timer expires before a payment is received, the timer event SEND PAY-REMINDER fires. After evaluating again which event caused the reactivation and reading the data-container, the root activity defines and runs a child activity (Reminder) that is executed asynchronously. This activation logic is shown in Figure 30.

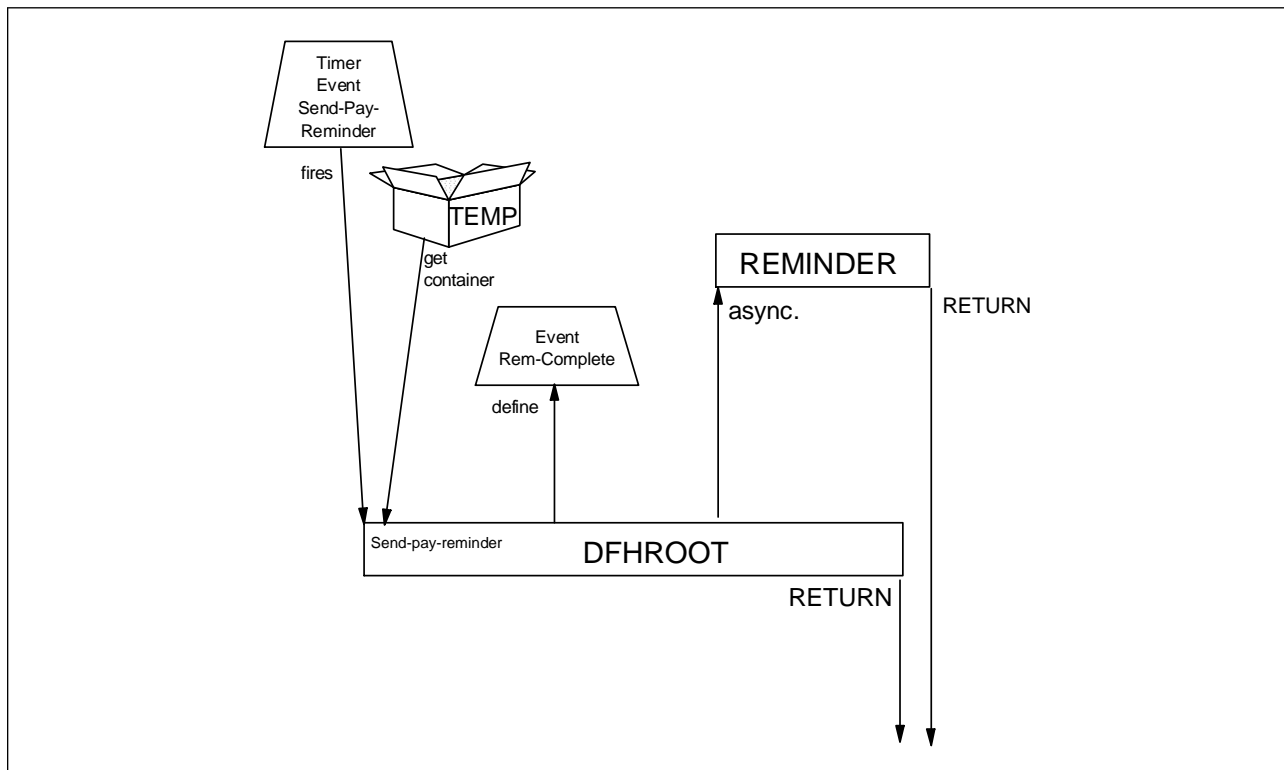


Figure 30. Process SALES, Activation with SEND-PAY-REMINDER

Figure 31 on page 50 shows the pseudo-code of the reactivation.

```

PROCEDURE DIVISION.
BEGIN-PROCESS.
  EXEC CICS RETRIEVE REATTACH EVENT(EVENT-NAME)
    RESP(RESP-AREA)
    RESP2(RESP2-AREA)
  END-EXEC.
  IF RESP-AREA NOT = DFHRESP(NORMAL) THEN
    ... do something to report the error
  END-IF.
  EVALUATE TRUE
    WHEN .....
    WHEN SEND-PAY-REMINDER
      PERFORM SEND-REMINDER
    WHEN OTHER
      ...
  END-EVALUATE.
  EXEC CICS RETURN
  END-EXEC.
.....
SEND-REMINDER.
* if not first time through you must reset the activity before
* you can rerun it
PERFORM GET-TEMP-CONTAINER.
IF REM-IND = 'Y' THEN
  EXEC CICS RESET ACTIVITY('SEND-REMINDER')
  END-EXEC
ELSE
  MOVE 'Y' TO REM-IND
  EXEC CICS DEFINE ACTIVITY('SEND-REMINDER')
    TRANSID('REM1')
    PROGRAM('DFHOREM1')
    EVENT('REM-COMPLETE')
    RESP(RESP-AREA)
    RESP2(RESP2-AREA)

  END-EXEC
END-IF.
ADD 1 TO REM-COUNT.
PERFORM PUT-TEMP-CONTAINER.
IF REM-COUNT NOT > 3 THEN
  EXEC CICS RUN ACTIVITY('SEND-REMINDER')
    ASYNCHRONOUS
    RESP(RESP-AREA)
    RESP2(RESP2-AREA)

  END-EXEC
ELSE
* you have now sent out three reminders and not received any payment
* Put your logic here as per your business requirements.
  NEXT SENTENCE
END-IF.
.....

```

Figure 31. Process SALES, Pseudo-Code of Activation with SEND-PAY-REMINDER

- EXEC CICS RESET ACTIVITY('SEND-REMINDER')

The child activity SEND-REMINDER is reset to its initial state with this command. Its completion event is added to the parent's event pool with the fired status set to NOTFIRED.

This command is necessary before a second RUN command is issued on this activity. With this second RUN command, the activity is started with DFHINITIAL.

The activity to be reset has to be in a complete or initial mode.

### 3.5.4 Completion Event REM-COMLETE

When the reminder is written, the root activity becomes reactivated by the completion event REM-COMLETE. This reactivation just informs the root activity of the successfully written reminder. Figure 32 shows that, during this reactivation, the time is defined again up to three times.

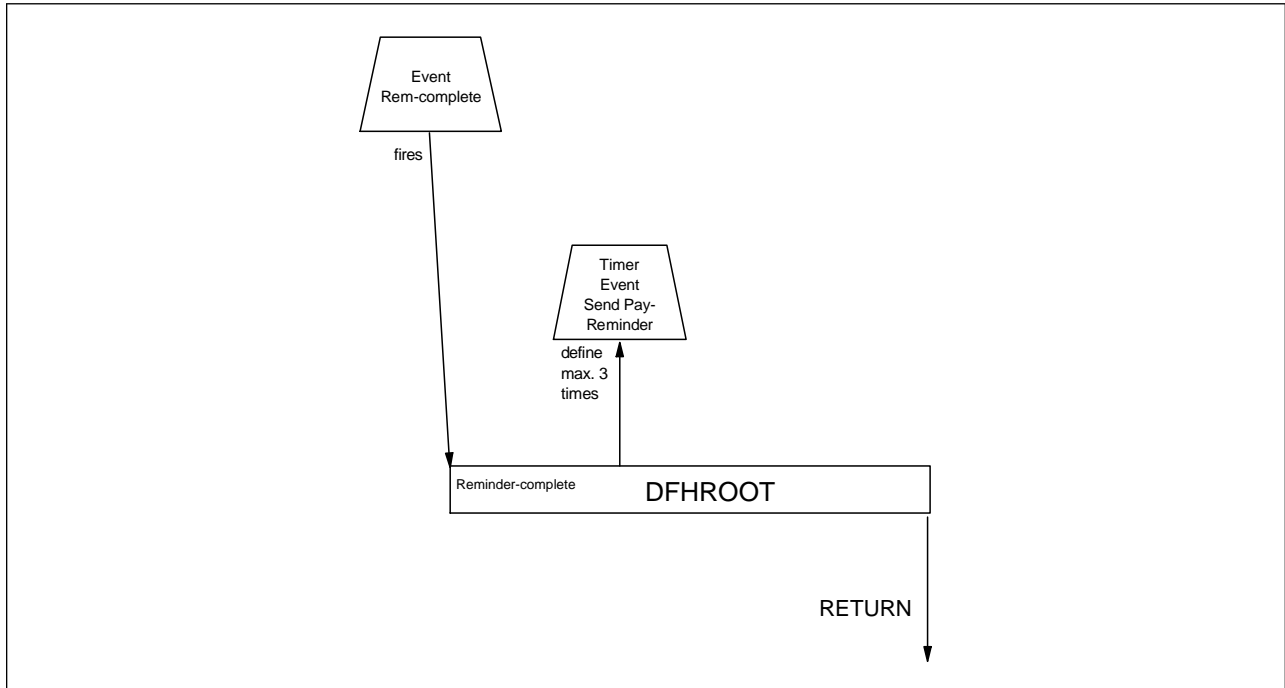


Figure 32. Process SALES, Activation with REMINDER-COMLETE

The logic that must be done if no payment was received after three reminders have been sent is not developed, as this does not give any further insight into the use of CICS BTS. It is purely a business approach. Figure 33 on page 52 shows the pseudo-code of the reactivation.

```

PROCEDURE DIVISION.
BEGIN-PROCESS.
  EXEC CICS RETRIEVE REATTACH EVENT(EVENT-NAME)
    RESP(RESP-AREA)
    RESP2(RESP2-AREA)
  END-EXEC.
  IF RESP-AREA NOT = DFHRESP(NORMAL) THEN
    ... do something to report the error
  END-IF.
  EVALUATE TRUE
    WHEN .....
    WHEN REMINDER-COMPLETE
      PERFORM DEL-REM-COMPLETE
      PERFORM DELETE-TIMER-EVENT
      PERFORM SET-TIMER-EVENT
    WHEN OTHER
      ...
  END-EVALUATE.
  EXEC CICS RETURN
  END-EXEC.

.....
DEL-REM-COMPLETE.
  EXEC CICS CHECK ACTIVITY('SEND-REMINDER')
    COMPSTATUS(WS-STATUS)
    ABCODE(ABEND-CODE)

  END-EXEC.
DELETE-TIMER-EVENT.
  EXEC CICS DELETE TIMER('REM-TIMER')
  END-EXEC.
SET-TIMER-EVENT.
  EXEC CICS DEFINE TIMER('REM-TIMER')
    EVENT('SEND-PAY-REM')
    AFTER DAYS(28)

  END-EXEC.
.....

```

Figure 33. Process SALES, Pseudo-Code of Activation with REMINDER-COMPLETE

- EXEC CICS DELETE TIMER('REM-TIMER')

The timer REM-TIMER is deleted. The associated event is removed from the activity's event pool. Notice that there will be no event with the timer if the timer has expired, and a CHECK TIMER command has been issued.

A timer can only be deleted by the activity that owns the timer, but it can delete it whether or not it has expired.

- EXEC CICS DEFINE TIMER('REM-TIMER')

This command defines a timer that will expire after 28 days. The associated event is SEND-PAY-REM. When the EVENT parameter is not used, the event has the same name the timer has.

### 3.5.5 Input Event DELETE-TIMER

Figure 34 on page 53 shows what happens when a payment is made. The PAYM 3270 transaction reactivates the root activity using the RUN ACQPROCESS INPUTEVENT command. The process is executed synchronously. The firing of the event DELETE-TIMER causes CICS BTS to reactivate the process. The root activity deletes the container PROCESS, the timer, and the event DELETE-TIMER.

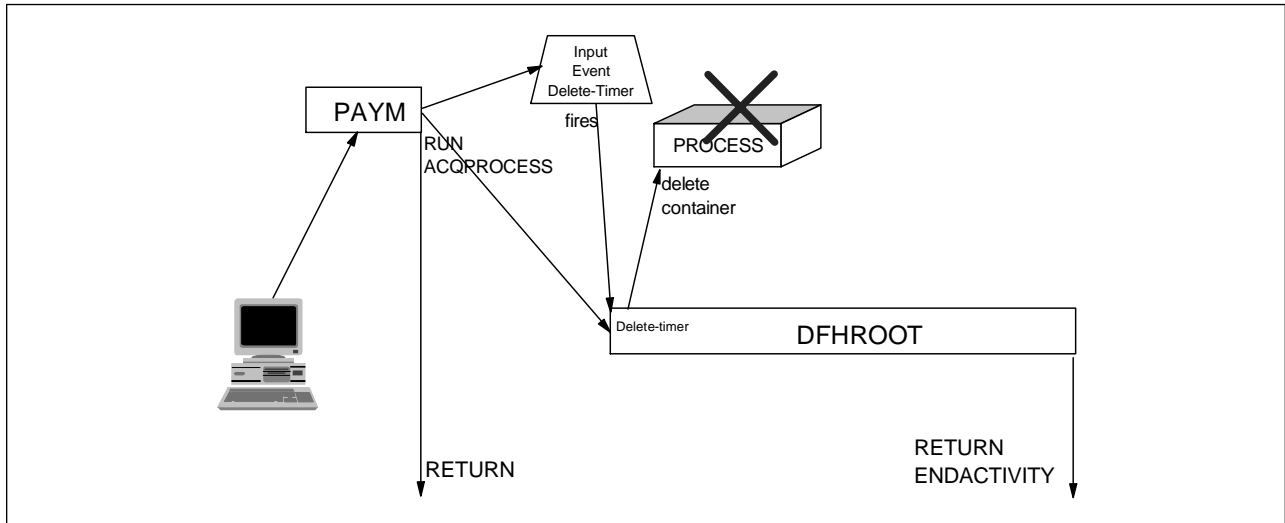


Figure 34. Process SALES, Activation with DELETE-TIMER

- EXEC CICS ACQUIRE PROCESS

This command enables the PAYM transaction to access the root activity within the process referenced by the PROCESS parameter.

When using the ACQUIRE PROCESS command, the DEFINE PROCESS command cannot be used within the same UOW. Also an ACQUIRE ACTIVITYID command cannot be issued within this UOW.

The program is allowed to read or update the process containers and the root activity's containers.

- EXEC CICS RUN ACQPROCESS

Used with the parameter INPUTEVENT, this command causes CICS BTS to reactivate the acquired process with the input event. Therefore, the root activity has to define this input event beforehand.

If you issue multiple asynchronous RUN commands against the same activity within the same UOW:

- Each RUN command after the first fails, if you specify the same input event.
- If you specify different input events, the activity may or may not be invoked as many times as the number of RUN requests. The only guarantee is that it will be invoked at least once. For example, if, within the same UOW, you issue five asynchronous RUN requests for the same activity, specifying different input events, the activity might be invoked twice. At the first invocation, three input events might be presented and at the second — two.

If it is necessary to issue RUN requests in this manner, you should design your activity so that it issues RETRIEVE REATTACH EVENT commands until the END condition occurs. Remember, no syncpoint processing can be used within a CICS BTS UOW.

Figure 34 illustrates this part of the root activity. Figure 35 on page 54 shows the pseudo-code of the reactivation.

```

PROCEDURE DIVISION.
BEGIN-PROCESS.
  EXEC CICS RETRIEVE REATTACH EVENT(EVENT-NAME)
    RESP(RESP-AREA) RESP2(RESP2-AREA)
  END-EXEC.
  IF RESP-AREA NOT = DFHRESP(NORMAL) THEN
    ... do something to report the error
  END-IF.
  EVALUATE TRUE
    WHEN .....
    WHEN DELETE-TIMER
      PERFORM DEL-TIMER-AND-EVENTS
      PERFORM DELETE-PROCESS-CONTAINER
    WHEN OTHER
      ...
  END-EVALUATE.
  EXEC CICS RETURN
  END-EXEC.
.....
DEL-TIMER-AND-EVENTS.

  EXEC CICS DELETE TIMER('REM-TIMER')
  END-EXEC.
  EXEC CICS DELETE EVENT('DELETE-TIMER')
  END-EXEC.

DELETE-PROCESS-CONTAINER.

  EXEC CICS ASSIGN PROCESS(PROCESS-NAME)
    RESP(RESP-AREA)
    RESP2(RESP2-AREA)
  END-EXEC.
  EXEC CICS DELETE CONTAINER(PROCESS-NAME)
    PROCESS
  END-EXEC.
.....

```

Figure 35. Process SALES, Pseudo-Code of Activation with DELETE-TIMER

- EXEC CICS DELETE EVENT('DELETE-TIMER')
- This command is used to remove a no longer needed input event or composite event from the current activity's event pool. However, completion events, timer events, and system events cannot be deleted.
- EXEC CICS DELETE CONTAINER(PROCESS-NAME) PROCESS
- A container will be deleted with this command. Using the parameter PROCESS will delete the process container. This can be done only by the root activity or a program that has acquired the process.

### 3.5.6 Activities (Children)

A child activity in its simplest form is the implementation of a small step of an application. An activity fulfills a part of a business process. For example, when *Customer Advice* is the business process to be implemented, *Mortgage Contract* may be a business task (a part of the business process). *Credit Disbursement* may be an element of this task. This element is typically implemented by a child activity. A template for such an activity is shown in Figure 36 on page 55.



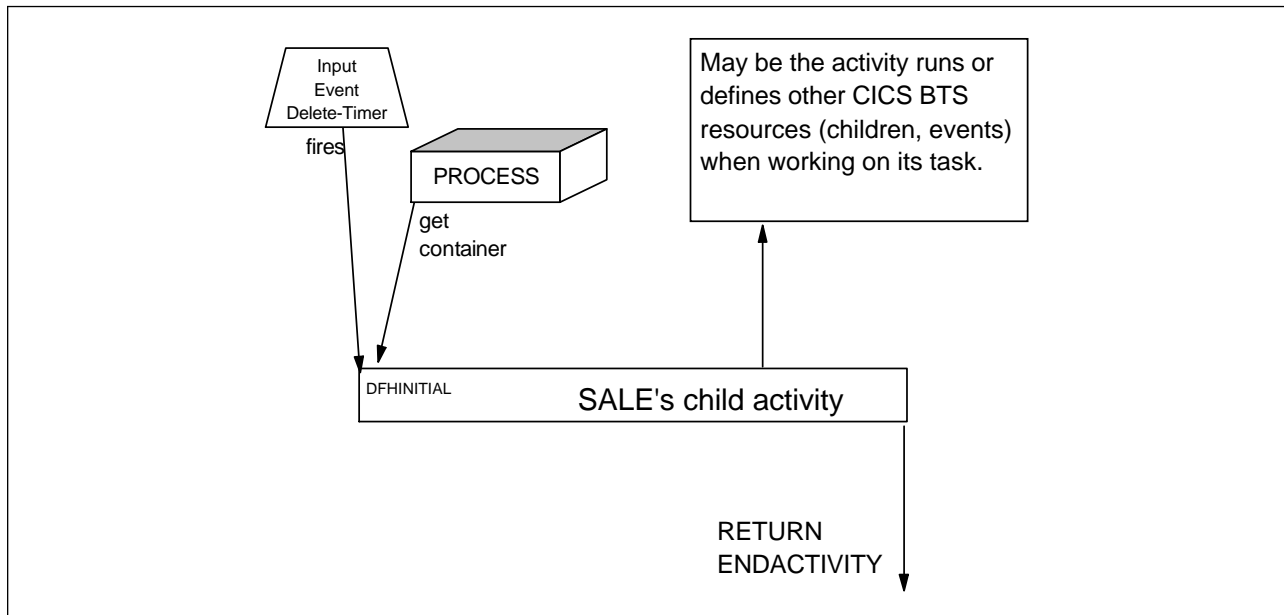


Figure 36. Categorized Child Activity

Figure 37 shows the pseudo-code of such an activity.

```

PROGRAM-ID. CHILDEXAMPLE
DATA DIVISION.
WORKING-STORAGE SECTION.
01  EVENT-NAME                PIC X(16).
    88 DFH-INITIAL            VALUE 'DFHINITIAL'.
LINKAGE SECTION.
PROCEDURE DIVISION.
BEGIN-PROCESS.
    EXEC CICS RETRIEVE REATTACH EVENT(EVENT-NAME)
        RESP(RESP-AREA) RESP2(RESP2-AREA)
    END-EXEC.
    IF RESP-AREA NOT = DFHRESP(NORMAL) THEN
        PERFORM ABEND-PROCESS
    END-IF.
    EVALUATE TRUE
        WHEN DFH-INITIAL
            PERFORM INITIAL-ACTIVITY
        WHEN OTHER
            EXEC CICS WRITEQ TD
                QUEUE('CSMT')
                FROM(INVALID-EVENT)
                LENGTH(INVALID-EVENT-LEN)
            END-EXEC
    END-EVALUATE.
*   ... do something ...
    EXEC CICS RETURN ENDACTIVITY
    END-EXEC.
INITIAL-ACTIVITY.
    EXEC CICS ASSIGN PROCESS(PROCESS-NAME)
        RESP(RESP-AREA) RESP2(RESP2-AREA)
    END-EXEC.
    EXEC CICS GET CONTAINER(PROCESS-NAME)
        INTO(PROCESS-CONTAINER)
        FLENGTH(PC-LENGTH)
        PROCESS
    END-EXEC.
*   ... do application logic here ...

```

Figure 37. CICS BTS, Pseudo-Code of an Activity

- EXEC CICS RETURN ENDACTIVITY

Used with the ENDACTIVITY option, this command indicates that the current activity is about to complete.

It returns control to the next higher CICS level. With CICS BTS, an activity returns control to its parent, the root activity returns control to CICS.

If there are no more user events in the activity's event pool it completes normally. With the normal completion of a root activity, CICS discards the process and all its components.

We recommend to issue a RETURN ENDACTIVITY command at the end of the final activation of the activity. If the event pool still contains user events (not completion events), these events are deleted by CICS BTS.

If the RETURN ENDACTIVITY command is not used, the same situation can cause the process to become dormant forever.

---

## Part 2. More Complex Application Systems



---

## Chapter 4. HOLIDAY Application

The holiday application introduces the concept of *compensation* (see 4.1, “Compensation”), and looks at ways that containers can be used to pass data between a parent and its child activities. We also demonstrate, by example, the differences between RUN ACTIVITY SYNCHRONOUS and LINK ACTIVITY commands and give an example of how the browse commands can be used (see 4.5.7.2, “Tidy Up Children” on page 88).

---

### 4.1 Compensation

Compensation is the act of modifying the effects of a completed activity. How this is implemented is decided by the application designer, but often means the undoing, or reversing, of the actions that the activity took. For example, compensation for a booked flight might be to cancel the flight and refund the money.

Two possible ways of implementing the compensation of a completed child activity are:

- Define and run a new compensation activity.

This is the method used in the HOLIDAY application (see 4.5.7.1, “Compensating Activities” on page 87).

You define an activity that reverses the business logic of the activity to be compensated. In the application, the money deducted from the personal balance is added back if the flight cannot be taken. The activity is executed by a completely new program.

- Re-run the activity

The original activity is re-run with logic to compensate the original activation. A flag in the activity's container must have been set by the parent in order for the activity to know which course of action to take. The activity will have to have been reset with the RESET ACTIVITY command.

---

### 4.2 HOLIDAY Application Overview

The HOLIDAY application has been based on a travel agent scenario. It assumes that a customer has entered the agent's office and has requested to book a holiday package consisting of a hotel reservation, a flight, the use of a car, and up to three excursions. These items are parts of a complete holiday package.

Initially, all parts of the package have to be confirmed before a holiday business transaction can be completed.

If any one of the parts cannot be confirmed, it is necessary to look at this part to establish whether the holiday booking can proceed. It is acceptable for a holiday to proceed if any excursions or the car booking fails, but if either the hotel or the flight booking fails, the customer is unable to have his chosen holiday.

There are two main reasons why a booking may fail:

- The customer does not have enough money in his bank account to be able to pay for an element.
- The overall booking took too long to complete. Maybe, for example, the airline did not respond quickly enough.

If the holiday is not able to proceed then the parts that have been booked will have to be reversed or *compensated* (see 4.5.7.1, “Compensating Activities” on page 87) and the money previously deducted from the customer's account refunded.

The holiday business transaction starts by defining a process and running a root activity to raise a holiday booking. This will control activities that book all the individual parts of a holiday package - flight, hotel, car, and up to three excursions. Parallel processing of activities and compensation activities is involved.

The application was developed as part of the CICS TS 1.3 System Test suite and uses a control file to determine variable details. This control file also holds data that can be used to build the personal details file.

Throughout the application, messages were written to a user-defined TD queue, BMTD, as a means of monitoring progress. The paragraph WHICH-TRANSACTION was used to write messages at the start of each activity. The output and copybooks to produce the output can be seen in 4.8.1, “The TD Queue Output” on page 96. In a production system, we would advise that this record is written to a more permanent medium, or that the AUDIT log facility provided by CICS BTS be used.

The HOLIDAY application implements a travel agent's booking system and is made up of 11 actions:

- An initial entry transaction to invoke the HOLIDAY business transaction
- Root Activity
- Book Hotel
- Book Flight
- Book Car
- Book Excursions
- Cancel Hotel
- Cancel Flight
- Cancel Car
- Cancel Excursions
- Confirmation or refusal (written to CSMT log)

The holiday business transaction will be started by a terminal user initiating a CICS transaction that will initiate the holiday request. This transaction, in turn, will define and run an instance of the business transaction that will control the business activities. The root activity will define and run activities that can implement the customer's requirements.

## 4.3 Transactions, Programs, Containers, and Control File

In this section we outline the transactions and programs used in the application, show the layout of the containers, and describe the way that the control file is used.

### 4.3.1 Transactions and Programs

Table 4 shows the transaction IDs and programs used in the business transaction.

<b>Activity</b>	<b>Transid</b>	<b>Program</b>	<b>Comments</b>
-	H001	HOLI001	A traditional CICS transaction used to invoke the CICS BTS process
DFHROOT	HHOL	HHOLIDAY	BTS root activity, manages the child activities that comprise the Holiday business transaction
BOOKFLIGHT	HFLY	HBOOKFLY	Book Flight activity
BOOKHOTEL	HHOT	HBOOKHOT	Book Hotel activity
BOOKCAR	HCAR	HBOOKCAR	Book Car activity
BOOKEXCURx	HEXC	HBOOKEXC	Book Excursions activity
CANCELFLIGHT	HCFL	HCANFLY	Cancel Flight activity
CANCELHOTEL	HCHT	HCANHOT	Cancel Hotel activity
CANCELCAR	HCCR	HCANCAR	Cancel Car activity
CANCELEXCURx	HCEX	HCANEXC	Cancel Excursions activity

### 4.3.2 Container Contents

Figure 38 on page 62 shows the working storage definitions for the contents of the containers used in the holiday application. It should be noted that not all the fields are used in the application.

```

*****
*   GENERAL container LAYOUT COPYBOOK   *
*****
01 FILLER PIC X(20) VALUE IS '*** C/B general ***'.
01 C-GENERAL.
   03 C-PROCESS-TYPE          PIC X(8) VALUE SPACES.
   03 C-LOCAL-PROCESS-NAME    PIC X(36) VALUE SPACES.
   03 C-REMOTE-PROCESS-NAME   PIC X(36) VALUE SPACES.
   03 C-TSQUEUE-ITEM          PIC 9(4) BINARY VALUE ZEROS.
   03 C-MORTGAGE-REF-KEY      PIC 9(9) VALUE zeros.
   03 C-CONTROL-FILE-NAME     PIC X(8) VALUE SPACES.

*****
*   RETRY container LAYOUT COPYBOOK     *
*****
01 FILLER PIC X(20) VALUE IS '*** C/B RETRY ***'.
01 C-RETRY.
   03 C-RETRY-COUNT           PIC 99.

*****
*   BOOKFL container LAYOUT COPYBOOK as an example *
*****
01 FILLER PIC X(20) VALUE IS '*** C/B BOOKFL ***'.
01 C-BOOKFL.
   03 C-FLIGHT-AGREED        PIC X VALUE 'N'.
   88 FLIGHT-AGREED         VALUE 'Y'.
   03 C-FLIGHT-COST          PIC 9(3)V99 VALUE ZEROS.
   03 C-FLIGHT-PLUS-MINUS    PIC X VALUE '+'.
   88 FLIGHT-PLUS           VALUE '+'.
   88 FLIGHT-MINUS          VALUE '-'.

*****
*   HOLIDAY CONTAINER LAYOUT COPYBOOK   *
*   TO BE USED TO HOLD COMPLETION OF EACH STAGE OF THE HOLIDAY *
*   FOR USE IN HHOLIDAY                 *
*****
01 FILLER PIC X(20) VALUE IS '*** C/B HOLIDAY ***'.
01 C-HOLIDAY-EVENTS.
   03 C-HOLIDAY-BOOKED      PIC X VALUE 'N'.
   88 HOLIDAY-BOOKED       VALUE 'Y'.
   03 C-FLIGHT-BOOKED      PIC X VALUE 'N'.
   88 FLIGHT-BOOKED        VALUE 'Y'.
   88 FLIGHT-CANCEL         VALUE 'C'.
   88 FLIGHT-FAILED        VALUE 'N'.
   03 C-HOTEL-BOOKED       PIC X VALUE 'N'.
   88 HOTEL-BOOKED         VALUE 'Y'.
   88 HOTEL-CANCEL         VALUE 'C'.
   88 HOTEL-FAILED        VALUE 'N'.
   03 C-CAR-BOOKED         PIC X VALUE 'N'.
   88 CAR-BOOKED          VALUE 'Y'.
   88 CAR-CANCEL           VALUE 'C'.
   88 CAR-FAILED          VALUE 'N'.
   03 C-EX1-BOOKED         PIC X VALUE 'N'.
   88 EX1-BOOKED          VALUE 'Y'.
   88 EX1-CANCEL           VALUE 'C'.
   88 EX1-FAILED          VALUE 'N'.
   03 C-EX2-BOOKED         PIC X VALUE 'N'.
   88 EX2-BOOKED          VALUE 'Y'.
   88 EX2-CANCEL           VALUE 'C'.
   88 EX2-FAILED          VALUE 'N'.
   03 C-EX3-BOOKED         PIC X VALUE 'N'.
   88 EX3-BOOKED          VALUE 'Y'.
   88 EX3-CANCEL           VALUE 'C'.
   88 EX3-FAILED          VALUE 'N'.

```

Figure 38. Contents of Containers



### 4.3.3 The Control File

The control file includes a master control record 999999999. This master control record stores in its second nine bytes the key of the next record to use. As multiple control files could be defined, the file name is stored in the GENERAL container of the process together with the key for the control and personal details file record for this invocation of the process. See Figure 38 on page 62 for the layout of the GENERAL process container.

The file contains initialization data for the personal details file, timer fields for events and a number of processing control flags. The control file was set up for the entire System Test CICS BTS suite and, therefore, may contain fields that are not used in the HOLIDAY process.

The following fields are relevant:

- Reference key
- The process type to be used when defining the process
- Flags to indicate how each component of a holiday booking should be handled, for example, available, not available, or no response
- Flags to indicate whether to use RUN SYNCHRONOUS or LINK to start certain activities
- Timer fields to delay the processing of child activities
- Data about the customer's personal balance and salary
- Data about the customer's bank account
- Data showing the cost of each element to be booked

A transaction was also written so that the control file could be updated as needed. Figure 39 on page 64 shows some of the panels used to update the control file.

```

BAM MORTGAGE CONTROL FILE

ENTER CONTROL FILE NAME: MORTCONT
ENTER OPTION: VIEW (ADD,UPDATE,DELETE,VIEW)
MORTGAGE REFERENCE KEY: 000000001

CUSTOMER NAME: FRED BLOGGS - NO 1
ADDRESS: 1 ACACIA AVENUE
NEWTOWN
HANTS
TELEPHONE: 01962 777777
SALARY: 50000 BALANCE: 0005000
BONUS: 00300

BANK NAME: NATWEST BANK
ADDRESS: HIGH STREET
WINCHESTER
HAMPSHIRE
SORT CODE: 012345 ACCOUNT NUMBER: 54545454

F3 EXIT F5 ACTION F8 FORWARD F10 VIEW NEXT RECORD F11 VIEW PREV RECORD
APPLID IYCQTSF

.....

BAM MORTGAGE CONTROL FILE - 6

HOLIDAY AVAILABILITY AND COST
AVAILABILITY (Y/N) COST: $$$PP
FLIGHT: Y FLIGHT: 00123
HOTEL : Y HOTEL : 45000
CAR : Y CAR : 26500
EX1 : Y EX1 : 05000
EX2 : Y EX2 : 05000
EX2 : Y EX3 : 05000

HOLIDAY TIMEOUT TIMERS - FOR NO RESPONSE TESTING
AFTER(Y/N): Y AT(Y/N): N ON(Y/N): N
DAYS: 0000 HOURS: 0000 MINUTES: 0000 SECONDS: 0030
YEAR: 0000 MONTH: 0000 DAYOFMONTH: 0000 DAYOFYEAR: 0000

.....

BAM MORTGAGE CONTROL FILE - 7

ENVIRONMENTAL CONTROL
PROCESSTYPE: BAMSHRA
RUN/LINK (R/L)
HOLIDAY : L

GENERAL CONTAINER LENGTH: 0000108

```

Figure 39. Panels Used to Update the Control File

### 4.3.4 Coverage of API

This design uses the following API calls:

- ASSIGN PROCESS(PROCESS-NAME)
- ASSIGN ACTIVITY(ACTIVITY-NAME)
- DEFINE PROCESS(PROCESS-NAME)
- RUN ACQPROCESS
- DEFINE ACTIVITY(ACTIVITY-NAME)
- RUN ACTIVITY(ACTIVITY-NAME)
- LINK ACTIVITY(ACTIVITY-NAME)
- CHECK ACTIVITY(ACTIVITY-NAME)

- RESET ACTIVITY(ACTIVITY-NAME)
- DELETE ACTIVITY(ACTIVITY-NAME)
- RETRIEVE REATTACH EVENT(EVENT-NAME)
- GET CONTAINER(CONTAINER-NAME)
- PUT CONTAINER(CONTAINER-NAME)
- DELETE CONTAINER(CONTAINER-NAME)
- DEFINE COMPOSITE EVENT(COMPOSITE-EVENT-NAME)
- ADD SUBEVENT(EVENT-NAME)
- DEFINE TIMER(TIMER-NAME)
- DELETE TIMER(TIMER-NAME)
- DELETE EVENT(EVENT-NAME)
- TEST EVENT(EVENT-NAME)
- STARTBROWSE EVENT
- GETNEXT EVENT(DATA-AREA)
- ENDBROWSE EVENT
- STARTBROWSE CONTAINER
- GETNEXT CONTAINER(DATA-AREA)
- ENDBROWSE CONTAINER
- STARTBROWSE ACTIVITY
- GETNEXT ACTIVITY(DATA-AREA)
- ENDBROWSE ACTIVITY
- INQUIRE ACTIVITYID(ACTIVITYID)
- RETURN ENDACTIVITY

---

## 4.4 The Initiating Transaction

The initiating transaction is used to invoke the process initially and to update the global record of the control file for the next invocation of the process.

### 4.4.1 BTS Resources Used

The initiating transaction H001 uses the following resources:

- Container GENERAL            O    (process)
- File MORTCONT                I/O
- File PERSDET                 O
- Event DFHINITIAL

## 4.4.2 The High-Level Logic

The high-level logic is as follows:

Program HOLI001 is started with transaction ID H001. It reads the control file's master control record to discover the next key to use and then reads the appropriate control file record.

A HOLIDAY process is defined with TRANSID(HHOL) using program HHOLIDAY and the process name and process type are put to the GENERAL process container.

The acquired process is then run asynchronously.

The personal details file fields are initialized with data from the control file, and then the new record is written to the file.

The next key field in the global record is updated by adding one to the current value, and the record is written back to the control file.

Figure 40 on page 67 shows the pseudo-code for the initiating transaction H001.

```

AA-MAIN SECTION.
*
* The last record on the control file (master control record)
* holds the record key of the new process being defined.
* HOLI concatenated with the next record key gives the process
* name of the new process.
*
  MOVE 999999999 TO MORTGAGE-REF-KEY1
  EXEC CICS READ FILE(WS-CONTROL-FILE)
           INTO(WS-MASTER-CONTROL-RECORD)
           RIDFLD(MORTGAGE-REF-KEY1)
           UPDATE
           RESP(WS-RESP) RESP2(WS-RESP2)
  END-EXEC.
*
* The next record key is stored in general container and used
* to access the control data in the control file
* Additionally this key is used to form the process name.
*
  MOVE MORTGAGE-NEXT-KEY TO WS-PROCESS-KEY C-MORTGAGE-REF-KEY
  EXEC CICS READ FILE(WS-CONTROL-FILE)
           INTO(MORTGAGE-CONTROL-RECORD)
           RIDFLD(C-MORTGAGE-REF-KEY)
           RESP(WS-RESP) RESP2(WS-RESP2)
  END-EXEC.
*
* The new HOLIDAY process is defined
*
  EXEC CICS DEFINE PROCESS(WS-NEW-PROCESS)
           PROCESSTYPE(MC-PROCESSTYPE)
           TRANSID('HHOL')
           PROGRAM('HHOLIDAY')
           RESP(WS-RESP) RESP2(WS-RESP2)
  END-EXEC
  END-IF
*
* Other process wide information is moved to the
* GENERAL container field and PUT the GENERAL container
*
  MOVE WS-NEW-PROCESS TO C-LOCAL-PROCESS-NAME
  MOVE MC-PROCESSTYPE TO C-PROCESS-TYPE
  MOVE WS-CONTROL-FILE TO C-CONTROL-FILE-NAME
  EXEC CICS PUT CONTAINER('GENERAL')
           ACQPROCESS
           FROM(C-GENERAL)
           RESP(WS-RESP) RESP2(WS-RESP2)
  END-EXEC
*
* Output to BMTD TD queue for diagnostics
*
  MOVE WS-NEW-PROCESS TO WH-PROCESS-NAME
  PERFORM WHICH-TRANSACTION
*
* Run the acquire HOLIDAY process
*
  EXEC CICS RUN ACQPROCESS
           ASYNCHRONOUS
           RESP(WS-RESP) RESP2(WS-RESP2)
  END-EXEC

```

Figure 40 (Part 1 of 2). Pseudo-Code for the Initiating Transaction

```

*
* Initialize the new personal details record with data from
* the control file
*
  MOVE MC-MORTGAGE-REF-KEY TO PD-MORTGAGE-REF-KEY
  INITIALIZE PD-MORTCOMP-DETAILS
  INITIALIZE PD-MORTGAGE-DETAILS
  MOVE MC-BANK-DETAILS TO PD-BANK-DETAILS
  MOVE MC-CUSTOMER-SALARY TO PD-SALARY
  MOVE MC-CUSTOMER-BALANCE TO PD-PERSONAL-BALANCE
  INITIALIZE PD-PAYMENT-HISTORY
*
* write the personal record
*
  EXEC CICS WRITE FILE(WS-FILE-NAME)
           FROM(PD-PERSONAL-DETAILS-RECORD)
           RIDFLD(PD-MORTGAGE-REF-KEY)
           RESP(WS-RESP) RESP2(WS-RESP2)
  END-EXEC
*
* Update the master control record in the control file
* by adding 1 to the next key field and re-write this
* record to the file.
*
  ADD 1 TO MORTGAGE-NEXT-KEY
  EXEC CICS REWRITE FILE(WS-CONTROL-FILE)
           FROM(WS-MASTER-CONTROL-RECORD)
           RESP(WS-RESP) RESP2(WS-RESP2)
  END-EXEC
*****
* EXEC CICS RETURN
*****
BZ-RETURN.
  EXEC CICS RETURN
  END-EXEC.
GOBACK.
*****
* Copybooks
*****
COPY WHICH.

```

Figure 40 (Part 2 of 2). Pseudo-Code for the Initiating Transaction

- EXEC CICS RUN ACQPROCESS ASYNCHRONOUS

This command executes the process. The ASYNCHRONOUS parameter means that the process will run independently of its initiator. However the process will not actually run until the initiator performs a SYNCPOINT. The initiator does not wait for the requested process to complete.

## 4.5 The Root Activity

The customer is unable to go on holiday if the flight or the hotel are unbooked after an initial attempt is made to book them. However, if the car or any of the excursions remain unbooked after their activities have been retried, the client can still go on holiday.

The root activity can be invoked by a number of events, but in all cases, this activity starts by reading the general container to obtain information as to which control and personal details file to read. It then retrieves the event that caused the activity to be attached.

## 4.5.1 BTS Resources Used

The root activity uses the following resources:

- Container RETRY I/O (process)
- Container GENERAL I/O (process)
- Container HOLIDAY I/O (process)
- Container BOOKFL I (child activity)
- Container BOOKHO I (child activity)
- Container BOOKCA I (child activity)
- Container BOOKEXCUR1 I (child activity)
- Container BOOKEXCUR2 I (child activity)
- Container BOOKEXCUR3 I (child activity)
- File MORTCONT I
- File PERSDET I/O
- Event DFHINITIAL (system)
- Event NO-RESPONSE (timer)
- Event BOOK-HOLIDAY (composite)
- Event FLIGHT-BOOK (completion)
- Event HOTEL-BOOK (completion)
- Event CAR-BOOK (completion)
- Event EX1-BOOK (completion)
- Event EX2-BOOK (completion)
- Event EX3-BOOK (completion)
- Event EX1-EVENT (completion)
- Event EX2-EVENT (completion)
- Event EX3-EVENT (completion)
- Event CANCEL-HOLIDAY (composite)
- Event FLIGHT-CANCEL (completion)
- Event HOTEL-CANCEL (completion)
- Event CAR-CANCEL (completion)
- Event EX1-CANCEL (completion)
- Event EX2-CANCEL (completion)
- Event EX3-CANCEL (completion)
- Activity BOOKFLIGHT (transaction HFLY)
- Activity BOOKHOTEL (transaction HHOT)
- Activity BOOKCAR (transaction HCAR)
- Activity BOOKEXCUR1 (transaction HEXC)
- Activity BOOKEXCUR2 (transaction HEXC)
- Activity BOOKEXCUR3 (transaction HEXC)
- Activity CANCELFLIGHT (transaction HCFL)

- Activity CANCELHOTEL (transaction HCHT)
- Activity CANCELCAR (transaction HCCR)
- Activity CANCELEXCUR1 (transaction HCEX)
- Activity CANCELEXCUR2 (transaction HCEX)
- Activity CANCELEXCUR3 (transaction HCEX)

## 4.5.2 System Event DFHINITIAL

Figure 41 shows the flow of the program when DFHINITIAL is the reattachment event.

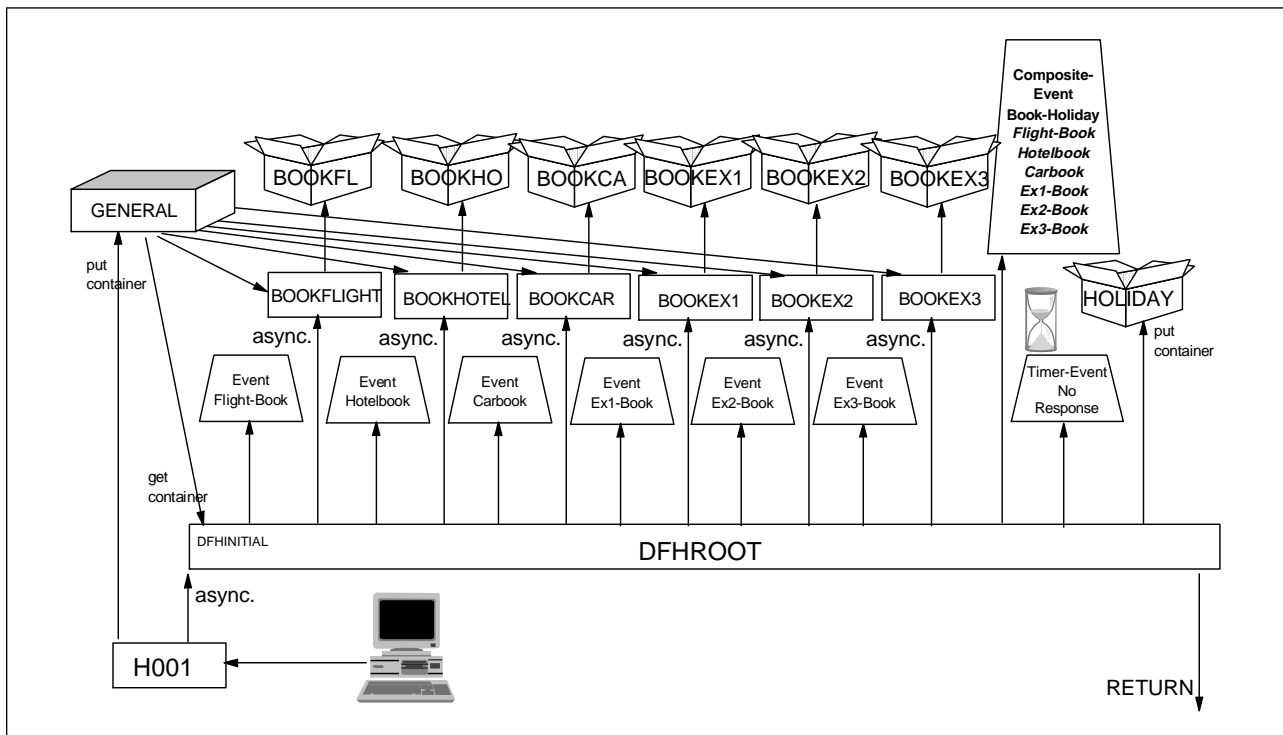


Figure 41. Flow of Program when Attached by DFHINITIAL

Define and run asynchronously the booking activities for all elements of the holiday. Define an AND composite event, BOOK-HOLIDAY, containing all the elements' completion events. Additionally, a NO-RESPONSE timer is defined. The NO-RESPONSE timer will be used if any of the booking events have not been able to complete within a given time and, therefore, the AND predicate has not become true.

For details of the business logic and program logic of the booking activities, see 4.6, "Booking Child Activities" on page 90.

Figure 42 on page 71 shows the pseudo-code when DFHINITIAL is the reattachment event.



```

AA-MAIN SECTION.
*
*  get the general container with the process name
*
EXEC CICS GET CONTAINER('GENERAL')
        INTO(C-GENERAL)
        PROCESS
        RESP(WS-RESP)
        RESP2(WS-RESP2)
END-EXEC
*
*  then read the control file
*
PERFORM READ-CONTROL-FILE
*
*  retrieve reattach event requests cics to tell it which
*  event has fired
*
EXEC CICS RETRIEVE REATTACH EVENT(WS-EVENT)
        RESP(WS-RESP) RESP2(WS-RESP2)
END-EXEC
*
*  Output to BMTD TD queue for diagnostics
*
PERFORM WHICH-TRANSACTION
*
*  The following evaluate uses the event from the retrieve reattach
*  to decide on the flow of the program.
*
EVALUATE TRUE
*
*  first time through
*
        WHEN DFH-INITIAL
*
*  define all the activities, add them to the composite 'AND'
*  event 'BOOK-HOLIDAY'
*  and run them all
*  then set up the no-response timer
*
        PERFORM DEFINE-ACTS-AND-ADD-SUBEVENTS
        PERFORM RUN-ACTIVITIES
        PERFORM SETUP-NO-RESPONSE-TIMER
*
*  return until the activities have done their bit or the
*  no response timer fires
*
        PERFORM HOLIDAY-RETURN
        WHEN . . . . .
END-EVALUATE.

```

Figure 42 (Part 1 of 3). Pseudo-Code when Attached by DFHINITIAL

```

*****
*   define the activities and add the subevents to the composite
*   event book-holiday
*****
:DEFINE-ACTS-AND-ADD-SUBEVENTS.
* bookflight
  EXEC CICS DEFINE ACTIVITY('BOOKFLIGHT')
            EVENT('FLIGHT-BOOK')
            TRANSID('HFLY')
            PROGRAM('HBOOKFLY')
            RESP(WS-RESP) RESP2(WS-RESP2)

  END-EXEC
* bookhotel
  EXEC CICS DEFINE ACTIVITY('BOOKHOTEL')
            EVENT('HOTEL-BOOK')
            TRANSID('HHOT')
            PROGRAM('HBOOKHOT')
            RESP(WS-RESP) RESP2(WS-RESP2)

  END-EXEC
* bookcar
  EXEC CICS DEFINE ACTIVITY('BOOKCAR')
            EVENT('CAR-BOOK')
            TRANSID('HCAR')
            PROGRAM('HBOOKCAR')
            RESP(WS-RESP) RESP2(WS-RESP2)

  END-EXEC
* bookex1
  EXEC CICS DEFINE ACTIVITY('BOOKEXCUR1')
            EVENT('EX1-BOOK')
            TRANSID('HEXC')
            PROGRAM('HBOOKEXC')
            RESP(WS-RESP) RESP2(WS-RESP2)

  END-EXEC
* bookex2
  EXEC CICS DEFINE ACTIVITY('BOOKEXCUR2')
            EVENT('EX2-BOOK')
            TRANSID('HEXC')
            PROGRAM('HBOOKEXC')
            RESP(WS-RESP) RESP2(WS-RESP2)

  END-EXEC
* bookex3
  EXEC CICS DEFINE ACTIVITY('BOOKEXCUR3')
            EVENT('EX3-BOOK')
            TRANSID('HEXC')
            PROGRAM('HBOOKEXC')
            RESP(WS-RESP) RESP2(WS-RESP2)

  END-EXEC
EXEC CICS DEFINE COMPOSITE
EVENT('BOOK-HOLIDAY')
AND
SUBEVENT1('FLIGHT-BOOK')
SUBEVENT2('HOTEL-BOOK')
SUBEVENT3('CAR-BOOK')
SUBEVENT4('EX1-BOOK')
SUBEVENT5('EX2-BOOK')
SUBEVENT6('EX3-BOOK')
RESP(WS-RESP) RESP2(WS-RESP2)

  END-EXEC

```

Figure 42 (Part 2 of 3). Pseudo-Code when Attached by DFHINITIAL

```

*****
* run-activities.
* run all the booking activities.
*****
RUN-ACTIVITIES.
  EXEC CICS RUN ACTIVITY('BOOKFLIGHT')
            ASYNCHRONOUS
            RESP(W5-RESP) RESP2(W5-RESP2)

  END-EXEC
  EXEC CICS RUN ACTIVITY('BOOKHOTEL')
            ASYNCHRONOUS
            RESP(W5-RESP) RESP2(W5-RESP2)

  END-EXEC
  EXEC CICS RUN ACTIVITY('BOOKCAR')
            ASYNCHRONOUS
            RESP(W5-RESP) RESP2(W5-RESP2)

  END-EXEC
  EXEC CICS RUN ACTIVITY('BOOKEXCUR1')
            ASYNCHRONOUS
            RESP(W5-RESP) RESP2(W5-RESP2)

  END-EXEC
  EXEC CICS RUN ACTIVITY('BOOKEXCUR2')
            ASYNCHRONOUS
            RESP(W5-RESP) RESP2(W5-RESP2)

  END-EXEC
  EXEC CICS RUN ACTIVITY('BOOKEXCUR3')
            ASYNCHRONOUS
            RESP(W5-RESP) RESP2(W5-RESP2)

  END-EXEC
*****
* setup the timer event no-response
*****
SETUP-NO-RESPONSE-TIMER.
  EXEC CICS DEFINE TIMER('NO-RESPONSE')
            EVENT('NO-RESPONSE')
            AFTER
            DAYS(W5-DAYS)
            HOURS(W5-HOURS)
            MINUTES(W5-MINUTES)
            SECONDS(W5-SECONDS)
            RESP(W5-RESP) RESP2(W5-RESP2)

  END-EXEC
*****
* put the holiday booked container and return
*****
HOLIDAY-RETURN.
  EXEC CICS PUT CONTAINER('HOLIDAY')
            FROM(C-HOLIDAY-EVENTS)
            PROCESS
            RESP(W5-RESP) RESP2(W5-RESP2)

  END-EXEC
  EXEC CICS RETURN END-EXEC.

```

Figure 42 (Part 3 of 3). Pseudo-Code when Attached by DFHINITIAL

The EXEC CICS DEFINE COMPOSITE EVENT('BOOK-HOLIDAY') AND SUBEVENT('FLIGHT-BOOK') SUBEVENT('HOTEL-BOOK') SUBEVENT('CAR-BOOK') SUBEVENT('EX1-BOOK') SUBEVENT('EX2-BOOK') SUBEVENT('EX3-BOOK') command defines a composite event and can specify up to eight sub-events to be added to a composite event when it is created. The sub-events that you specify must have previously been defined to the current activity by means of the DEFINE INPUT EVENT, DEFINE ACTIVITY or DEFINE TIMER commands. The sub-events must not be system events, composite events in their own right, or sub-events of existing composite events. Additionally, the sub-event cannot be an input event if the parameter AND is used for the composite event.

The EXEC CICS DEFINE TIMER('NO-RESPONSE') command is used as a safety net. It allows the root activity to be re-invoked if the BOOK-HOLIDAY composite does not fire within a reasonable time frame. We would recommend that a timer event, or an input event, be defined whenever a composite AND is defined.

### 4.5.3 Composite Event BOOK-HOLIDAY - No Retry Container

The root activity has been reattached before the NO-RESPONSE timer fired; so we must use the DELETE TIMER('NO-RESPONSE') command.

The program then tries to GET the retry container. Here, the program is using the existence or non-existence of the container to determine the flow of control of this invocation of the root activity. If the container does not exist, this is the first time the root activity has been invoked with the BOOK-HOLIDAY reattachment event.

Figure 43 shows the flow of the program when BOOK-HOLIDAY is the reattachment event and there is no retry container available.

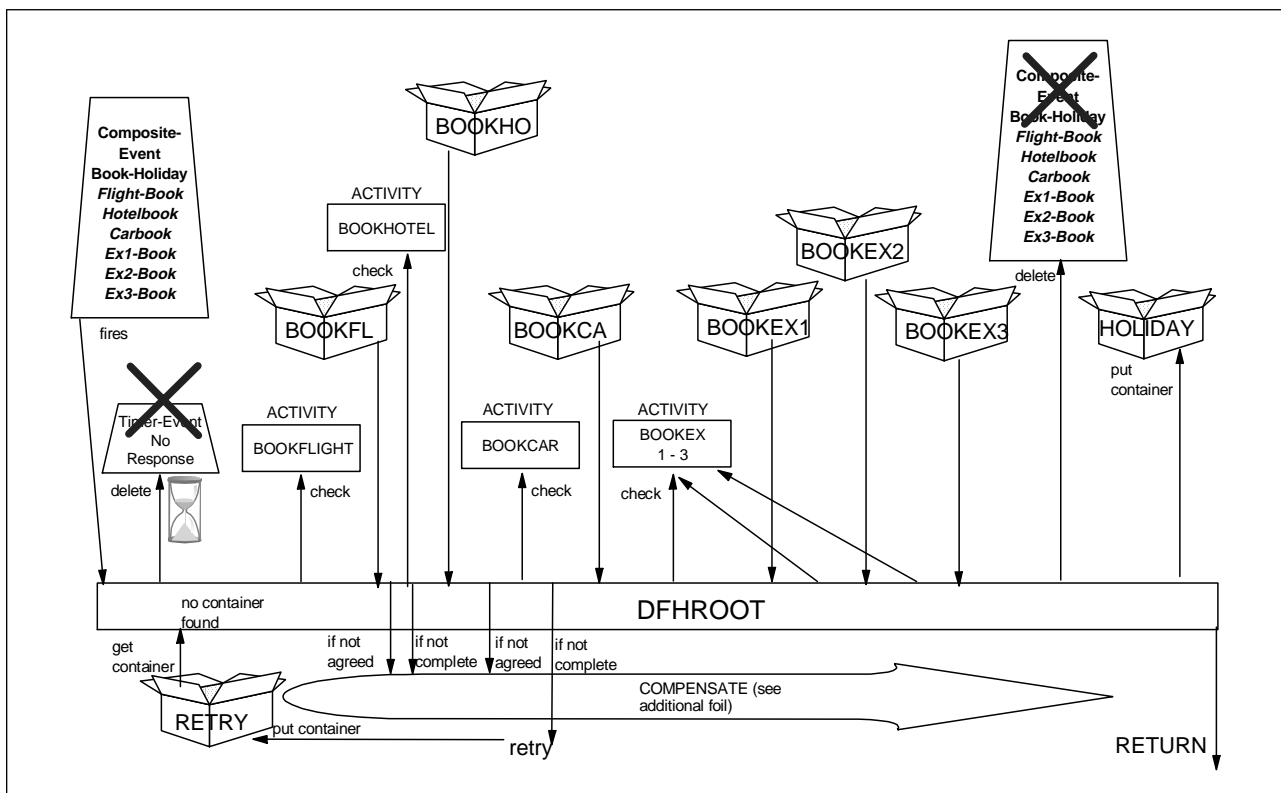


Figure 43. Flow of Program when Reattached by BOOK-HOLIDAY - No Retry Container

The child activities are checked one by one. If they completed without abending, they will have PUT the result of the booking into each of their own containers. The root activity can then GET this information. If all the bookings were successful, the root activity writes a message to a TD queue log and can issue a RETURN ENDACTIVITY command, and the process completes.

If the flight booking failed, either because of an abend of the child activity or the business logic preventing the booking of the flight, then any other elements that may have been successfully booked have to be reversed (or compensated). Similarly, if the hotel booking failed, then any other booked elements have to be compensated (see 4.5.7.1, "Compensating Activities" on page 87).

If the car booking failed, then we should RESET and retry the car booking and any failed excursion booking attempts. These retries will be executed asynchronously and the activities included once again in the BOOK-HOLIDAY composite event. The retry container is defined, so that when the root activity is again invoked, we know that we have already retried the activities. The NO-RESPONSE timer must be redefined in case these retries take too long to complete.

However, if the car booking did not abend, regardless of whether the booking was successful or not, the excursion activities are checked and retried if necessary. Here we can examine the difference between RUN ACTIVITY SYNCHRONOUS and LINK ACTIVITY commands.

Figure 44 on page 76 shows the pseudo-code when BOOK-HOLIDAY is the reattachment event and there is no retry container.

```

AA-MAIN SECTION.
*
* get the general container with the process name
*
EXEC CICS GET CONTAINER('GENERAL')
          INTO(C-GENERAL)
          PROCESS
          RESP(WS-RESP)
          RESP2(WS-RESP2)
END-EXEC
*
* then read the control file
*
PERFORM READ-CONTROL-FILE
*
* retrieve reattach event requests cics to tell it which
* event has fired
*
EXEC CICS RETRIEVE REATTACH EVENT(WS-EVENT)
          RESP(WS-RESP)
END-EXEC
*
* Output to BMTD TD queue for diagnostics
*
PERFORM WHICH-TRANSACTION
*
* The evaluate uses the retrieve reattach to decide on the
* flow of the program.
*
EVALUATE TRUE
*
* we have been reattached because book-holiday has fired
*
WHEN BOOK-HOLIDAY
*
* We have been reinvoked before the NO-RESPONSE timer fired,
* therefore it has to be deleted.
*
EXEC CICS DELETE TIMER('NO-RESPONSE')
          RESP(WS-RESP)
          RESP2(WS-RESP2)
END-EXEC
*
* This is not the first time we have run this program for this
* process. Find out if we are on a retry cycle
*
EXEC CICS GET CONTAINER('RETRY')
          INTO(C-RETRY)
          RESP(WS-RESP)
          RESP2(WS-RESP2)
END-EXEC
*
*

```

Figure 44 (Part 1 of 5). Pseudo-Code when Attached by BOOK-HOLIDAY - No Retry Container

```

        IF WS-RESP = DFHRESP(CONTAINERERR)
*
* We do not have a retry container.
* We have to find out whether flight has completed
        MOVE 'BOOKFLIGHT' TO ACTIVITY-NAME
        PERFORM CHECK-ACTIVITY
*
* If the flight activity has completed get the flight container
*
        IF WS-COMPLETE-OK = 'Y'
            PERFORM GET-FLIGHT-CONTAINER
*
* If the flight has not been booked we cannot go on
* holiday - compensate
*
        IF C-FLIGHT-AGREED = 'N'
            PERFORM COMPENSATE
        ELSE
*
* Now we have to find out whether hotel has completed
*
        MOVE 'BOOKHOTEL' TO ACTIVITY-NAME
        PERFORM CHECK-ACTIVITY
*
* If the hotel has not been booked we cannot go on
* holiday - compensate
*
        IF WS-COMPLETE-OK = 'Y'
            PERFORM GET-HOTEL-CONTAINER
            IF C-HOTEL-AGREED = 'N'
                PERFORM COMPENSATE
            ELSE
*
* Now we have to find out whether car has completed
*
        MOVE 'BOOKCAR' TO ACTIVITY-NAME
        PERFORM CHECK-ACTIVITY
        IF WS-COMPLETE-OK NOT = 'Y'
*
* the car hasn't completed ok so retry the car and the
* necessary excursions - asynchronous and return
*
            PERFORM RESET-RETRY-CAR-EXCURSIONS
            PERFORM SET-UP-RETRY-CONTAINER
            PERFORM SETUP-NO-RESPONSE-TIMER
            PERFORM HOLIDAY-RETURN
        ELSE
*
* if the car activity has completed ok
* check and retry 'sync' the excursions if necessary
*
            PERFORM GET-CAR-CONTAINER
            PERFORM CHECK-AND-RERUN-EXCURSIONS
*
* Then tidy up all events and children if necessary and
* complete with endactivity set.
*
            EXEC CICS DELETE EVENT(BOOK-HOLIDAY)
                RESP(WS-RESP) RESP2(WS-RESP2)
            END-EXEC
            PERFORM TIDY-UP-CHILDREN
            MOVE 'Y' TO C-HOLIDAY-BOOKED
            PERFORM HOLIDAY-DONE-RETURN
        END-IF
    END-IF
ELSE

```

Figure 44 (Part 2 of 5). Pseudo-Code when Attached by BOOK-HOLIDAY - No Retry Container

```

*
* the hotel hasn't worked so we need to compensate
*
          PERFORM COMPENSATE
        END-IF
      END-IF
    ELSE
*
* the flight hasn't worked so we need to compensate
*
          PERFORM COMPENSATE
        END-IF
      END-IF
    WHEN . . . .
*****
* check-and-rerun-excursions.
* check each excursion.  if it completed ok get the
* excursion container.
* if not retry the excursion activity
* either with a RUN SYNCHRONOUS or LINK command
*****
CHECK-AND-RERUN-EXCURSIONS.
*
* excursion 1
*
  MOVE 'BOOEXCUR1' TO ACTIVITY-NAME
  PERFORM CHECK-ACTIVITY
  IF WS-COMPLETE-OK = 'Y'
    PERFORM GET-EX1-CONTAINER
  ELSE
    EXEC CICS RESET ACTIVITY(ACTIVITY-NAME)
              RESP(WS-RESP) RESP2(WS-RESP2)
    END-EXEC
    IF MC-HOLIDAY-RUN-LINK = 'R'
      EXEC CICS RUN ACTIVITY(ACTIVITY-NAME)
                SYNCHRONOUS
                RESP(WS-RESP) RESP2(WS-RESP2)
      END-EXEC
      PERFORM CHECK-ACTIVITY
    ELSE
      EXEC CICS LINK ACTIVITY(ACTIVITY-NAME)
                RESP(WS-RESP) RESP2(WS-RESP2)
      END-EXEC
      PERFORM CHECK ACTIVITY
    END-IF
    IF WS-COMPLETE-OK = 'Y'
      PERFORM GET-EX1-CONTAINER
    END-IF
  END-IF.
* Repeat for the other 2 excursions.

```

Figure 44 (Part 3 of 5). Pseudo-Code when Attached by BOOK-HOLIDAY - No Retry Container



```

*****
* Re-run car and failing excursions
* Uses reset are run asynchronous within a composite
* event.
*****
RESET-RETRY-CAR-EXCURSIONS.

EXEC CICS RESET ACTIVITY('BOOKCAR')
      RESP(WS-RESP) RESP2(WS-RESP2)
END-EXEC
EXEC CICS ADD SUBEVENT('CAR-BOOK')
      EVENT('BOOK-HOLIDAY')
      RESP(WS-RESP) RESP2(WS-RESP2)
END-EXEC
EXEC CICS RUN ACTIVITY('BOOKCAR')
      ASYNCHRONOUS
      RESP(WS-RESP) RESP2(WS-RESP2)
END-EXEC
* check and rerun if necessary excursion 1
MOVE 'BOKEXCURI' TO ACTIVITY-NAME
PERFORM CHECK-ACTIVITY
IF WS-COMPLETE-OK = 'Y'
  PERFORM GET-EX1-CONTAINER
ELSE
  EXEC CICS RESET ACTIVITY(ACTIVITY-NAME)
        RESP(WS-RESP) RESP2(WS-RESP2)
  END-EXEC
  EXEC CICS ADD SUBEVENT('EX1-BOOK')
        EVENT('BOOK-HOLIDAY')
        RESP(WS-RESP) RESP2(WS-RESP2)
  END-EXEC
  EXEC CICS RUN ACTIVITY(ACTIVITY-NAME)
        ASYNCHRONOUS
        RESP(WS-RESP) RESP2(WS-RESP2)
  END-EXEC
END-IF
* Repeat for excursions 2 & 3.

*****
* get-element-container
* This is a generic GET, please substitute the appropriate
* activity name for element.
*****
GET-ELEMENT-CONTAINER.
  EXEC CICS GET CONTAINER(BOKELEMENT)
        INTO(C-BOKELEMENT)
        ACTIVITY(BOKELEMENT)
        RESP(WS-RESP)
        RESP2(WS-RESP2)
  END-EXEC
*
* prepare top copy the result to the holiday container
*
  MOVE C-ELEMENT-AGREED TO C-ELEMENT-BOOKED.
*****
* set up the retry container
*****
SET-UP-RETRY-CONTAINER.
  MOVE 1 TO C-RETRY-COUNT
  EXEC CICS PUT CONTAINER('RETRY')
        FROM(C-RETRY)
        RESP(WS-RESP)
        RESP2(WS-RESP2)
  END-EXEC.

```

Figure 44 (Part 4 of 5). Pseudo-Code when Attached by BOOK-HOLIDAY - No Retry Container

```

*****
* put the holiday booked container and return with endactivity
* set
*****
HOLIDAY-DONE-RETURN.
  IF HOLIDAY-BOOKED
    MOVE C-MORTGAGE-REF-KEY TO WS-MESSAGE-KEY
    IF C-CAR-BOOKED = 'Y'
      MOVE ', A CAR ' TO WS-CAR-MESSAGE
    END-IF
    IF EX1-BOOKED
      ADD 1 TO WS-NUMBER-OF-EXCURSIONS
    END-IF
    IF EX2-BOOKED
      ADD 1 TO WS-NUMBER-OF-EXCURSIONS
    END-IF
    IF EX3-BOOKED
      ADD 1 TO WS-NUMBER-OF-EXCURSIONS
    END-IF
  * A message saying your holiday has been booked
  EXEC CICS WRITEQ TD QUEUE('CSMT')
    FROM(WS-HOLIDAY-MESSAGE-LINE)
    RESP(WS-RESP)
    RESP2(WS-RESP2)
  END-EXEC
  ELSE
    MOVE ' SORRY - NO HOLIDAY FOR YOU ' TO WS-SORRY-MESSAGE
    MOVE C-MORTGAGE-REF-KEY TO WS-SORRY-KEY
    EXEC CICS WRITEQ TD QUEUE('CSMT')
      FROM(WS-SORRY-MESSAGE-LINE)
      RESP(WS-RESP)
      RESP2(WS-RESP2)
    END-EXEC
  END-IF
  EXEC CICS RETURN
    ENDACTIVITY
  END-EXEC.
*****
* put the holiday booked container and return
* this is state date for the next reattachment of the
* root activity.
*****
HOLIDAY-RETURN.

  EXEC CICS PUT CONTAINER('HOLIDAY')
    FROM(C-HOLIDAY-EVENTS)
    PROCESS
    RESP(WS-RESP)
    RESP2(WS-RESP2)

  END-EXEC
  EXEC CICS RETURN END-EXEC.

```

Figure 44 (Part 5 of 5). Pseudo-Code when Attached by BOOK-HOLIDAY - No Retry Container

- EXEC CICS GET CONTAINER('RETRY') (implied ACTIVITY)

In this example, we used the values in the RESP fields to determine the flow of the program. A possible error response from a GET CONTAINER command is CONTAINERERR. The only RESP2 value assigned to CONTAINERERR at this time is 10. This means that the named container could not be found, and, therefore, a previous PUT CONTAINER had not been issued.

We have used an implied ACTIVITY CONTAINER as there is no need for any other activities within the process to read this container. For the root activity, only the root itself can read its own activity container.

- EXEC CICS GET CONTAINER(CHILD-ACTIVITY)  
ACTIVITY(CHILD-ACTIVITY-NAME)

A parent activity can GET the containers of its children by adding the parameter ACTIVITY(CHILD-ACTIVITY-NAME) to the command. In this way, a child and its parent can communicate in both directions.

- EXEC CICS LINK ACTIVITY('BOOEEXCURx')

EXEC CICS RUN ACTIVITY('BOOEEXCURx') SYNCHRONOUS

When a LINK command is issued, the requesting activity and the requested activity run in the same UOW. Therefore, any failure will cause both activities to back out.

When the RUN SYNCHRONOUS command is issued, the two activities are executed in separate UOWs, although both are committed when the requestor's UOW is committed using the existing CICS two-phase commit protocol.

We can apply either of these methods to the HOLIDAY application, depending upon the business rules that should be applied. If the excursions to be booked are independent of each other, we suggest using the RUN ACTIVITY SYNCHRONOUS command. This means that if one of the booking activities abends and needs to back out, the others can continue to be committed. However, for example, if excursions 2 and 3 can only be booked if excursion 1 did not abend, we suggest using the LINK ACTIVITY command.

- EXEC CICS PUT CONTAINER('HOLIDAY') PROCESS

The HOLIDAY process container stores information about all parts of the HOLIDAY application and whether they have been booked or not. We chose a PROCESS CONTAINER in case we needed this information to be read by activities other than the root activity.

#### 4.5.4 Composite Event BOOK-HOLIDAY - With Retry Container

Figure 45 on page 82 shows the flow of the program when BOOK-HOLIDAY is the reattachment event and there is a retry container available.

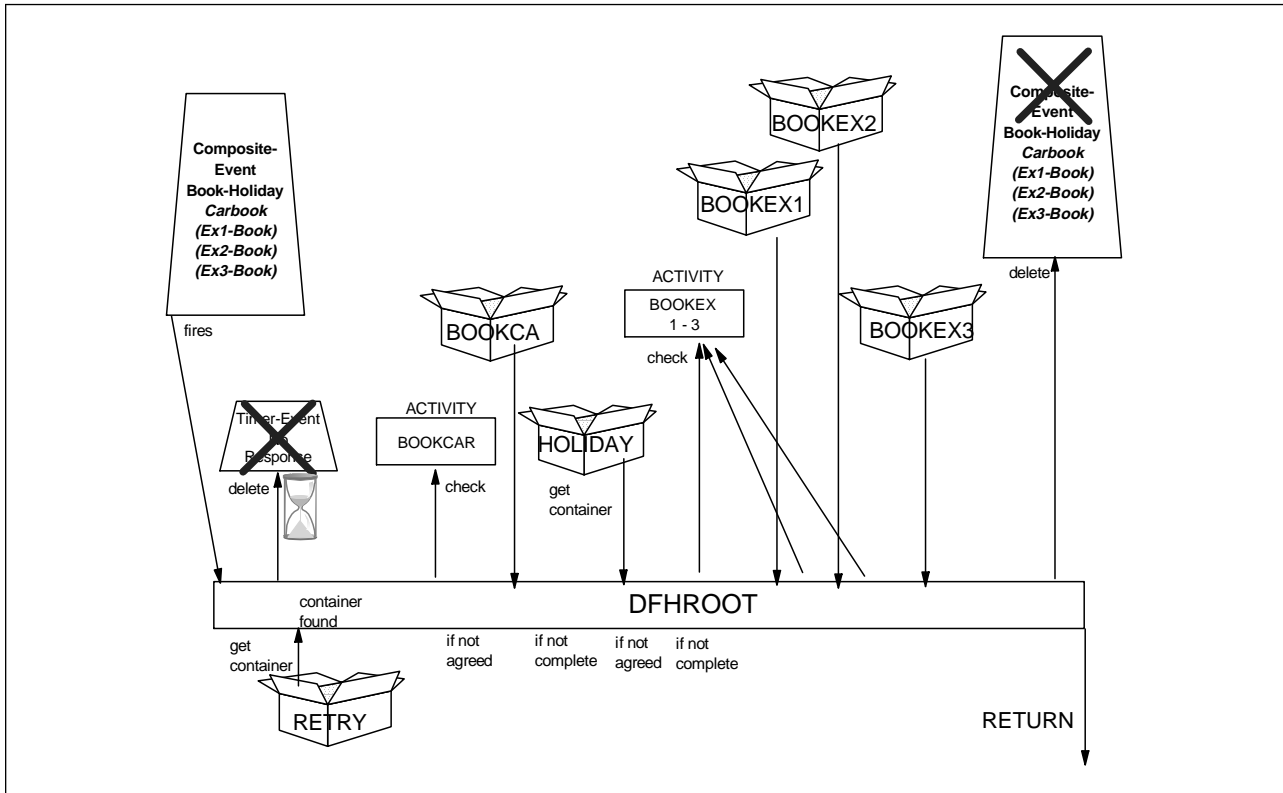


Figure 45. Flow of Program when Reattached by BOOK-HOLIDAY - With Retry Container

The root activity has been reattached before the NO-RESPONSE timer fired; so, we must use the DELETE TIMER('NO-RESPONSE') command.

As we are being attached with a retry container, only the car and possibly an excursion will have been included in the composite. We used the container HOLIDAY to decide which excursion to check. However, we could have used the RETRIEVE SUBEVENT command.

Check that the car activity and any excursions ran successfully and report back the outcome in the CSMT TD queue.

#### 4.5.5 Timer Event NO-RESPONSE

Figure 46 on page 83 shows the flow of the program when NO-RESPONSE is the reattachment event.

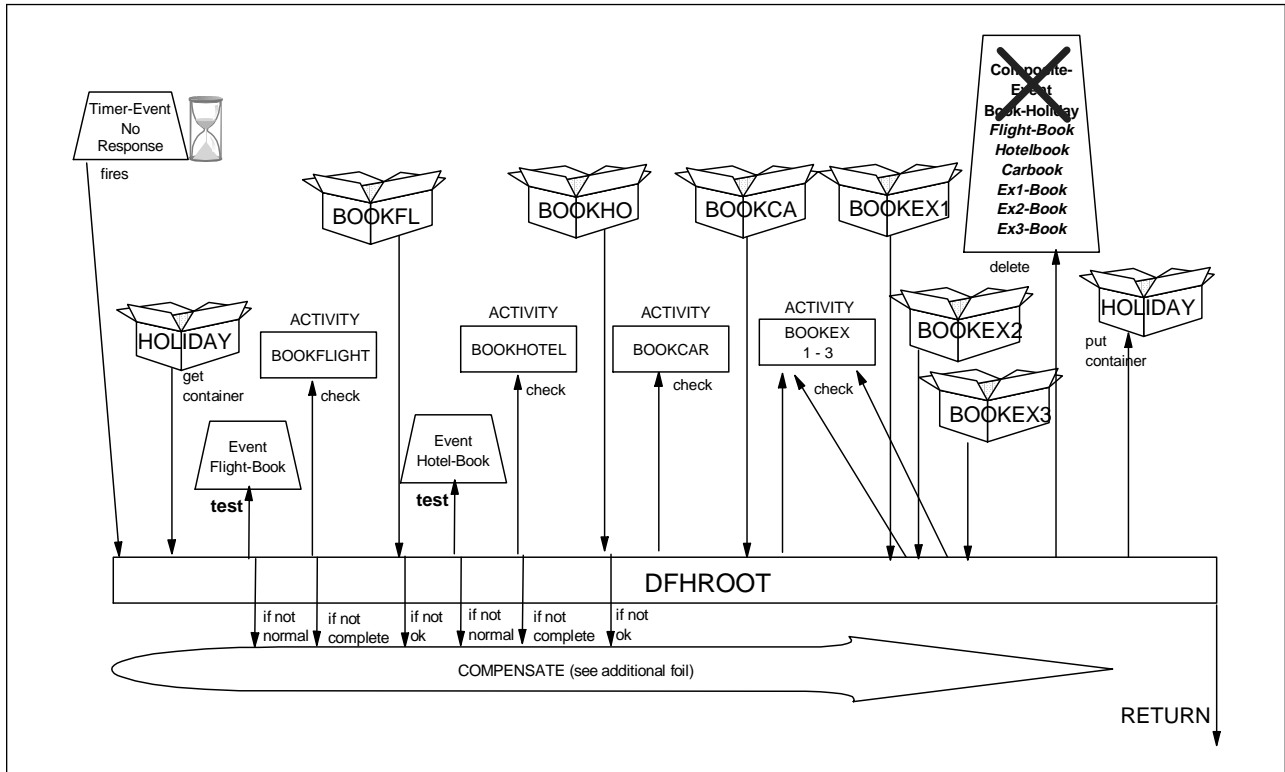


Figure 46. Flow of Program when Reattached by NO-RESPONSE

We test whether the flight or hotel event has fired. If not, there can be no holiday, and any booked elements must be compensated (see 4.5.7.1, “Compensating Activities” on page 87). Otherwise, we check the car and excursions activities, delete the BOOK-HOLIDAY composite event, and then tidy up all the children (see 4.5.7.2, “Tidy Up Children” on page 88 for a description of the tidy up routine.)

The result is reported back in the CSMT TD queue and RETURN ENDACTIVITY command is executed.

Figure 47 on page 84 shows the pseudo-code when NO-RESPONSE is the reattachment event.

```

AA-MAIN SECTION.
*
* get the general container with the process name
*
EXEC CICS GET CONTAINER('GENERAL')
              INTO(C-GENERAL)
              PROCESS
              RESP(WS-RESP)
              RESP2(WS-RESP2)
END-EXEC
*
* then read the control file
*
PERFORM READ-CONTROL-FILE
*
* retrieve reattach event requests cics to tell it which
* event has fired
*
EXEC CICS RETRIEVE REATTACH EVENT(WS-EVENT)
              RESP(WS-RESP)
END-EXEC
*
* Output to a TD queue for diagnostics
*
PERFORM WHICH-TRANSACTION
*
* The evaluate uses the retrieve reattach to decide on the
* flow of the program.
*
EVALUATE TRUE
*
* we have been invoked with a no response timer event
*
WHEN NO-RESPONSE
  PERFORM GET-HOLIDAY-CONTAINER
*
* if the flight has not already been booked
* test whether it has fired.
* If the test fails compensate.
*
  IF C-FLIGHT-BOOKED = 'N'
    EXEC CICS TEST EVENT('FLIGHT-BOOK')
              FIRESTATUS(WS-FIRE-STATUS)
              RESP(WS-RESP)
              RESP2(WS-RESP2)
    END-EXEC
    IF WS-RESP NOT = DFHRESP(NORMAL)
      PERFORM COMPENSATE
    ELSE

```

Figure 47 (Part 1 of 3). Pseudo-Code when Attached by NO-RESPONSE

```

*
* if flight has not fired, compensate any booked elements
* else check the flight activity.
*
      IF WS-FIRE-STATUS NOT = DFHVALUE(FIRED)
        PERFORM COMPENSATE
      ELSE
        MOVE 'BOOKFLIGHT' TO ACTIVITY-NAME
        PERFORM CHECK-ACTIVITY
*
* If the activity completed ok, get its container
*
      IF WS-COMPLETE-OK = 'Y'
        PERFORM GET-FLIGHT-CONTAINER
*
* If the flight was not booked, then compensate any booked
* elements.
*
      IF C-FLIGHT-AGREED = 'N'
        PERFORM COMPENSATE
      END-IF
*
* The activity didn't complete ok, then compensate any booked
* elements.
*
      ELSE
        PERFORM COMPENSATE
      END-IF
    END-IF
  END-IF
* Now do the same as the above for the hotel.
* That is if the hotel has not already been booked
* test whether it has fired.
      PERFORM CHECK-CAR-AND-EXCURS
*
* Both flight and hotel have been booked. Check the status of
* the car and excursion elements, although if they have not been
* booked the holiday can still go ahead.
* Tidy up and return with endactivity.
*
      MOVE 'BOOK-HOLIDAY' TO EVENT-NAME
      PERFORM EVENT-DELETE
      PERFORM TIDY-UP-CHILDREN
      MOVE 'Y' TO C-HOLIDAY-BOOKED
      PERFORM HOLIDAY-DONE-RETURN

CHECK-CAR-AND-EXCURS.
* Perform a CHECK ACTIVITY on bookcar and all bookexcurs
COMPENSATE.

```

Figure 47 (Part 2 of 3). Pseudo-Code when Attached by NO-RESPONSE

```

HOLIDAY-DONE-RETURN.
*****
* put the holiday booked container and return with endactivity
* set
*****
  IF HOLIDAY-BOOKED
    MOVE C-MORTGAGE-REF-KEY TO WS-MESSAGE-KEY
    IF C-CAR-BOOKED = 'Y'
      MOVE ', A CAR ' TO WS-CAR-MESSAGE
    END-IF
    IF EX1-BOOKED
      ADD 1 TO WS-NUMBER-OF-EXCURSIONS
    END-IF
    IF EX2-BOOKED
      ADD 1 TO WS-NUMBER-OF-EXCURSIONS
    END-IF
    IF EX3-BOOKED
      ADD 1 TO WS-NUMBER-OF-EXCURSIONS
    END-IF
    EXEC CICS WRITEQ TD QUEUE('CSMT')
           FROM(WS-HOLIDAY-MESSAGE-LINE)
           RESP(WS-RESP)
           RESP2(WS-RESP2)
    END-EXEC
  ELSE
    MOVE ' SORRY - NO HOLIDAY FOR YOU ' TO WS-SORRY-MESSAGE
    MOVE C-MORTGAGE-REF-KEY TO WS-SORRY-KEY
    EXEC CICS WRITEQ TD QUEUE('CSMT')
           FROM(WS-SORRY-MESSAGE-LINE)
           RESP(WS-RESP)
           RESP2(WS-RESP2)
    END-EXEC
  END-IF
EXEC CICS RETURN
      ENDACTIVITY
END-EXEC.

```

Figure 47 (Part 3 of 3). Pseudo-Code when Attached by NO-RESPONSE

The EXEC CICS TEST EVENT('FLIGHT-BOOKED') command tests whether a named event has occurred. We used this command in this example to determine whether the two most important activities had fired their completion events. If either of these events have not fired, it is necessary to compensate any elements that have been booked.

#### 4.5.6 Composite Event CANCEL-HOLIDAY

Figure 48 on page 87 shows the flow of the program when CANCEL-HOLIDAY is the reattachment event.



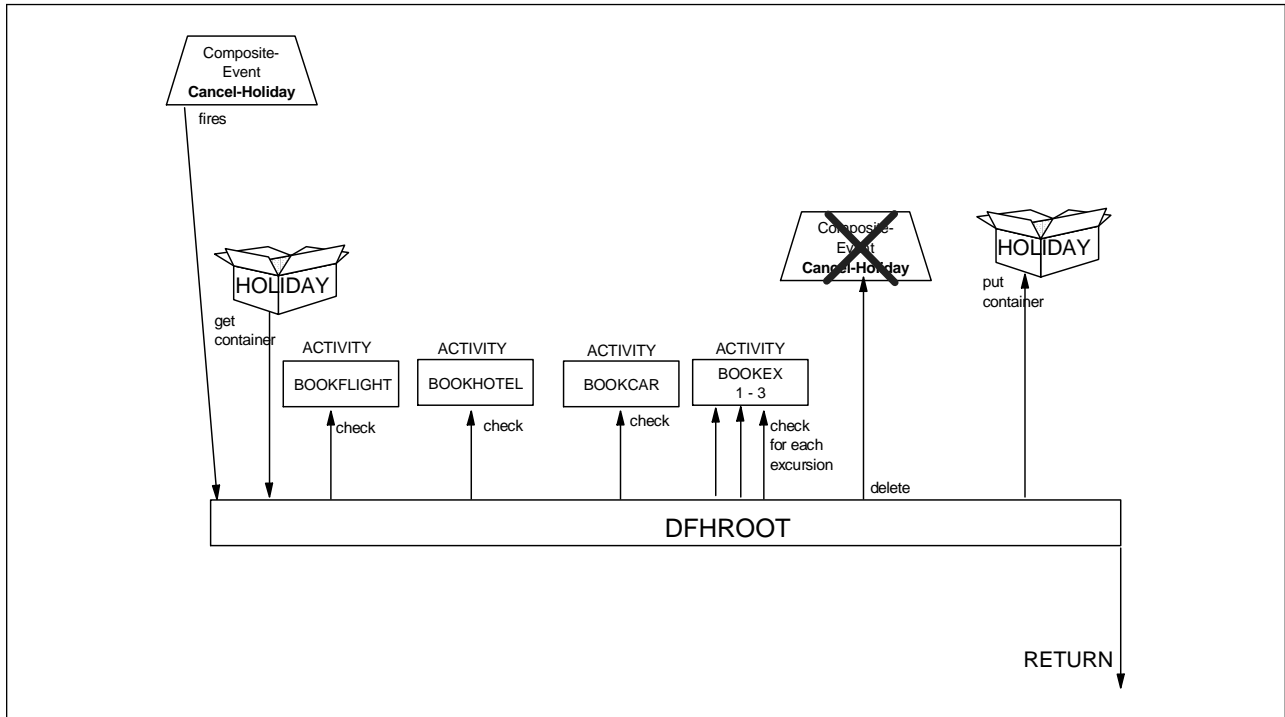


Figure 48. Flow of Program when Reattached by CANCEL-HOLIDAY

All the activities that reverse any previous booking have completed. The child activities are checked, and then we used a tidy up routine to ensure that any outstanding children and events are deleted. Finally, the root activity records in the CSMT TD queue the outcome and the RETURN ENDACTIVITY command is executed.

## 4.5.7 Other Functionality

Other functions of the program are:

### 4.5.7.1 Compensating Activities

Compensation is discussed in 4.1, "Compensation" on page 59. Here, we show the way it is used within the HOLIDAY application.

Figure 49 on page 88 shows the flow of the program when there is a need to compensate and cancel the holiday.

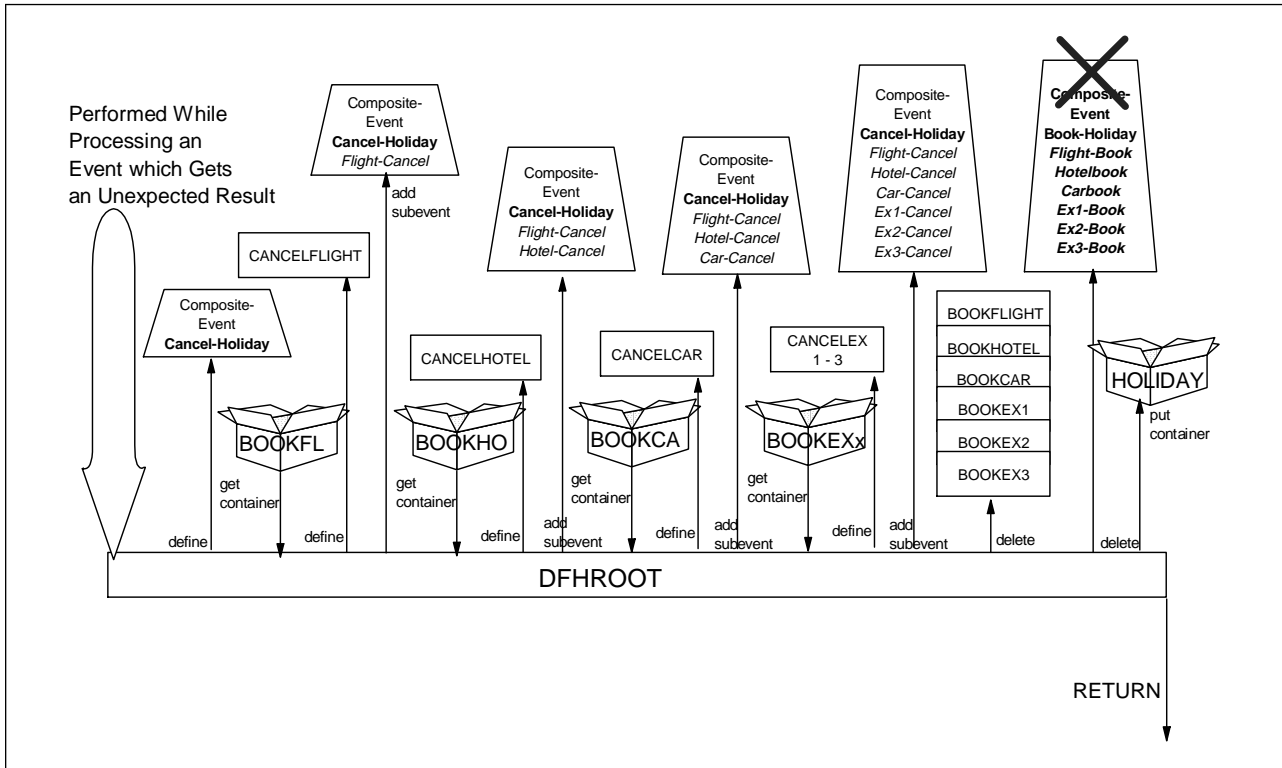


Figure 49. Flow of Program when There Is a Need to COMPENSATE

Define a new composite AND event, CANCEL-HOLIDAY, then define and run cancellation activities for any elements of the holiday that have been booked but now must have their booking reversed. The holiday container has information about activities that completed successfully, and the element was booked. Each booking then has to be reversed, and the money refunded to the customer's personal balance.

The cancellation activities show how application programs have to be designed to reverse the action of any completed activity.

#### 4.5.7.2 Tidy Up Children

Browse through all activities that are associated with the process to obtain information about their events. This is an application programmer's way of obtaining the information that is available through the CICS-supplied transaction CBAM. There should be no need to delete any of the components during the tidy up phase as the individual activities have been designed to tidy up themselves. Additionally, CICS BTS will discard any components of a process when the root activity issues a RETURN ENDACTIVITY command, and there are no outstanding child completion events. However, this routine could be used when you test an application to determine which outstanding event may have caused the RETURN ENDACTIVITY command to abend.

Figure 50 on page 89 shows the pseudo-code for TIDY-UP-CHILDREN.

```

TIDY-UP-CHILDREN.
EXEC CICS STARTBROWSE ACTIVITY
      BROWSETOKEN(B-ACTIVITY)
END-EXEC
do until Resp = END
EXEC CICS GETNEXT ACTIVITY(WS-ACTIVITY-NAME)
      ACTIVITYID(WS-ACTIVITYID)
      BROWSETOKEN(B-ACTIVITY)
END-EXEC
EXEC CICS STARTBROWSE EVENT
      ACTIVITYID(WS-ACTIVITYID)
      BROWSETOKEN(B-EVENT)
END-EXEC
do until Resp = END
EXEC CICS GETNEXT EVENT(WS-EVENT-NAME)
      BROWSETOKEN(B-EVENT)
      EVENTTYPE(WS-EVENTTYPE)
END-EXEC
IF WS-EVENTTYPE is a completion event then
  put out some sort of meaningful error message
END-IF
end-do
EXEC CICS ENDBROWSE EVENT
      BROWSETOKEN(B-EVENT)
END-EXEC
end-do
EXEC CICS DELETE ACTIVITY(WS-ACTIVITY-NAME)
END-EXEC
EXEC CICS ENDBROWSE ACTIVITY
      BROWSETOKEN(B-ACTIVITY)
END-EXEC.

```

Figure 50. Pseudo-Code for TIDY-UP-CHILDREN

- EXEC CICS STARTBROWSE ACTIVITY BROWSETOKEN(B-ACTIVITY)

The STARTBROWSE command is used to identify either child activities of an activity or descendant activities of a specific process. In this case, we know that the root activity has only one level of descendant activities; therefore, there is no need to include any other parameters on the command. If, however, the child activities have children themselves, the command could be issued with the PROCESS and PROCESSTYPE parameters, then all activities within the process can be browsed. Also see the GETNEXT ACTIVITY command below. You could also specify the ACTIVITYID parameter. This narrows the browse to only children of the activity that is identified by the ACTIVITYID.

- EXEC CICS GETNEXT ACTIVITY(WS-ACTIVITY-NAME)  
ACTIVITYID(WS-ACTIVITYID) BROWSETOKEN(B-ACTIVITY)

In this case, the command can return the name and the identifier of the next child of an activity. This is because we have omitted the PROCESS and PROCESSTYPE parameters from the STARTBROWSE command. Had we included these parameters, the command would return the next descendant activity of a process. The parameter LEVEL would then return the depth in the activity-tree at which the next activity resides. A value of '0' indicates the root activity, '1' indicates a child of the root, '2' indicates a child of a child of the root, and so on.

- EXEC CICS STARTBROWSE EVENT ACTIVITYID(WS-ACTIVITYID)  
BROWSETOKEN(B-EVENT)

The STARTBROWSE EVENT command initializes a browse token to be used to identify events within the scope of an activity. If the ACTIVITYID parameter is omitted, only the current activities are browsed. However, in this case, we

are browsing child activity events; therefore, we use the ACTIVITYID returned on the previous GETNEXT ACTIVITY command. The browse will find atomic events, which may or may not be sub-events, composite events, and system events.

- EXEC CICS GETNEXT EVENT(EVENT-NAME) BROWSETOKEN(B-EVENT)  
This command returns a lot of useful information about an event. We can find out the event type - activity completion, composite, input, system, or timer. If the event is a sub-event, the COMPOSITE parameter will return the composite event this sub-event is associated with. The PREDICATE and FIRESTATUS parameters return FIRED/NOTFIRED and AND/OR, respectively, and finally, TIMER returns the associated timer if the event is a timer event.
- EXEC CICS ENDBROWSE EVENT BROWSETOKEN(B-EVENT)  
ENDBROWSE EVENT ends a browse of the events with the scope of the activity and invalidates the browse token.
- EXEC CICS DELETE ACTIVITY(WS-ACTIVITY-NAME)  
This command is used to remove a child activity from the BTS repository data set. The child activity's completion event is also removed from the parent's event pool, and any descendant of the child activity is also deleted. Again, this is useful during testing to enable the process to complete, but we would advise that activities be allowed to complete normally. To be eligible for deletion, the child activity must be either in a complete, dormant, or an initial state (mode). Use the CHECK ACTIVITY command to determine the current state of an activity.
- EXEC CICS ENDBROWSE ACTIVITY BROWSETOKEN(B-ACTIVITY)  
This command ends the browse of the child activities of an activity and invalidates the browse token.

For more examples of using the browse facility refer to *CICS Business Transaction Services, SC34-5268*.

---

## 4.6 Booking Child Activities

In this section we describe various booking activities.

### 4.6.1 BTS Resources Used

Booking activities use various resources.

BOOKFLIGHT uses the following resources:

- Container GENERAL            I            (process)
- Container BOOKFLIGHT        0            (activity)
- File MORTCONT                I
- File PERSDET                  0

BOOKHOTEL uses the following resources:

- Container GENERAL            I            (process)
- Container BOOKHOTEL         0            (activity)
- File MORTCONT                I

- File PERSDET 0

BOOKCAR uses the following resources:

- Container GENERAL I (process)
- Container BOOKCAR 0 (activity)
- File MORTCONT I
- File PERSDET 0

BOOKEXCURx uses the following resources:

- Container GENERAL I (process)
- Container BOOKEXCURx 0 (activity)
- File MORTCONT I
- File PERSDET 0

## 4.6.2 System Event DFHINITIAL

All the booking activities have the same logic flow as follows:

- Read the control file record for element availability and cost with the key from the general container.
- If testing the NO-RESPONSE timer, then delay and return.
- If the element is not available, report this back to the root activity in the BOOK container and return.
- If the element is available, read for update the personal details file and subtract the cost from the personal balance.
  - If this leaves a negative balance, unlock the file record.
  - If this leaves a positive balance, rewrite the record.

Report the outcome back to the root activity in the container and return.

Figure 51 on page 92 shows the pseudo-code for the child activity that books an element.

```

PROCEDURE DIVISION.
AA-MAIN SECTION.
  EXEC CICS RETRIEVE REATTACH EVENT(WS-EVENT)
    RESP(WS-RESP) RESP2(WS-RESP2)
  END-EXEC

* Get the general process container which includes the key of the record
* to read and the control file details

  EXEC CICS GET CONTAINER('GENERAL')
    INTO(C-GENERAL)
    PROCESS
    RESP(WS-RESP) RESP2(WS-RESP2)
  END-EXEC

* Find out which activity this program is executing on behalf of.
* The activity name is also the name of the activity container used
* to pass information back to the root activity.

  EXEC CICS ASSIGN ACTIVITY(WS-ACTIVITY)
    RESP(WS-RESP) RESP2(WS-RESP2)
  END-EXEC

* Read the control file with the key from the general container

  EXEC CICS READ FILE(C-CONTROL-FILE-NAME)
    INTO(MC-MORTGAGE-CONTROL-RECORD)
    RIDFLD(C-MORTGAGE-REF-KEY)
    RESP(WS-RESP) RESP2(WS-RESP2)
  END-EXEC

* There would be code in here if the test was for the NO-RESPONSE
* timer. This would delay the completion of this activity until
* after the parent's NO-RESPONSE timer had fired

* If the control file requires - this element will not be booked
* PUT the activity's container with 'N' in WS-ELEMENT-AGREED
* and RETURN with ENDACTIVITY set.

  IF (element not available)
    MOVE 'N' TO C-ELEMENT-AGREED
    PERFORM PUT-ELEMENT-CONTAINER
    EXEC CICS RETURN
      ENDACTIVITY
    END-EXEC
  END-IF

* If the control file requires - this element can be booked
* unless there are not enough funds in the personal balance of
* then personal details file to pay for the element.

  EXEC CICS READ FILE(WS-FILE-NAME)
    INTO(PD-FILE-RECORD)
    RIDFLD(C-MORTGAGE-REF-KEY)
    UPDATE
    RESP(WS-RESP) RESP2(WS-RESP2)
  END-EXEC

* Subtract the cost of the element from the customer's personal balance

  SUBTRACT MC-ELEMENT-COST FROM PD-PERSONAL-BALANCE

```

Figure 51 (Part 1 of 2). Pseudo-Code for Booking an Element

```

* If the personal balance is now less than zero, unlock the
* personal details file.

* PUT the activity's container with 'N' in WS-ELEMENT-AGREED
* and RETURN with ENDACTIVITY set.
  IF PD-PERSONAL-BALANCE < ZERO
    EXEC CICS UNLOCK FILE(WS-FILE-NAME)
      RESP(WS-RESP) RESP2(WS-RESP2)
    END-EXEC
    MOVE 'N' TO C-ELEMENT-AGREED
    PERFORM PUT-ELEMENT-CONTAINER
    EXEC CICS RETURN
      ENDACTIVITY
    END-EXEC
  ELSE

* If the personal balance is OK, rewrite the personal details file.

* PUT the activity's container with 'Y' in WS-ELEMENT-AGREED
* and RETURN with ENDACTIVITY set.

    EXEC CICS REWRITE FILE(WS-FILE-NAME)
      FROM(PD-FILE-RECORD)
      RESP(WS-RESP) RESP2(WS-RESP2)
    END-EXEC
    MOVE 'Y' TO C-ELEMENT-AGREED
    PERFORM PUT-ELEMENT-CONTAINER
    EXEC CICS RETURN
      ENDACTIVITY
    END-EXEC
  END-IF.

PUT-ELEMENT-CONTAINER.

  EXEC CICS PUT CONTAINER(WS-ACTIVITY)
    FROM(C-BOOELEMENT)
    RESP(WS-RESP) RESP2(WS-RESP2)
  END-EXEC.

```

Figure 51 (Part 2 of 2). Pseudo-Code for Booking an Element

## 4.7 Compensating Child Activities

In this section we describe the child activities that are used to reverse the booking actions.

### 4.7.1 BTS Resources Used

Compensation activities use various resources.

CANCELFLIGHT uses the following resources:

- Container GENERAL            I            (process)
- File MORTCONT                I
- File PERSDET                 0

CANCELHOTEL uses the following resources:

- Container GENERAL            I            (process)
- File MORTCONT                I
- File PERSDET                 0

CANCELCAR uses the following resources:

- Container GENERAL I (process)
- File MORTCONT I
- File PERSDET 0

CANCELEXCURx uses the following resources:

- Container GENERAL I (process)
- File MORTCONT I
- File PERSDET 0

## 4.7.2 System Event DFHINITIAL

All the cancelling activities have the same logic flow as follows:

- Read the control file record for element's cost with the key from the general container.
- Read for update the personal details file and add the element's cost to the personal balance.
- Return.

Figure 52 on page 95 shows the pseudo-code for child activity that reverses the element booking.



```

PROCEDURE DIVISION.
AA-MAIN SECTION.
  EXEC CICS RETRIEVE REATTACH EVENT(WS-EVENT)
    RESP(WS-RESP) RESP2(WS-RESP2)
  END-EXEC

* Get the general process container which includes the key of the record
* to read and the control file details

  EXEC CICS GET CONTAINER('GENERAL')
    INTO(C-GENERAL)
    PROCESS
    RESP(WS-RESP) RESP2(WS-RESP2)
  END-EXEC

* Find out which activity this program is executing on behalf of.

  EXEC CICS ASSIGN ACTIVITY(WS-ACTIVITY)
    RESP(WS-RESP) RESP2(WS-RESP2)
  END-EXEC

* Read the control file with the key from the general container

  EXEC CICS READ FILE(C-CONTROL-FILE-NAME)
    INTO(MC-MORTGAGE-CONTROL-RECORD)
    RIDFLD(C-MORTGAGE-REF-KEY)
    RESP(WS-RESP) RESP2(WS-RESP2)
  END-EXEC

* Read the personal details file with the key from the general container

  EXEC CICS READ FILE(WS-FILE-NAME)
    INTO(PD-FILE-RECORD)
    RIDFLD(C-MORTGAGE-REF-KEY)
    UPDATE
    RESP(WS-RESP) RESP2(WS-RESP2)
  END-EXEC

* Add back the cost of the element from the customer's personal balance

  ADD MC-ELEMENT-COST FROM PD-PERSONAL-BALANCE

  EXEC CICS REWRITE FILE(WS-FILE-NAME)
    FROM(PD-FILE-RECORD)
    RESP(WS-RESP) RESP2(WS-RESP2)
  END-EXEC

```

Figure 52. Pseudo-Code for Cancelling a Booking

## 4.8 TD Output, Audit Log, and CBAM Output

The flow of a HOLIDAY business transaction can be followed in a number of ways.

- The output to the TD queue that is mapped by the WHICH copybook
- The audit log output if auditing is switched on
- The CBAM displays to see the state of the repository

## 4.8.1 The TD Queue Output

Figure 53 shows the output produced by the activities to the TD queue for a HOLIDAY application where all the elements, except the car, have been booked.

```
0000147 H001      ** RUNNING;HOLI001  27-02-99  00:48:54  PRC;HOLI000000003
          ACTV;                EVENT;                SUB:
0000148 HHOL    BAMSHRF ** RUNNING;HHOLIDAY  27-02-99  00:48:54  PRC;HOLI000000003
          ACTV;DFHROOT        EVENT;DFHINITIAL    SUB:
0000149 HFLY    BAMSHRF ** RUNNING;HBOOKFLY  27-02-99  00:48:54  PRC;HOLI000000003
          ACTV;BOOKFLIGHT     EVENT;DFHINITIAL    SUB:
0000154 HEXC    BAMSHRF ** RUNNING;HBOOKEXC  27-02-99  00:48:54  PRC;HOLI000000003
          ACTV;BOOKEXCUR3     EVENT;DFHINITIAL    SUB:
0000153 HEXC    BAMSHRF ** RUNNING;HBOOKEXC  27-02-99  00:48:54  PRC;HOLI000000003
          ACTV;BOOKEXCUR2     EVENT;DFHINITIAL    SUB:
0000150 HHOT    BAMSHRF ** RUNNING;HBOOKHOT  27-02-99  00:48:54  PRC;HOLI000000003
          ACTV;BOOKHOTEL      EVENT;DFHINITIAL    SUB:
0000151 HCAR    BAMSHRF ** RUNNING;HBOOKCAR  27-02-99  00:48:54  PRC;HOLI000000003
          ACTV;BOOKCAR        EVENT;DFHINITIAL    SUB:
0000152 HEXC    BAMSHRF ** RUNNING;HBOOKEXC  27-02-99  00:48:54  PRC;HOLI000000003
          ACTV;BOOKEXCUR1     EVENT;DFHINITIAL    SUB:
0000160 HHOL    BAMSHRF ** RUNNING;HHOLIDAY  27-02-99  00:48:57  PRC;HOLI000000003
          ACTV;DFHROOT        EVENT;BOOK-HOLIDAY  SUB:
0000161 HCAR    BAMSHRF ** RUNNING;HBOOKCAR  27-02-99  00:48:57  PRC;HOLI000000003
          ACTV;BOOKCAR        EVENT;DFHINITIAL    SUB:
0000162 HHOL    BAMSHRF ** RUNNING;HHOLIDAY  27-02-99  00:48:57  PRC;HOLI000000003
          ACTV;DFHROOT        EVENT;BOOK-HOLIDAY  SUB:
000000003 WE HAVE BOOKED FOR YOU A FLIGHT AND HOTEL AND 3 EXCURSIONS
```

Figure 53. Holiday Audit Output

This output was produced by the WHICH-TRANSACTION routine shown in Figure 54 on page 97.

```

*****
*   WHICH REPORT LAYOUT COPYBOOK   *
*****
01 FILLER PIC X(20) VALUE IS '*** C/B WHICHWS ***'.
01 WHICH-REPORT.
  03 WS-WH-REPORT-LINE1.
    05 WH-TASKNO          PIC 9(07) VALUE ZEROS.
    05 FILLER             PIC XX  VALUE SPACES.
    05 WH-TRANID         PIC X(04) VALUE SPACES.
    05 FILLER             PIC XX  VALUE SPACES.
    05 WH-PROCESS-TYPE   PIC X(08) VALUE SPACES.
    05 FILLER             PIC X(11) VALUE '** RUNNING:'.
    05 WH-PROGRAM        PIC X(08) VALUE SPACES.
    05 FILLER             PIC XX  VALUE SPACES.
    05 WH-DATE           PIC X(08) VALUE SPACES.
    05 FILLER             PIC XX  VALUE SPACES.
    05 WH-TIME           PIC X(08) VALUE SPACES.
    05 FILLER             PIC X(05) VALUE ' PRC:'.
    05 WH-PROCESS-NAME   PIC X(44) VALUE SPACES.
  03 WS-WH-REPORT-LINE2.
    05 FILLER             PIC X(13) VALUE SPACES.
    05 FILLER             PIC X(05) VALUE 'ACTV:'.
    05 WH-ACTIVITY        PIC X(16) VALUE SPACES.
    05 FILLER             PIC XX  VALUE SPACES.
    05 FILLER             PIC X(06) VALUE 'EVENT:'.
    05 WH-EVENT           PIC X(16) VALUE SPACES.
    05 FILLER             PIC XX  VALUE SPACES.
    05 FILLER             PIC X(04) VALUE 'SUB:'.
    05 WH-SUBEVENT       PIC X(16) VALUE SPACES.
WHICH-TRANSACTION.
  EXEC CICS ASSIGN PROGRAM(WH-PROGRAM)
                PROCESS(WH-PROCESS-NAME)
                PROCESSTYPE(WH-PROCESS-TYPE)
                ACTIVITY(WH-ACTIVITY)

  END-EXEC.
  MOVE EIBTASKN TO WH-TASKNO.
  MOVE EIBTRNID TO WH-TRANID.
  MOVE WS-EVENT TO WH-EVENT.
  EXEC CICS ASKTIME
          ABSTIME(WS-ABSTIME)

  END-EXEC.
  EXEC CICS FORMATTIME
          ABSTIME(WS-ABSTIME)
          DATESEP('-')
          DDMYY(WH-DATE)
          TIME(WH-TIME)
          TIMESEP(':')

  END-EXEC.
  SKIP1
  EXEC CICS WRITEQ TD QUEUE('BMTD')
          FROM(WS-WH-REPORT-LINE1)

  END-EXEC.
  SKIP1
  EXEC CICS WRITEQ TD QUEUE('BMTD')
          FROM(WS-WH-REPORT-LINE2)

  END-EXEC.

```

Figure 54. Working Storage and Pseudo-code for the WHICH Output

## 4.8.2 Audit Log for the HOLIDAY Application

Figure 55 on page 98 shows the audit log print for the HOLIDAY application when all the elements have been booked. The audit level was set to ACTIVITY.

Refer to 11.5, “Audit Log” on page 211 for JCLs needed to define and print the CICS BTS audit log.

1CBTS Audit Trail Utility - Parameter Validation Date : 27/02/1999 Time : 00:56:43 Page 000001  
 0Exec Parm Options: Natlang (EN) (Default)  
 Translate (mixedcase) (Default)  
 Pagesize (60)

AUDITLOG(AUDIT)  
 PTYPE(BAMSHRF)

1CBTS Audit Trail Utility - Audit Print Date : 27/02/1999 Time : 00:56:43 Page 000002  
 0Ptype(BAMSHRF ) Function(Define Process ) Process(HOLI000000003 ) System(YCQF) Auditlog(CBTS )  
 Taskno(0000147) Activity(DFHROOT ) Transid(HHOL) Program(HHOLIDAY) Userid(CTS0QFD )  
 ActivityId(DFHBSHR ..GBIBMIYA.SC02070D.U...d..DFHROOT )  
 (CCCCCD411CCCCDCEC4ECFFFCDECF400CCDDDE444444444)  
 (46822890A172924981B23020704D4EAAA0146896630000000000)

Current: Transid(H001) Program(HOLI001) Userid(CTS0QFD) Date(1999.058) Time(00:49:14.482856)  
 0Ptype(BAMSHRF ) Function(Put Container ) Process(HOLI000000003 ) System(YCQF) Auditlog(CBTS )  
 Taskno(0000147) Activity(DFHROOT ) Container(GENERAL )  
 Current: Transid(H001) Program(HOLI001) Userid(CTS0QFD) Date(1999.058) Time(00:49:15.646475)  
 0Ptype(BAMSHRF ) Function(Run Process ) Process(HOLI000000003 ) System(YCQF) Auditlog(CBTS )  
 Taskno(0000147) Activity(DFHROOT ) Asynchronous

ActivityId(DFHBSHR ..GBIBMIYA.SC02070D.U...d..DFHROOT )  
 (CCCCCD411CCCCDCEC4ECFFFCDECF400CCDDDE444444444)  
 (46822890A172924981B23020704D4EAAA0146896630000000000)

Current: Transid(H001) Program(HOLI001) Userid(CTS0QFD) Date(1999.058) Time(00:49:15.646866)  
 0Ptype(BAMSHRF ) Function(Activation ) Process(HOLI000000003 ) System(YCQF) Auditlog(CBTS )  
 Taskno(0000148) Activity(DFHROOT ) Event(DFHINITIAL )

ActivityId(DFHBSHR ..GBIBMIYA.SC02070D.U...d..DFHROOT )  
 (CCCCCD411CCCCDCEC4ECFFFCDECF400CCDDDE444444444)  
 (46822890A172924981B23020704D4EAAA0146896630000000000)

Current: Transid(HHOL) Program(HHOLIDAY) Userid(CTS0QFD) Date(1999.058) Time(00:49:15.697063)  
 0Ptype(BAMSHRF ) Function(Put Container ) Process(HOLI000000003 ) System(YCQF) Auditlog(CBTS )  
 Taskno(0000148) Activity(DFHROOT ) Container(HOLIDAY )  
 Current: Transid(HHOL) Program(HHOLIDAY) Userid(CTS0QFD) Date(1999.058) Time(00:49:15.755758)  
 0Ptype(BAMSHRF ) Function(Activation ) Process(HOLI000000003 ) System(YCQF) Auditlog(CBTS )  
 Taskno(0000149) Activity(BOOKFLIGHT ) Event(DFHINITIAL )

ActivityId(DFHBSHR ..GBIBMIYA.IYCQCTSF.U)...e..BOOKFLIGHT )  
 (CCCCCD411CCCCDCEC4CECDCECED10800CDDDCDCE444444444)  
 (46822890A172924981B98383326D4066501266263978300000000)

Current: Transid(HFLY) Program(HB00KFLY) Userid(CTS0QFD) Date(1999.058) Time(00:49:15.869995)  
 0Ptype(BAMSHRF ) Function(Completion ) Process(HOLI000000003 ) System(YCQF) Auditlog(CBTS )  
 Taskno(0000149) Activity(BOOKFLIGHT ) Compstatus(Normal )

ActivityId(DFHBSHR ..GBIBMIYA.IYCQCTSF.U)...e..BOOKFLIGHT )  
 (CCCCCD411CCCCDCEC4CECDCECED10800CDDDCDCE444444444)  
 (46822890A172924981B98383326D4066501266263978300000000)

Current: Transid(HFLY) Userid(CTS0QFD) Date(1999.058) Time(00:49:15.873682)  
 0Ptype(BAMSHRF ) Function(Activation ) Process(HOLI000000003 ) System(YCQF) Auditlog(CBTS )  
 Taskno(0000154) Activity(BOOKEXCUR3 ) Event(DFHINITIAL )

ActivityId(DFHBSHR ..GBIBMIYA.IYCQCTSF.U)...BOOKEXCUR3 )  
 (CCCCCD411CCCCDCEC4CECDCECED2A300CDDDCEDCF44444444)  
 (46822890A172924981B98383326D401BC01266257349300000000)

Current: Transid(HEXC) Program(HB00KEXC) Userid(CTS0QFD) Date(1999.058) Time(00:49:15.889573)  
 0Ptype(BAMSHRF ) Function(Activation ) Process(HOLI000000003 ) System(YCQF) Auditlog(CBTS )  
 Taskno(0000153) Activity(BOOKEXCUR2 ) Event(DFHINITIAL )

ActivityId(DFHBSHR ..GBIBMIYA.IYCQCTSF.U)...BOOKEXCUR2 )  
 (CCCCCD411CCCCDCEC4CECDCECED12E00CDDDCEDCF44444444)  
 (46822890A172924981B98383326D40FB101266257349200000000)

Current: Transid(HEXC) Program(HB00KEXC) Userid(CTS0QFD) Date(1999.058) Time(00:49:15.901483)

1CBTS Audit Trail Utility - Audit Print Date : 27/02/1999 Time : 00:56:43 Page 000003  
 0Ptype(BAMSHRF ) Function(Activation ) Process(HOLI000000003 ) System(YCQF) Auditlog(CBTS )  
 Taskno(0000150) Activity(BOOKHOTEL ) Event(DFHINITIAL )

ActivityId(DFHBSHR ..GBIBMIYA.IYCQCTSF.U)...BOOKHOTEL )  
 (CCCCCD411CCCCDCEC4CECDCECED15700CDDDCEDCD444444444)  
 (46822890A172924981B98383326D4086401266286353000000000)

Current: Transid(HHOT) Program(HB00KHOT) Userid(CTS0QFD) Date(1999.058) Time(00:49:15.918778)  
 0Ptype(BAMSHRF ) Function(Activation ) Process(HOLI000000003 ) System(YCQF) Auditlog(CBTS )  
 Taskno(0000151) Activity(BOOKCAR ) Event(DFHINITIAL )

ActivityId(DFHBSHR ..GBIBMIYA.IYCQCTSF.U)...BOOKCAR )  
 (CCCCCD411CCCCDCEC4CECDCECED1A000CDDDCD44444444444)  
 (46822890A172924981B98383326D40A220126623190000000000)

Current: Transid(HCAR) Program(HB00KCAR) Userid(CTS0QFD) Date(1999.058) Time(00:49:15.931199)  
 0Ptype(BAMSHRF ) Function(Activation ) Process(HOLI000000003 ) System(YCQF) Auditlog(CBTS )  
 Taskno(0000152) Activity(BOOKEXCUR1 ) Event(DFHINITIAL )

ActivityId(DFHBSHR ..GBIBMIYA.IYCQCTSF.U)...BOOKEXCUR1 )  
 (CCCCCD411CCCCDCEC4CECDCECED1EA00CDDDCEDCF44444444)

Figure 55 (Part 1 of 2). Holiday Audit Printout

```

(46822890A172924981B98383326D40C5A0126625734910000000)
Current: Transid(HEXC) Program(HBOOKEXC) Userid(CTSQ0FD) Date(1999.058) Time(00:49:16.404505)
0Ptype(BAMSHRF ) Function(Completion ) Process(HOLI000000003 ) System(YCQF) Auditlog(CBTS )
Taskno(0000154) Activity(BOOKEXCUR3 ) Compstatus(Normal )
ActivityId(DFHBSHR ..GBIBMIYA.IYCQTSF.U)....BOOKEXCUR3 )
(CCCCECD411CCCCDCEC4CECDCECED2A300CDDCECEDF4444444)
(46822890A172924981B98383326D401BC0126625734930000000)

Current: Transid(HEXC) Userid(CTSQ0FD) Date(1999.058) Time(00:49:16.406920)
0Ptype(BAMSHRF ) Function(Completion ) Process(HOLI000000003 ) System(YCQF) Auditlog(CBTS )
Taskno(0000153) Activity(BOOKEXCUR2 ) Compstatus(Normal )
ActivityId(DFHBSHR ..GBIBMIYA.IYCQTSF.U)....BOOKEXCUR2 )
(CCCCECD411CCCCDCEC4CECDCECED12E00CDDCECEDF4444444)
(46822890A172924981B98383326D40FB10126625734920000000)

Current: Transid(HEXC) Userid(CTSQ0FD) Date(1999.058) Time(00:49:16.471753)
0Ptype(BAMSHRF ) Function(Completion ) Process(HOLI000000003 ) System(YCQF) Auditlog(CBTS )
Taskno(0000150) Activity(BOOKHOTEL ) Compstatus(Normal )
ActivityId(DFHBSHR ..GBIBMIYA.IYCQTSF.U)....BOOKHOTEL )
(CCCCECD411CCCCDCEC4CECDCECED15700CDDCECED44444444)
(46822890A172924981B98383326D40864012662863530000000)

Current: Transid(HHOT) Userid(CTSQ0FD) Date(1999.058) Time(00:49:16.925272)
0Ptype(BAMSHRF ) Function(Completion ) Process(HOLI000000003 ) System(YCQF) Auditlog(CBTS )
Taskno(0000151) Activity(BOOKCAR ) Compstatus(Normal )
ActivityId(DFHBSHR ..GBIBMIYA.IYCQTSF.U).s...BOOKCAR )
(CCCCECD411CCCCDCEC4CECDCECED1A000CDDCCD444444444)
(46822890A172924981B98383326D40A22012662319000000000)

Current: Transid(HCAR) Userid(CTSQ0FD) Date(1999.058) Time(00:49:17.979292)
0Ptype(BAMSHRF ) Function(Completion ) Process(HOLI000000003 ) System(YCQF) Auditlog(CBTS )
Taskno(0000152) Activity(BOOKEXCUR1 ) Compstatus(Normal )
ActivityId(DFHBSHR ..GBIBMIYA.IYCQTSF.U).V...BOOKEXCUR1 )
(CCCCECD411CCCCDCEC4CECDCECED1EA00CDDCECEDF4444444)
(46822890A172924981B98383326D40C5A0126625734910000000)

Current: Transid(HEXC) Userid(CTSQ0FD) Date(1999.058) Time(00:49:18.005022)
1CBTS Audit Trail Utility - Audit Print Date : 27/02/1999 Time : 00:56:43 Page 000004
0Ptype(BAMSHRF ) Function(Activation ) Process(HOLI000000003 ) System(YCQF) Auditlog(CBTS )
Taskno(0000160) Activity(DFHROOT ) Event(EX1-BOOK )
ActivityId(DFHBSHR ..GBIBMIYA.SC02070D.U...DFHROOT )
(CCCCECD411CCCCDCEC4ECFFFCDECE400CCDDDE444444444)
(46822890A172924981B23020704D4EAAA014689663000000000)

Current: Transid(HHOL) Program(HHOLIDAY) Userid(CTSQ0FD) Date(1999.058) Time(00:49:19.033195)
0Ptype(BAMSHRF ) Function(Put Container ) Process(HOLI000000003 ) System(YCQF) Auditlog(CBTS )
Taskno(0000160) Activity(DFHROOT ) Container(RETRY )
Current: Transid(HHOL) Program(HHOLIDAY) Userid(CTSQ0FD) Date(1999.058) Time(00:49:19.040961)
0Ptype(BAMSHRF ) Function(Put Container ) Process(HOLI000000003 ) System(YCQF) Auditlog(CBTS )
Taskno(0000160) Activity(DFHROOT ) Container(HOLIDAY )
Current: Transid(HHOL) Program(HHOLIDAY) Userid(CTSQ0FD) Date(1999.058) Time(00:49:19.041141)
0Ptype(BAMSHRF ) Function(Activation ) Process(HOLI000000003 ) System(YCQF) Auditlog(CBTS )
Taskno(0000161) Activity(BOOKCAR ) Event(DFHINITIAL )
ActivityId(DFHBSHR ..GBIBMIYA.IYCQTSF.U).s...BOOKCAR )
(CCCCECD411CCCCDCEC4CECDCECED1A000CDDCCD444444444)
(46822890A172924981B98383326D40A220 126623190000000000)

Current: Transid(HCAR) Program(HBOOKCAR) Userid(CTSQ0FD) Date(1999.058) Time(00:49:19.135012)
0Ptype(BAMSHRF ) Function(Completion ) Process(HOLI000000003 ) System(YCQF) Auditlog(CBTS )
Taskno(0000161) Activity(BOOKCAR ) Compstatus(Normal )
ActivityId(DFHBSHR ..GBIBMIYA.IYCQTSF.U).s...BOOKCAR )
(CCCCECD411CCCCDCEC4CECDCECED1A000CDDCCD444444444)
(46822890A172924981B98383326D40A22012662319000000000)

Current: Transid(HCAR) Userid(CTSQ0FD) Date(1999.058) Time(00:49:19.143723)
0Ptype(BAMSHRF ) Function(Activation ) Process(HOLI000000003 ) System(YCQF) Auditlog(CBTS )
Taskno(0000162) Activity(DFHROOT ) Event(CAR-BOOK )
ActivityId(DFHBSHR ..GBIBMIYA.SC02070D.U...DFHROOT )
(CCCCECD411CCCCDCEC4ECFFFCDECE400CCDDDE444444444)
(46822890A172924981B23020704D4EAAA014689663000000000)

Current: Transid(HHOL) Program(HHOLIDAY) Userid(CTSQ0FD) Date(1999.058) Time(00:49:19.171756)
0Ptype(BAMSHRF ) Function(Put Container ) Process(HOLI000000003 ) System(YCQF) Auditlog(CBTS )
Taskno(0000162) Activity(DFHROOT ) Container(HOLIDAY )
Current: Transid(HHOL) Program(HHOLIDAY) Userid(CTSQ0FD) Date(1999.058) Time(00:49:19.184739)
0Ptype(BAMSHRF ) Function(Completion ) Process(HOLI000000003 ) System(YCQF) Auditlog(CBTS )
Taskno(0000162) Activity(DFHROOT ) Compstatus(Normal )
ActivityId(DFHBSHR ..GBIBMIYA.SC02070D.U...DFHROOT )
(CCCCECD411CCCCDCEC4ECFFFCDECE400CCDDDE444444444)
(46822890A172924981B23020704D4EAAA014689663000000000)

Current: Transid(HHOL) Userid(CTSQ0FD) Date(1999.058) Time(00:49:19.226169)
1CBTS Audit Trail Utility - Selection Results Date : 27/02/1999 Time : 00:56:43 Page 000005
0Number of Audit records read : 321
0Number of records selected : 25
0 Selection : Ptype(BAMSHRF ) Records selected : 25
0 Processing Complete

```

Figure 55 (Part 2 of 2). Holiday Audit Printout

### 4.8.3 CBAM Displays

Figure 56 through Figure 61 on page 102 show the CBAM screen produced by the activities for a HOLIDAY application. To be able to capture this output it was necessary to hold transaction HHOT with CEDX.

From the first CBAM screen, we placed the cursor on the BAMSHRF processtype line and pressed Enter. This takes us to the Process screen as seen in Figure 56.

```

CBAM                                     Processtype BAMS
  Process                               Mode    Comp    Susp    Cont
  HOLI000000003                       Dormant  Incomplete No    -

Use cursor and Enter for Activities or Containers (tab to Conts)
PF3=Return  7=Back  8=Forward
  
```

Figure 56. Holiday CBAM Process Screen Dump

Placing the cursor alongside process HOLI000000003 and pressing Enter gives us the display shown in Figure 57. Here, we can see that all the child activities have finished normally except BOOKHOTEL, which is being held by CEDX.

```

CBAM          Process HOLI000000003          Processtype BAMSHRF
  Activity                               Mode    Comp
  DFHROOT                                Dormant  Incomplete
  BOOKEXCUR3                             Complete Normal
  BOOKEXCUR2                             Complete Normal
  BOOKEXCUR1                             Complete Normal
  BOOKCAR                                 Complete Normal
  BOOKHOTEL                               Initial  Incomplete
  BOOKFLIGHT                             Complete Normal

Use cursor and Enter for details
PF3=Return  7=Back  8=Forward
  
```

Figure 57. Holiday CBAM Activity Screen Dump

Placing the cursor alongside DFHROOT and pressing Enter gives us the display in Figure 58 on page 101. This shows us the program, transid and userid associated with the activity.

```

CBAM          Process HOLI000000003          Processtype BAMS
Activity DFHROOT
Program      HHOLIDAY
Transid     HHOL
Userid      CTSQ0FD

Containers
Events
Timers

Use cursor and Enter for Containers, Events or Timers

PF3=Return  7=Back  8=Forward

```

Figure 58. Holiday CBAM Activity DFHROOT Screen Dump

Placing the cursor alongside Containers and pressing Enter gives us the display in Figure 59. We see the containers associated with the activity and the data length.

```

CBAM          Process HOLI000000003          Processtype BAMS
Container      Datalength
GENERAL       99
HOLIDAY       7

Use cursor and Enter for Containers, Events or Timers

PF3=Return  7=Back  8=Forward

```

Figure 59. Holiday CBAM Activity DFHROOT Containers Screen Dump

Pressing PF3 to go back to the activity screen, then placing the cursor alongside Events and pressing enter gives us the display in Figure 60 on page 102. We can see the events associated with the activity. We can see composite events and associated subevents, timers, and system event. We can also see that all the child activity events have fired, except HOTEL-BOOK. As the composite event BOOK-HOLIDAY takes AND as its predicate, this event cannot fire until HOTEL-BOOK fires. The timer will fire when the time expires. The system event DFHINITIAL has already fired but is shown as NOTFIRED because the field shows the current status of the event, not whether the event has ever fired in the past. When DFHINITIAL fired and was retrieved, the act of retrieving the event reset its fire status to NOTFIRED.

CBAM	Process HOLI000000003	Processt		
Activity DFHROOT				
Event	Type	Fired	Composite	Timer
BOOK-HOLIDAY	Composite	No	AND	
CAR-BOOK	Activity	Yes	BOOK-HOLIDAY	
DFHINITIAL	System	No		
EX1-BOOK	Activity	Yes	BOOK-HOLIDAY	
EX2-BOOK	Activity	Yes	BOOK-HOLIDAY	
EX3-BOOK	Activity	Yes	BOOK-HOLIDAY	
FLIGHT-BOOK	Activity	Yes	BOOK-HOLIDAY	
HOTEL-BOOK	Activity	No	BOOK-HOLIDAY	
NO-RESPONSE	Timer	No		NO-RESPONSE
PF3=Return 7=Back 8=Forward				

Figure 60. Holiday CBAM Activity DFHROOT Events Screen Dump

Pressing PF3 to go back to the activity screen, then placing the cursor alongside Timers and pressing Enter gives us the display in Figure 61. This shows us the timers associated with the activity, the event associated with the timer, the status and its expiry time. In this example, the timer was set to fire at 23:36:28 on 1 March 1999. If we had continued to hold HHOT on CEDX beyond this time, then the NO-RESPONSE timer would have fired and the root activity reattached for this event.

CBAM	Process HOLI000000003	Processtype BAMS		
Activity DFHROOT				
Timer	Status	Event	Date	Time
NO-RESPONSE	Unexpired	NO-RESPONSE	03011999	233628
PF3=Return 7=Back 8=Forward				

Figure 61. Holiday CBAM Activity DFHROOT Timers Screen Dump



---

## Chapter 5. MORTGAGE and FINANCE Application

The MORTGAGE and FINANCE application was also developed for the CICS TS 1.3 System Test of CICS BTS. The application is quite complex, and a full business and application development description would not enhance your understanding of the power of the BTS API. However, this application shows some techniques that cannot be illustrated by either the SALES or the HOLIDAY applications. We feel that it is useful to give a brief overview of the FINANCE application to illustrate these techniques. The application formed a major part of the CICS TS 1.3 System Test suite and used the same control file as the HOLIDAY application. See 4.3.3, "The Control File" on page 63 for details of how this control file is used. Additional fields from the control file are utilized in the MORTGAGE and FINANCE application, and these are described where necessary in this chapter.

The functions we want to describe using this application are communication between two processes and the SUSPEND and RESUME commands.

Throughout this chapter we use the terms *process(F)* and *process(M)* when referring to the FINANCE and MORTGAGE business applications, respectively. These are generic names, as the process name can only be defined at run-time and has to be unique for each instance of a process. The terms *root activity(F)* and *root activity(M)* are also used.

---

### 5.1 Communication between Processes

This application is made up of two processes. The processes are designed to communicate with each other. Initially, this communication takes the form of one process starting the other process and passing data to it.

Subsequently, the communication happens when one process acquires the other and passes data to it. It is essential that the process being acquired has previously defined all the INPUTEVENTs for all possible RUN ACQPROCESS INPUTEVENT commands that are going to be issued by the partner process.

In this way, the two processes can interact. In order to pass data between the two processes, it is necessary for the acquiring process to put the data in a process container of the acquired process. The data is then available to all the acquired process's activities.

#### 5.1.1 Starting Partner Process

Starting a partner process from a process is similar to using a traditional CICS transaction to start the process. The new process is defined, process containers for the defined process can be filled with data, and the process is then executed with the LINK, RUN SYNCHRONOUS, or RUN ASYNCHRONOUS commands. The new root activity, together with all of its child activities within the process, can access the data stored in the process containers written by the activity that initiated the new process. However, the initiating transaction cannot be active when the partner process wishes to access the data. This is discussed more fully in 5.5, "The Initial Transactions" on page 107.

## 5.1.2 Communication between Two Active Processes

Whenever one process needs to communicate with another, the first process has to acquire the second process, place some data in its containers, and then run the acquired process. The parameters associated with these commands are described more fully in 5.7, “Forcing a Change to the Interest Rate” on page 116.

---

## 5.2 The SUSPEND and RESUME Commands

The SUSPEND ACQPROCESS and SUSPEND ACQACTIVITY commands prevent a process or activity from being reattached when events in its event pool are fired. The only process a program can suspend is the one it has acquired in the current UOW. It is possible to execute this command using CECI SUSPEND ACQPROCESS or CECI SUSPEND ACQACTIVITY following an ACQUIRE PROCESS or ACQUIRE ACTIVITYID command, respectively.

The only activities a program can suspend are:

- Its own child activities if it is running as the activation of an activity. It can, however, suspend several of its children within the same UOW.
- The activity it has acquired, by means of an ACQUIRE ACTIVITYID command, in the current UOW.

The RESUME ACQPROCESS and RESUME ACQACTIVITY commands resume a process or activity that has previously been suspended allowing for the reattachment of events in its event pool. If events that would normally have caused reattachment have occurred during the time the process or activity was suspended, the latter is reattached for all of these events.

The only process a program can resume is the one it has acquired in the current UOW. The only activities a program can resume are:

- Its own child activities, if it is running as the activation of an activity. It can, however, resume several of its children within the same UOW.
- The activity it has acquired, by means of an ACQUIRE ACTIVITYID command, in the current UOW.

See 5.7, “Forcing a Change to the Interest Rate” on page 116 for a description of these commands in the context of the application.

---

## 5.3 The MORTGAGE and FINANCE Application

This application is made up of two CICS BTS processes, process(M) and process(F). There is a need for the two to communicate, and this is achieved by using the ACQUIRE PROCESS command from one process to acquire the other process.

The processes we use have been designed to simulate the 25 year term of a loan to purchase a house. The personal finance process initially requests the loan, which will be either approved or denied by the lender or mortgage company.

In addition, the finance business transaction may wish to book a holiday (see Chapter 4, “HOLIDAY Application” on page 59.)

At regular intervals, the finance business transaction (FINANCE) makes payments to the mortgage business transaction (MORTGAGE). If the mortgage lender does not receive these regular payments, it will issue a payment-due notice to FINANCE. If the payments are still not made, then the mortgage lender may instigate repossession proceedings. MORTGAGE will issue an annual statement outlining the payments that have been made during the previous accounting period.

At times, it may be necessary to change the interest rate applicable to the mortgage, in which case, both business transactions will need to be suspended until the interest rate change can be applied to all accounts.

FINANCE may decide to make an early settlement of the loan. However, MORTGAGE may find that not all monies have, in fact, been paid and, therefore, the loan repayments have not ended. When the loan has been fully repaid, the details have to be removed from both the finance and mortgage files.

The FINANCE business transaction is made up of the following actions:

- An initial entry transaction
- A finance business transaction root activity
- An activity to instigate the mortgage
- An activity to make a payment
- An activity to deal with the payment-due notice from the mortgage business transaction
- An activity to make an early settlement
- An activity to deal with an incorrect early settlement (not ended)
- An activity to deal with an annual statement
- An activity to deal with a repossession order
- An activity to book a holiday
- An activity to deal with a notification that the loan term has completed
- An activity to remove details

The MORTGAGE business transaction is made up of the following actions:

- An initial entry transaction
- A mortgage business transaction root activity
- An activity to agree or deny the mortgage
- An activity to receive a payment
- An activity to send a payment-due notice to the finance business transaction
- An activity to accept an early settlement and to inform the finance business process if it disagrees that the loan has been paid off
- An activity to issue an annual statement
- An activity to issue a repossession order
- An activity to issue a notification that the loan term has completed (notify end)
- An activity to remove details

## 5.4 Transactions, Programs, Containers, and Control File

In this section, we outline the transactions and programs used in the application, show the layout of some of the containers, and describe some of the fields in the control file that are relevant to this chapter.

### 5.4.1 Transactions and Programs

Table 5 shows the transaction IDs and programs used in the FINANCE business transaction.

<i>Table 5. The Finance Application Programs Discussed in This Book</i>			
<b>Activity</b>	<b>Transid</b>	<b>Program</b>	<b>Comments</b>
-	P001	CUST001	A traditional CICS transaction to invoke the CICS BTS process
DFHROOT	PERS	FINANCE	BTS root activity that manages the child activities that comprise the FINANCE business transaction
MORTREQ	KNEW	CUSTNEW	New mortgage request activity

Table 6 shows the transaction ids and programs used in the mortgage business transaction.

<i>Table 6. The Mortgage Application Programs Discussed in This Book</i>			
<b>Activity</b>	<b>Transid</b>	<b>Program</b>	<b>Comments</b>
DFHROOT	MORT	MORTGAGE	BTS root activity, manages the child activities that comprise the MORTGAGE business transaction
NEWMORT	MNEW	MORTNEW	New mortgage request activity

Table 7 shows the transaction IDs and programs used in both MORTGAGE and FINANCE business transactions.

<i>Table 7. The Programs Applicable to Both Applications</i>			
<b>Activity</b>	<b>Transid</b>	<b>Program</b>	<b>Comments</b>
ACQUIREP	ACQP	ACQUIREP	Used by both processes to acquire and run the other process

### 5.4.2 Container Contents

Figure 62 on page 107 shows the working storage definitions for the contents of the ACQUIREP container. It should be noted that not all the fields are used in the application.

```

*****
* ACQUIREP container LAYOUT COPYBOOK *
*****
01 FILLER PIC X(20) VALUE IS '*** C/B CACQUIRE ***'.
01 AQ-ACQUIREP.
02 AQ-REQUEST.
03 AQ-PROCESS-TYPE PIC X(08) VALUE SPACES.
03 AQ-PROCESS-NAME PIC X(36) VALUE SPACES.
03 AQ-PROCESS-INPUT-EVENT PIC X(16) VALUE SPACES.
02 AQ-CALLED-FROM.
03 AQ-CALLING-ACTIVITY PIC X(16) VALUE SPACES.
03 AQ-CALLING-PROGRAM PIC X(08) VALUE SPACES.
03 AQ-CALLING-PROCESS-TYPE PIC X(08) VALUE SPACES.
03 AQ-CALLING-PROCESS-NAME PIC X(36) VALUE SPACES.
03 FILLER PIC X(10) VALUE SPACES.
02 AQ-DATA-CONTAINER.
03 AQ-CONTAINER-NAME PIC X(16) VALUE SPACES.

```

Figure 62. Contents of Containers

### 5.5 The Initial Transactions

Figure 63 shows the flow of the programs used to start up the mortgage and finance processes.

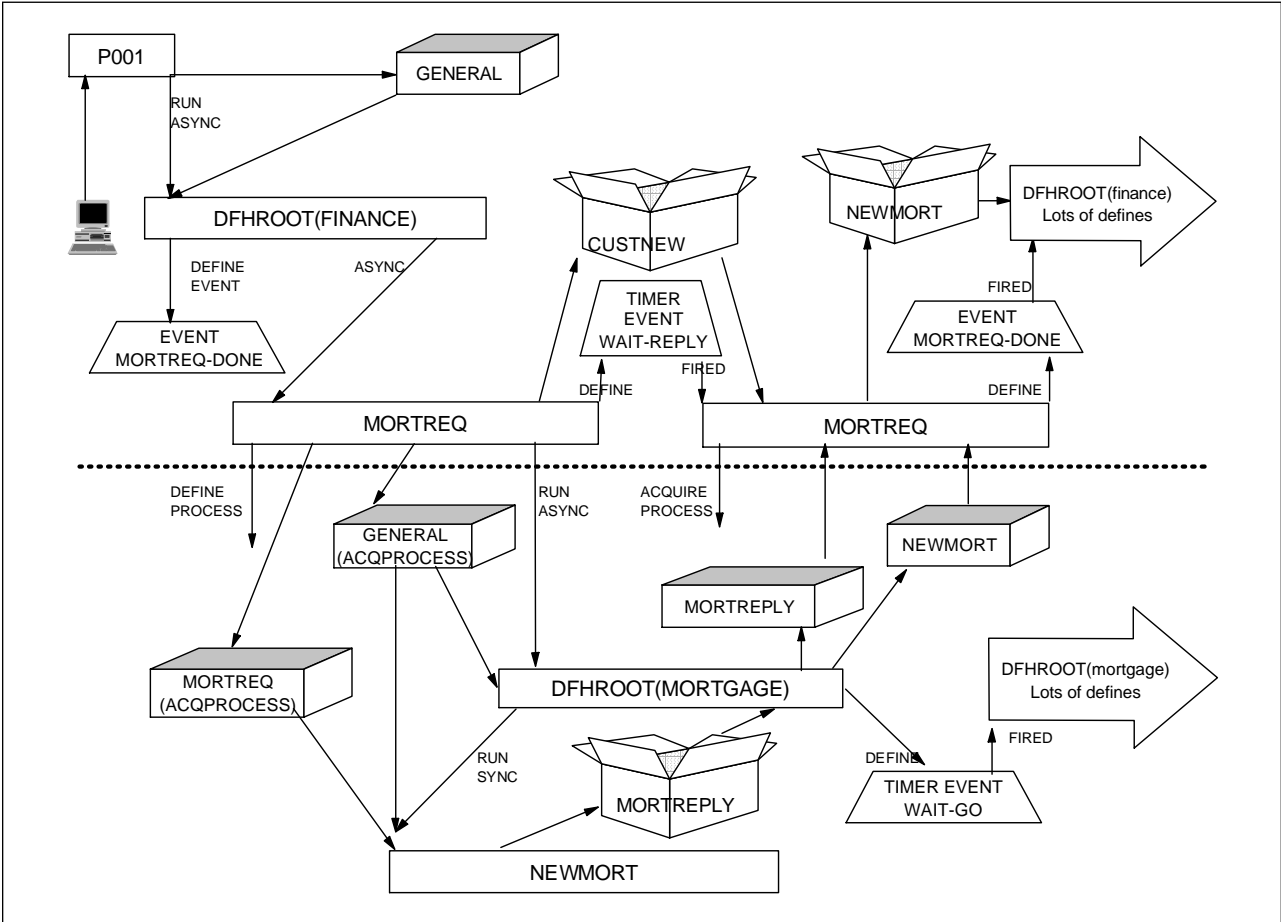


Figure 63. Flow of Programs for the Start of the Mortgage and Finance Application

In this business application, the process(F) is started from a traditional CICS transaction (transid P001), which, having read the control file, defines the process, puts data into the process(F) container GENERAL, and runs the process(F) asynchronously. The root activity(F) then defines and runs asynchronously an activity MORTREQ. Root activity(F) will be reattached when this activity completes.

MORTREQ activity is used to define a partner process(M). Data concerning the amount of the requested loan and information about the customer is passed in the process(M) container MORTREQ. General information about the running of the process is passed using the process(M) container GENERAL. Finally MORTREQ activity issues the RUN ACQPROCESS ASYNCHRONOUS command. MORTREQ needs to know whether the mortgage can be agreed. It defines a timer WAIT-REPLY that will allow it to be reinvoked when process(M) has had sufficient time to either approve or deny the loan request and put its response.

The new process(M)'s root activity(M) defines and runs synchronously an activity NEWMORT. This activity has the business logic to approve or deny the mortgage. When it has done either, it puts the reply in its activity container MORTREPLY and returns. Root activity(M) then has access to this child activity container and can put the information into its two process containers MORTREPLY and NEWMORT.

Root activity(M) defines a timer WAIT-GO in order for it to be reattached once process(F) has got the required data from its containers. Activity MORTREQ in process(F) is unable to gain access to process(M)'s containers while root activity(M) is running. Root activity(M) must issue a RETURN command and it becomes dormant to allow access and then waits until the WAIT-GO timer fires.

After the WAIT-GO timer has fired, root activity(M) is able to define all its input events, activities, and timers.

MORTREQ is reattached by the firing of the WAIT-REPLY timer. As process(M) is dormant, MORTREQ is able to GET the MORTREPLY and NEWMORT process(M) containers and process the data contained therein. The information regarding the granting or denial of the mortgage is passed back to root activity(F) in the MORTREQ activity container NEWMORT. MORTREQ can then issue a RETURN ENDACTIVITY command as it has finished its work.

Root activity(F) is reattached by the firing of the MORTREQ completion event. If the mortgage has been approved, it can then define several activities (with their associated completion events), composite and input events and timers.

The two root activities control the flow of each of the business transactions.

### 5.5.1 BTS Resources Used in the FINANCE Root Activity

The FINANCE root activity uses the following resources:

- Container RETRY                    I/O
- Container GENERAL                0
- Container MORTREQ                I            (MORTREQ)
- Container PAYREC                 I            (PAYREC)
- Container EARLYSETT               I            (EARLYSETT)
- Container PAYDUE                 I            (PAYDUE)

- File MORTCONT I
- Event DFHINITIAL (system)
- Event MORTREQ-DONE (completion)
- Event MAKE-PAYMENT (timer)
- Event MAKEPAY-DONE (completion)
- Event PAYMENT-DUE (input)
- Event PAYMENT-DUE-DONE (completion)
- Event ANNUAL-DONE (completion)
- Event EARLY-SETTLEMENT (timer)
- Event EARLY-SETT-DONE (completion)
- Event NOTIFY-END (input)
- Event NOTIFY-END-DONE (completion)
- Event NOTENDED (input)
- Event RESETPAY-DONE (completion)
- Event REMDET-DONE (completion)
- Event INTEREST-CHANGED (input)
- Event INTCHANGE-DONE (completion)
- Event ANNUAL-STATEMENT (input)
- Event REPOSSESS (input)
- Event REPOSSESS-DONE (completion)
- Event HOLIDAY (timer)
- Event HOLIDAY-DONE (completion)
- Activity MORTREQ
- Activity MAKEPAY
- Activity INTCHANGE
- Activity PAYDUE
- Activity ANNSTAT
- Activity SETTEARLY
- Activity REPOSSESS
- Activity NOTIFY-END
- Activity REMDET
- Activity NOTENDED
- Activity HOLIDAY

## 5.5.2 BTS Resources Used in the MORTGAGE Root Activity

The MORTGAGE root activity uses the following resources:

- Container RETRY I/O
- Container AUDIT I/O
- Container GENERAL I
- Container NEWMORT I (NEWMORT)

- Container PAYREC I (PAYREC)
- Container EARLYSETT I (EARLYSETT)
- Container PAYDUE I (PAYDUE)
- File MORTCONT I
- File MORTCONT I
- Event DFHINITIAL (system)
- Event TIMERS (composite)
- Event PAYMENT-RECEIVED (input)
- Event PAYMENT-REC-DONE (completion)
- Event INTEREST-DONE (completion)
- Event PAYMENT-DUE-DONE (completion)
- Event ANNUAL-DONE (completion)
- Event EARLY-SETTLEMENT (input)
- Event EARLY-SETT-DONE (completion)
- Event NOTIFY-END-DONE (completion)
- Event NOTENDED-DONE (completion)
- Event REMOVE-DET-DONE (completion)
- Event INTEREST-CHANGED (timer)
- Event PAYMENT-OVERDUE (timer)
- Event ANNUAL-STATEMENT (timer)
- Activity NEWMORT
- Activity PAYREC
- Activity INTEREST-CHANGED
- Activity PAYMENT-OVERDUE
- Activity ANNUAL-STATEMENT
- Activity EARLY-SETTLEMENT
- Activity NOTIFY-END
- Activity REMDET
- Activity NOTENDED

---

## 5.6 A Communicator Program

In this application, the two processes need to communicate with each other at regular intervals. The interface from one to the other uses a consistent mechanism. The activity ACQUIREP acquires and runs the partner process, passing messages into the process container of the acquired process.

For example, once the mortgage has been agreed upon, the process(F) has to make regular repayments to process(M). The MORTGAGE process sets up an input event (PAYMENT-RECEIVED) that allows it to be reattached when the FINANCE process wishes to make a payment. The process(F) sets up a timer to fire after, say, 1 month. Upon the firing of this timer, process(F) defines a child activity, MAKEPAY, which in turn needs to acquire and run the process(M) with the input event PAYMENT-RECEIVED.



Process(M) defines and runs activity PAYREC. This activity processes the data in the process message container, which, in this case, has information about the payment being made. See Figure 64 on page 111 for the flow of the activities when making a payment.

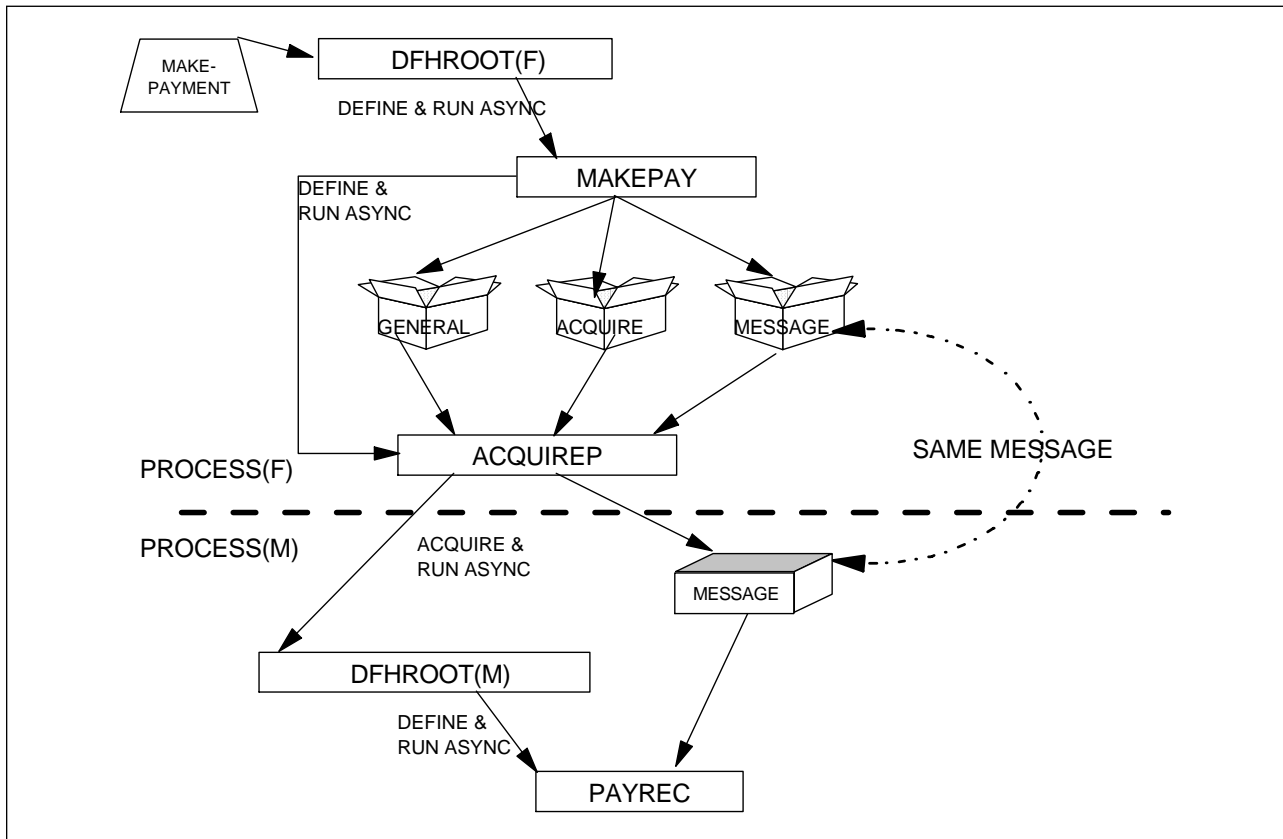


Figure 64. Flow of Activities for Making a Payment

We decided to use one program that could implement the ACQUIRE and RUN commands at all times. The program was used for the activity ACQUIREP.

This program contains the ACQUIRE PROCESS and RUN ACQPROCESS commands. This program was used within both processes and, depending on the data in the containers, the correct process (passing the correct input event) could be acquired.

Figure 65 on page 112 shows the pseudo-code for the program running under the MAKEPAY activity.

```

*****
AA-MAIN SECTION.
*
* find out which activity we are
*
EXEC CICS ASSIGN ACTIVITY(WS-ACTIVITY)
                RESP(WS-RESP) RESP2(WS-RESP2)
END-EXEC.
*
* RETRIEVE REATTACH EVENT to discover why we have been invoked
*
EXEC CICS RETRIEVE REATTACH EVENT(WS-EVENT)
                RESP(WS-RESP) RESP2(WS-RESP2)
END-EXEC
*
* output which report - TASK/PROGRAM/PROCESS ETC
*
PERFORM WHICH-TRANSACTION.
*
* GET the GENERAL CONTAINER with the process name
* and the control file details
*
EXEC CICS GET CONTAINER('GENERAL')
                INTO(C-GENERAL)
                PROCESS
                RESP(WS-RESP) RESP2(WS-RESP2)
END-EXEC
*
* If invoked with ACQUIREP, CHECK ACTIVITY and ENDACTIVITY
* used when reattachment event is not DFHINITIAL
*
IF WS-EVENT = WS-ACQUIREP
    PERFORM ACQUIREP-CHECK
    GO TO AA-RETURN
END-IF.
*
* If DFHINITIL GET GENERAL CONTAINER
*
* READ the appropriate control file record
*
EXEC CICS READ FILE(C-CONTROL-FILE-NAME)
                INTO(MC-MORTGAGE-CONTROL-RECORD)
                RIDFLD(C-CONTROL-FILE-KEY)
                RESP(WS-RESP) RESP2(WS-RESP2)
END-EXEC
*
* READ the appropriate personal details file record
*
EXEC CICS READ FILE(WS-FILE-NAME)
                INTO(PD-FILE-RECORD)
                RIDFLD(C-LOCAL-PROCESS-NAME(5:9))
                UPDATE
                RESP(WS-RESP) RESP2(WS-RESP2)
END-EXEC
* lots of business logic in here
* including updating the personal details file
*
* DEFINE and RUN ACQUIREP to acquire and run process mort
* with input event of PAYMENT-RECEIVED
*
PERFORM DEFINE-RUN-ACQUIREP.
*
* RETURN without endactivity
*
EXEC CICS RETURN
END-EXEC.

```

Figure 65 (Part 1 of 2). Pseudo-Code for the MAKEPAY Activity

```

DEFINE-RUN-ACQUIREP.
*
* DEFINE and RUN ACQUIREP to acquire and run process mort
* with input event of PAYMENT-RECEIVED
*
EXEC CICS DEFINE ACTIVITY(WS-ACQUIREP)
                TRANSID('ACQP')
                PROGRAM(WS-ACQUIREP)
                RESP(WS-RESP) RESP2(WS-RESP2)
END-EXEC.
*
EXEC CICS RUN ACTIVITY(WS-ACQUIREP)
            ASYNCHRONOUS
            RESP(WS-RESP) RESP2(WS-RESP2)
END-EXEC.
*
*
* get information about 'myself' for ACQUIREP CONTAINER
*
EXEC CICS ASSIGN PROGRAM(AQ-CALLING-PROGRAM)
                ACTIVITY(AQ-CALLING-ACTIVITY)
                PROCESS(AQ-CALLING-PROCESS-NAME)
                PROCESSTYPE(AQ-CALLING-PROCESS-TYPE)
                RESP(WS-RESP) RESP2(WS-RESP2)
END-EXEC.
MOVE MC-MORTGAGE-PROCESSTYPE TO AQ-PROCESS-TYPE.
MOVE WS-MORT TO AQ-PROCESS-GROUP.
MOVE C-MORTGAGE-REF-KEY TO AQ-PROCESS-MORTNO-KEY.
MOVE 'PAYMENT-RECEIVED' TO AQ-PROCESS-INPUT-EVENT.

MOVE 'MAKE.PAYMENT' TO AQ-CONTAINER-NAME
MOVE 'MAKE-PAYMENT-MESSAGE' TO AQ-CONTAINER-NAME
*
EXEC CICS PUT CONTAINER(WS-ACQUIREP)
            FROM(AQ-ACQUIREP)
            ACTIVITY(WS-ACQUIREP)
            RESP(WS-RESP) RESP2(WS-RESP2)
END-EXEC.
*
EXEC CICS PUT CONTAINER('MAKE.PAYMENT')
            FROM(MAKE-PAYMENT-MESSAGE)
            ACTIVITY(WS-ACQUIREP)
            RESP(WS-RESP) RESP2(WS-RESP2)
END-EXEC.
*
*****
ACQUIREP-CHECK.
*
* CHECK ACTIVITY ACQUIREP - If not of, abend
*
EXEC CICS CHECK ACTIVITY(WS-ACQUIREP)
                COMPSTATUS(WS-COMP-STATUS)
                RESP(WS-RESP) RESP2(WS-RESP2)
END-EXEC.
*****

```

Figure 65 (Part 2 of 2). Pseudo-Code for the MAKEPAY Activity

The two containers highlighted in the pseudo-code have some important data.

- CONTAINER(WS-ACQUIREP) has information telling activity ACQUIREP which process it will need to acquire. The contents of the container can be seen in Figure 62 on page 107.

The AQ-REQUEST fields show the processtype, process, and the reference key of the details files. Additionally, they store the input event to add to the RUN ACQPROCESS command.

The AQ-CALLED-FROM fields have information about the activity that initiated this instance of ACQUIREP activity and is mainly used for the logging to the TD queue BMTD.

The AQ-DATA-CONTAINER field holds the name of the message container.

- CONTAINER('MAKE.PAYMENT') is the container that holds the message to be processed by the partner process (for example, the amount of payment to be made).

This container will be PUT as an ACQPROCESS container in activity ACQUIREP.

Figure 66 on page 115 shows the pseudo-code for the program running under the ACQUIREP activity. This program is a generic program and can be used by either process(M) to acquire process(F) or vice versa.

```

AA-MAIN SECTION.
* GET ACQUIREP CONTAINER
  EXEC CICS GET CONTAINER('ACQUIREP')
           INTO(AQ-ACQUIREP)
           RESP(W5-RESP) RESP2(W5-RESP2)
  END-EXEC.
*
* SET PROCESS NAME FOR ERROR REPORTS UNTIL GENERAL CONT. READ
*
  MOVE AQ-CALLING-PROCESS-NAME TO C-LOCAL-PROCESS-NAME.
  EXEC CICS GET CONTAINER(AQ-CONTAINER-NAME)
           INTO(AQ-CONTAINER-MESSAGE)
           RESP(W5-RESP) RESP2(W5-RESP2)
  END-EXEC.
*
* GET INFORMATION ABOUT 'MYSELF'
*
  EXEC CICS ASSIGN ACTIVITY(W5-ACTIVITY)
           PROCESS(W5-PROCESS-NAME)
           PROCESSTYPE(W5-PROCESS-TYPE)
           RESP(W5-RESP) RESP2(W5-RESP2)
  END-EXEC.
*
* RETRIEVE REATTACH EVENT TO DISCOVER WHY WE'VE BEEN INVOKED
*
  EXEC CICS RETRIEVE REATTACH EVENT(W5-EVENT)
           RESP(W5-RESP)
           RESP2(W5-RESP2)
  END-EXEC.
*
* GET GENERAL CONTAINER FOR CURRENT ACTIVITY
*
  EXEC CICS GET CONTAINER('GENERAL')
           PROCESS
           INTO(C-GENERAL)
           RESP(W5-RESP) RESP2(W5-RESP2)
  END-EXEC.
  EXEC CICS GET CONTAINER(AQ-CONTAINER-NAME)
           INTO(AQ-CONTAINER-MESSAGE)
           RESP(W5-RESP) RESP2(W5-RESP2)
  END-EXEC.
  MOVE C-LOCAL-PROCESS-NAME TO W5-MAVED-C-PROCESS-NAME.
  MOVE C-PROCESS-TYPE      TO W5-MAVED-C-PROCESS-TYPE.
  MOVE W5-PROCESS-NAME    TO C-LOCAL-PROCESS-NAME.
  MOVE W5-PROCESS-TYPE    TO C-PROCESS-TYPE.
*
* ACQUIRE THE REQUIRED PROCESS
*
  EXEC CICS ACQUIRE PROCESS(AQ-PROCESS-NAME)
           PROCESSTYPE(AQ-PROCESS-TYPE)
           RESP(W5-RESP) RESP2(W5-RESP2)
  END-EXEC.
  EXEC CICS PUT CONTAINER(AQ-CONTAINER-NAME)
           FROM(AQ-CONTAINER-MESSAGE)
           ACQPROCESS
           RESP(W5-RESP) RESP2(W5-RESP2)
  END-EXEC.
*
* RUN THE ACQUIRED PROCESS
*
  EXEC CICS RUN ACQPROCESS
           ASYNCHRONOUS
           INPUTEVENT(AQ-PROCESS-INPUT-EVENT)
           RESP(W5-RESP) RESP2(W5-RESP2)
  END-EXEC.
  EXEC CICS RETURN
           ENDACTIVITY
  END-EXEC.

```

Figure 66. Pseudo-Code for the ACQUIREP Activity

The generic activity gets its own container ACQUIREP. This container gives it all the information it needs about who activated it and which process it will need to acquire. It also knows the name of the container that will hold the message that the other process has to have.

The ASSIGN PROCESS command is used to find out about itself so that the TD queue tracking can continue in BMTD.

The activity then gets the message activity container, acquires the other process, puts the message in a ACQPROCESS container and issues the RUN ACQPROCESS command.

The message is now available for any activity, within the scope of the acquired activity, to get.

---

## 5.7 Forcing a Change to the Interest Rate

Process(M) has the ability to change the interest rate following the firing of a timer. However, sometimes it may be necessary to force this to happen in advance of the due date.

In the application, this is achieved by suspending the processes, changing data on the control file, resuming the processes, and forcing the timer to fire.

The logic requires that the new interest rate be applied to all records in the control file. The interest change activity for each process is then run. As there is already a timer in place to reactivate process(M), the timer is forced so that this reactivation takes place immediately.

The following program flow provides the logic to achieve the stated objective.

```
Find the processtype
  Find the process
    Get the activityid
    Browse the events until you find the INTEREST-CHANGE event
    Inquire on the timer to see if it has already fired
    Write this process to the TS QUEUE(SUSPEND)
  Loop
Loop
Read TSQ(Suspend)
  Acquire the process
  Suspend process
  Syncpoint
Loop
Update all the records in the control file with the new interest rate
Read TSQ(Suspend)
  Acquire the process
  Resume process
  Force Interest-Change Timer
  Syncpoint
Loop
```

It is important to note that this program must issue an explicit syncpoint between each suspend or each resume. The SUSPEND and RESUME commands can only be issued against an acquired process, and only one process can be acquired in one UOW.

Figure 67 on page 117 shows the pseudo-code for the program to suspend the processes and change the interest rate.

```

01 WS-PROCESS-ID.
03 WS-ACTIVITYID          PIC X(52).          RVI00980
03 WS-PROCESSTYPE        PIC X(8).           RVI00980
03 WS-PROCESS            PIC X(36).          RVI00980
03 WS-SUSPEND-FLAG      PIC X VALUE SPACE.
    88 SUSPENDED          VALUE 'Y'.
03 EXP-TIME              PIC X(8).
03 EXP-DATE              PIC X(8).
AA-MAIN SECTION.          RVI01050
    EXEC CICS DELETEQ TS QUEUE('SUSPEND')
        RESP(WO-RESP)
    END-EXEC

*
* Start browsing processtypes using INQUIRE PROCESSTYPE START
*
    EXEC CICS INQUIRE PROCESSTYPE
        START
        RESP(WO-RESP)
        RESP2(WO-RESP2)
    END-EXEC

*
* Get the next process instance defined with this processtype
*
    EXEC CICS INQUIRE PROCESSTYPE(WO-PROCESSTYPE)
        NEXT
        AUDITLEVEL(WO-AUDITLEVEL)
        RESP(WO-RESP)
        RESP2(WO-RESP2)
    END-EXEC

*
* Get all the processes for this processtype
*
    PERFORM UNTIL WO-RESP = DFHRESP(END)
*
* Start browsing processes for this processtype
*
        EXEC CICS STARTBROWSE PROCESS
            PROCESSTYPE(WO-PROCESSTYPE)
            BROWSETOKEN(WO-BROWSETOKEN)
            RESP(WO-RESP)
            RESP2(WO-RESP2)
        END-EXEC

*
* Get the first process instance for this processtype
*
        EXEC CICS GETNEXT PROCESS(WO-PROCESS)
            BROWSETOKEN(WO-BROWSETOKEN)
            ACTIVITYID(WO-ACTIVITY)
            RESP(WO-RESP)
            RESP2(WO-RESP2)
        END-EXEC

*
* And loop through until we have got them all
*
        PERFORM UNTIL WO-RESP = DFHRESP(END)
*

```

Figure 67 (Part 1 of 8). Pseudo-Code for the Program to Suspend and Resume a Process

```

*
*   Start browsing the events for this activityid
*
      EXEC CICS STARTBROWSE EVENT
          BROWSETOKEN(WS-EVENT-BROWSETOKEN)
          ACTIVITYID(WS-ACTIVITYID)
          RESP(WS-RESP)
          RESP2(WS-RESP2)
      END-EXEC
*
*   Get the first event for this activityid
*
*
*   And loop through them until we find INTEREST-CHANGED
*
      PERFORM UNTIL WS-EVENT = INTEREST-CHANGED
          OR WS-RESP = DFHRESP(END)
          EXEC CICS GETNEXT EVENT(WS-EVENT)
              BROWSETOKEN(WS-EVENT-BROWSETOKEN)
              FIRESTATUS(WS-FIRESTATUS)
              TIMER(WS-TIMER)
              RESP(WS-RESP)
              RESP2(WS-RESP2)
          END-EXEC
      END-PERFORM
*
      IF WS-EVENT = INTEREST-CHANGED
          AND WS-TIMER NOT = SPACES
          AND WS-TIMER NOT = LOW-VALUES
          EXEC CICS INQUIRE TIMER(WS-TIMER)
              ACTIVITYID(WS-ACTIVITYID)
              STATUS(WS-TIMERSTATUS)
              ABSTIME(WS-ABSTIME)
              RESP(WS-RESP)
              RESP2(WS-RESP2)
          END-EXEC
          EXEC CICS FORMATTIME
              ABSTIME(WS-ABSTIME)
              DATESEP('-')
              DDMMYY(EXP-DATE)
              TIME(EXP-TIME)
              TIMESEP(':')
          END-EXEC
          EXEC CICS WRITEQ TS QUEUE('SUSPEND')
              FROM(WS-PROCESS-ID)
              RESP(WS-RESP)
              RESP2(WS-RESP2)
          END-EXEC
      END-IF

```

Figure 67 (Part 2 of 8). Pseudo-Code for the Program to Suspend and Resume a Process



```

*
* And get the next process
*
      EXEC CICS GETNEXT PROCESS(WS-PROCESS)
              BROWSETOKEN(WS-BROWSETOKEN)
              RESP(WS-RESP)
              RESP2(WS-RESP2)
      END-EXEC
      END-PERFORM
*
* End this browse
*
      EXEC CICS ENDBROWSE PROCESS
              BROWSETOKEN(WS-BROWSETOKEN)
              RESP(WS-RESP)
              RESP2(WS-RESP2)
      END-EXEC
*
* Get the next processtype
*
      EXEC CICS INQUIRE PROCESSTYPE(WS-PROCESSTYPE)
              NEXT
              AUDITLEVEL(WS-AUDITLEVEL)
              RESP(WS-RESP)
              RESP2(WS-RESP2)
      END-EXEC
      END-PERFORM
*
* Get the first item from our list of processes
*
      MOVE 1 TO WS-ITEM
      EXEC CICS READQ TS QUEUE('SUSPEND')
              INTO(WS-PROCESS-ID)
              ITEM(WS-ITEM)
              RESP(WS-RESP)
              RESP2(WS-RESP2)
      END-EXEC

```

Figure 67 (Part 3 of 8). Pseudo-Code for the Program to Suspend and Resume a Process

```

*
* If we have any processes to suspend, we need to suspend them
*
  IF WS-RESP = DFHRESP(NORMAL)
    PERFORM UNTIL WS-RESP = DFHRESP(ITEMERR)
*
* If the firesstatus is NOTFIRED we are going to suspend
* the process
*
    IF WS-FIRESTATUS = DFHVALUE(NOTFIRED)
      IF ACQ-ACT
        PERFORM DO-SUSPEND-ACTIVITY
      ELSE
*
* First we have to acquire it
*
        EXEC CICS ACQUIRE PROCESS(WS-PROCESS)
          PROCESSTYPE(WS-PROCESSTYPE)
          RESP(WS-RESP)
          RESP2(WS-RESP2)
        END-EXEC
*
*
* Then we can suspend it
*
        EXEC CICS SUSPEND ACQPROCESS
          RESP(WS-RESP)
          RESP2(WS-RESP2)
        END-EXEC
        IF WS-RESP NOT = DFHRESP(NORMAL)
          AND WS-RESP2 NOT = 14
          MOVE SUSPEND-PROC-FAIL TO CER-CICS-CALL
          MOVE WS-PROCESS TO CER-CICS-RESOURCE
          PERFORM CICS-ERROR
        END-IF

```

Figure 67 (Part 4 of 8). Pseudo-Code for the Program to Suspend and Resume a Process

```

*
* Make a note of the fact that we have suspended it
*
      IF WS-RESP2 NOT = 14
      MOVE 'Y' TO WS-SUSPEND-FLAG
      EXEC CICS WRITEQ TS QUEUE('SUSPEND')
              REWRITE
              ITEM(WS-ITEM)
              FROM(WS-PROCESS-ID)
              RESP(WS-RESP)
              RESP2(WS-RESP2)
      END-EXEC
      MOVE WS-PROCESS TO WS-SUSPEND-PROCESS
      EXEC CICS WRITEQ TD QUEUE('BMTD')
              FROM(WS-PROCESS-SUSPENDED)
              RESP(WS-RESP)
              RESP2(WS-RESP2)
      END-EXEC
    ELSE
      MOVE WS-PROCESS TO WS-NOT-SUSPEND-PROCESS
      EXEC CICS WRITEQ TD QUEUE('BMTD')
              FROM(WS-PROCESS-NOT-SUSPENDED)
              RESP(WS-RESP)
              RESP2(WS-RESP2)
      END-EXEC
    END-IF
EXEC CICS CHECK ACQPROCESS
      COMPSTATUS(WS-COMPSTATUS)
      SUSPSTATUS(WS-SUSPSTATUS)
      RESP(WS-RESP)
      RESP2(WS-RESP2)
END-EXEC
    IF (WS-SUSPSTATUS NOT = DFHVALUE(SUSPENDED)
        AND WS-SUSPEND-FLAG = 'Y')
        OR (WS-SUSPSTATUS NOT = DFHVALUE(NOTSUSPEND)
            AND WS-SUSPEND-FLAG = 'N')
      MOVE SUSPEND-STATUS-INCORRECT TO CER-CICS-CALL
      PERFORM CICS-ERROR
    END-IF
  END-IF (from the if acq-act)
*
* Need to syncpoint because we can only have one process
* acquired at a time
*
      EXEC CICS SYNCPOINT
      END-EXEC
    END-IF

```

Figure 67 (Part 5 of 8). Pseudo-Code for the Program to Suspend and Resume a Process

```

*
* Get the next process from our list of suspended processes
*
      ADD 1 TO WS-ITEM
      EXEC CICS READQ TS QUEUE('SUSPEND')
              INTO(WS-PROCESS-ID)
              ITEM(WS-ITEM)
              RESP(WS-RESP)
              RESP2(WS-RESP2)
      END-EXEC
    END-PERFORM
  END-IF
  MOVE 000000001 TO MC-RECORD-REF-KEY
*
* Start browsing the control file at the front
*
      EXEC CICS STARTBR FILE(WS-CONTROL-FILE)
              RIDFLD(MC-RECORD-REF-KEY)
              RESP(WS-RESP)
              RESP2(WS-RESP)
      END-EXEC
*
* Note that the browse with update
* requires token to be specified. Also
* the file must be defined as RLS capable
*
      EXEC CICS READNEXT FILE(WS-CONTROL-FILE)
              RIDFLD(MC-RECORD-REF-KEY)
              INTO(MC-MORTGAGE-CONTROL-RECORD)
              UPDATE
              TOKEN(WS-TOKEN)
              RESP(WS-RESP)
              RESP2(WS-RESP)
      END-EXEC
*
* Update the interest change for all records.
* This perform is simply business logic and therefore
* not expanded
*

```

Figure 67 (Part 6 of 8). Pseudo-Code for the Program to Suspend and Resume a Process

```

PERFORM UPDATE-INTEREST UNTIL MC-RECORD-REF-KEY
                        = 99999999
*
* Get the first item from our list of suspended processes
*
MOVE 1 TO WS-ITEM
EXEC CICS READQ TS QUEUE('SUSPEND')
      INTO(WS-PROCESS-ID)
      ITEM(WS-ITEM)
      RESP(WS-RESP)
      RESP2(WS-RESP2)
END-EXEC
*
* If we have any suspended processes, we need to resume them
*
IF WS-RESP = DFHRESP(NORMAL)
  PERFORM UNTIL WS-RESP = DFHRESP(ITEMERR)
*****
* When we have updated all the records, we can
* resume all the processes
*****
*
* First we have to acquire it
*
IF SUSPENDED
  IF ACQ-ACT
    PERFORM DO-RESUME-ACTIVITY
  ELSE
    EXEC CICS ACQUIRE PROCESS(WS-PROCESS)
          PROCESSTYPE(WS-PROCESSTYPE)
          RESP(WS-RESP)
          RESP2(WS-RESP2)
    END-EXEC
    IF WS-RESP NOT = DFHRESP(NORMAL)
      AND WS-RESP2 NOT = 14
      MOVE ACQUIRE-PROCESS-FAIL TO CER-CICS-CALL
      MOVE WS-PROCESS TO CER-CICS-RESOURCE
      PERFORM CICS-ERROR
    END-IF
*
* Resume the process
*
EXEC CICS RESUME ACQPROCESS
          RESP(WS-RESP)
          RESP2(WS-RESP2)
END-EXEC
MOVE WS-PROCESS TO WS-RESUME-PROCESS
EXEC CICS WRITEQ TD QUEUE('BMTD')
      FROM(WS-PROCESS-RESUMED)
      RESP(WS-RESP)
      RESP2(WS-RESP2)
END-EXEC
EXEC CICS CHECK ACQPROCESS
          COMPSTATUS(WS-COMPSTATUS)
          SUSPSTATUS(WS-SUSPSTATUS)
          RESP(WS-RESP)
          RESP2(WS-RESP2)
END-EXEC
IF WS-SUSPSTATUS NOT = DFHVALUE(NOTSUSPENDED)
  MOVE RESUME-STATUS-INCORRECT TO CER-CICS-CALL
  PERFORM CICS-ERROR
END-IF

```

Figure 67 (Part 7 of 8). Pseudo-Code for the Program to Suspend and Resume a Process

```

EXEC CICS FORCE TIMER(INTEREST-CHANGED)
      ACQPROCESS
      RESP(WS-RESP)
      RESP2(WS-RESP2)
END-EXEC
MOVE WS-PROCESS TO WS-FORCE-PROCESS
EXEC CICS WRITEQ TD QUEUE('BMTD')
      FROM(WS-TIMER-FORCED)
      RESP(WS-RESP)
      RESP2(WS-RESP2)
END-EXEC
END-IF

*
* Need to SYNCPOINT because we can only have one process
* acquired at a time
*
EXEC CICS SYNCPOINT
END-EXEC
END-IF

*
* Get the next process from our list of suspended processes
*
EXEC CICS READQ TS QUEUE('SUSPEND')
      INTO(WS-PROCESS-ID)
      NEXT
      RESP(WS-RESP)
      RESP2(WS-RESP2)
END-EXEC
END-PERFORM
END-IF
EXEC CICS DELETEQ TS QUEUE('SUSPEND')
END-EXEC
EXEC CICS RETURN
END-EXEC

```

Figure 67 (Part 8 of 8). Pseudo-Code for the Program to Suspend and Resume a Process

- EXEC CICS INQUIRE PROCESSTYPE(PROCESSTYPE-NAME)  
The INQUIRE PROCESSTYPE command with START, NEXT, and END options is an SPI command and uses browsing in the same way as other CICS SPI commands. We are using these commands to find all process types available in the system. This command will also give the status of the process type, enabled or disabled, the audit level, the audit log and the repository.
- EXEC CICS STARTBROWSE PROCESS PROCESSTYPE(PROCESSTYPE-NAME) BROWSETOKEN(B-PROCESS)  
With each process type found by the INQUIRE PROCESSTYPE BROWSE, this command is used to initialize a browse token to be used to identify each process of that specified process type. This enables you to browse through all the processes of a particular type. The command is then followed by a GETNEXT command.
- EXEC CICS GETNEXT PROCESS(PROCESS-NAME) BROWSETOKEN(B-PROCESS) ACTIVITYID(ACTIVITY-ID)  
The GETNEXT command uses the browse token to find the next process of the process type specified on the related STARTBROWSE command. The command returns the process name and the activity ID of the root activity. It is necessary to know the activity ID in order to be able to browse the events relating to the activity.

- EXEC CICS STARTBROWSE EVENT BROWSETOKEN(B-EVENT) ACTIVITYID(ACTIVITY-ID)

This command is discussed in 4.5.7.2, “Tidy Up Children” on page 88, but is used here to browse through all the events of the root activity of the process.

- EXEC CICS GETNEXT EVENT(EVENT-NAME) BROWSETOKEN(B-EVENT) TIMER(TIMER-NAME) FIRESTATUS(FIRE-STATUS)

This command is discussed in 4.5.7.2, “Tidy Up Children” on page 88, but is used here to search for the INTEREST-CHANGED event together with a timer name that exists. Only process(M) defines the event INTEREST-CHANGED with a timer. Process(F) defines an event INTEREST-CHANGED, but this is an input event. We then know that we have found the process(M) and the relevant event.

- EXEC CICS INQUIRE TIMER(TIMER-NAME) ABSTIME(WS-ABSTIME) ACTIVITYID(ACTIVITY-ID) STATUS(TIMER-STATUS)

To establish when a timer is due to fire, we use the INQUIRE TIMER command. As this timer does not belong to the current activity, we need to use the ACTIVITYID parameter. The possible STATUS values that can be returned are:

- Expired - The timer expired normally.
- Forced - Expiry of the timer was forced by means of the FORCE TIMER command.
- Unexpired - The timer has not yet expired.

We can also determine the event (if any) associated with the timer. The program then writes to a TS queue the process name, process type, activity ID of the root, expiring time of the interest changed event, and some flags, to be used for the suspend.

- EXEC CICS SUSPEND ACQPROCESS(PROCESS-NAME)

SUSPEND prevents a process that the program has acquired from being reattached when events in its event pool are fired. The only process a program can suspend is the one that it has acquired in the current UOW. If the command gets a response back of INVREQ with a RESP2 value of 14, the process, in this case, has already completed. A RESP2 of 14 can also mean that the process is in cancelling mode. Either way, the process cannot be suspended.

- EXEC CICS CHECK ACQPROCESS

CHECK ACQPROCESS returns the completion status of the currently acquired process. Typically, it is used to check the success of a previous RUN or LINK ACQPROCESS command. The following information can be obtained from using this command:

- COMPSTATUS - Completion status of the process
- SUSPSTATUS - Suspended or not suspended
- MODE - The processing state of the process
- ABCODE - The abend code if the process has abended
- ABPROGRAM - The program that was in control at the time of the abend

The COMPSTATUS field contains one of four CVDA values:

- ABEND - The process has abended. Any children of the root activity will have been cancelled.
- FORCED - The process was forced to complete. For example, it was cancelled with a CANCEL ACQPROCESS command (see 7.3.5, “A Useful Tool” on page 169).
- INCOMPLETE - This could mean:
  - That it has not yet been run.
  - That it has returned from one or more activations but needs to be reattached in order to complete all its processing steps.
  - That it is currently active.
- NORMAL - The process has completed successfully.

The returned values of the MODE are discussed in 2.9, “Activity Modes Appearing During Process Execution” on page 23.

The information we are interested in from this command in this example is, however, the SUSPSTATUS. This status will confirm that the process has been suspended so that we can action the interest change.

- EXEC CICS RESUME ACQPROCESS

RESUME resumes a process that has previously been suspended, allowing the process to be reattached when events in its event pool are fired. If events that would normally have caused reattachment have occurred during the time the process was suspended, the process is reattached for all of these events. The only process a program can resume is the one it has acquired in the current UOW.



---

## Part 3. Designing and Developing Your Application



---

## Chapter 6. Designing Your Application

In this chapter, through the use of examples, we discuss the steps of business analysis and application design. This discussion is offered as a recommendation, since it is not the only way to be successful; however, in our experience, it is a proven way.

For substantiation, we refer to *A Professional's Guide to Systems Analysis*, ISBN 0-07-042948-0, and *Object-Oriented SSADM*, ISBN 0-13-309444-8. Other authors, such as Rumbaugh, Jacobson or Courtney, propose similar methodologies using other terminologies.

We use the words *business process* in terms of systems analysis and the words *CICS BTS business process* when talking about a CICS BTS application.

We do not intend to introduce or explain systems analysis methodologies. Therefore, we do not describe the intermediate steps leading from a business process model (enterprise level) to a business entity relationship model (operational level).

---

### 6.1 Business Process

A business process is a set of activities or tasks that are performed in sequence or in parallel to accomplish a specific goal. The process may be manual or automated and may comprise one or more activities or tasks.

For each function identified at the preceding step, the analyst must identify those processes or activities that are performed to support that function. They may be performed by one or more persons in one or more divisions of the company.

A major portion of the effort in the analysis phase is devoted to this analysis of user processes and activities. This analysis describes and defines each user process and activity to determine whether, and under what conditions, they may be automated. As with the functional analysis, these descriptions in and of themselves do not provide adequate insight into the flow, complexity, or interdependence of the user processing activities. All too often, each user process is treated as if it stood alone and was complete in and of itself. This individual treatment of processes does not capture the processing flow, nor does it necessarily detect related processes that are external to the immediate user's function. (Taken from *A Professional's Guide to Systems Analysis*, ISBN 0-07-042948-0).

Everyone dealing with information technology wants to improve reusability. In terms of reuse, it does not matter whether you talk about systems, entities, events, design components, programs, or functions.

Reusable components are necessary to increase the areas to be covered by information technology. Most business functions follow simple rules to develop simple elements. These elements may be combined to make a more complex element, which, together with other elements, can be used to build an element at the next level of complexity. Developing applications covering one level of complexity will prepare you to work on the next level of complexity.

For our purposes, we use the three-level analysis model based on Robert Anthony's organizational pyramid (*Robert Anthony, "Planning and Control Systems: A Framework for Analysis", Harvard Business Review, 1965*). He sees the organizational management arranged in a pyramid (Figure 68 on page 130 ).

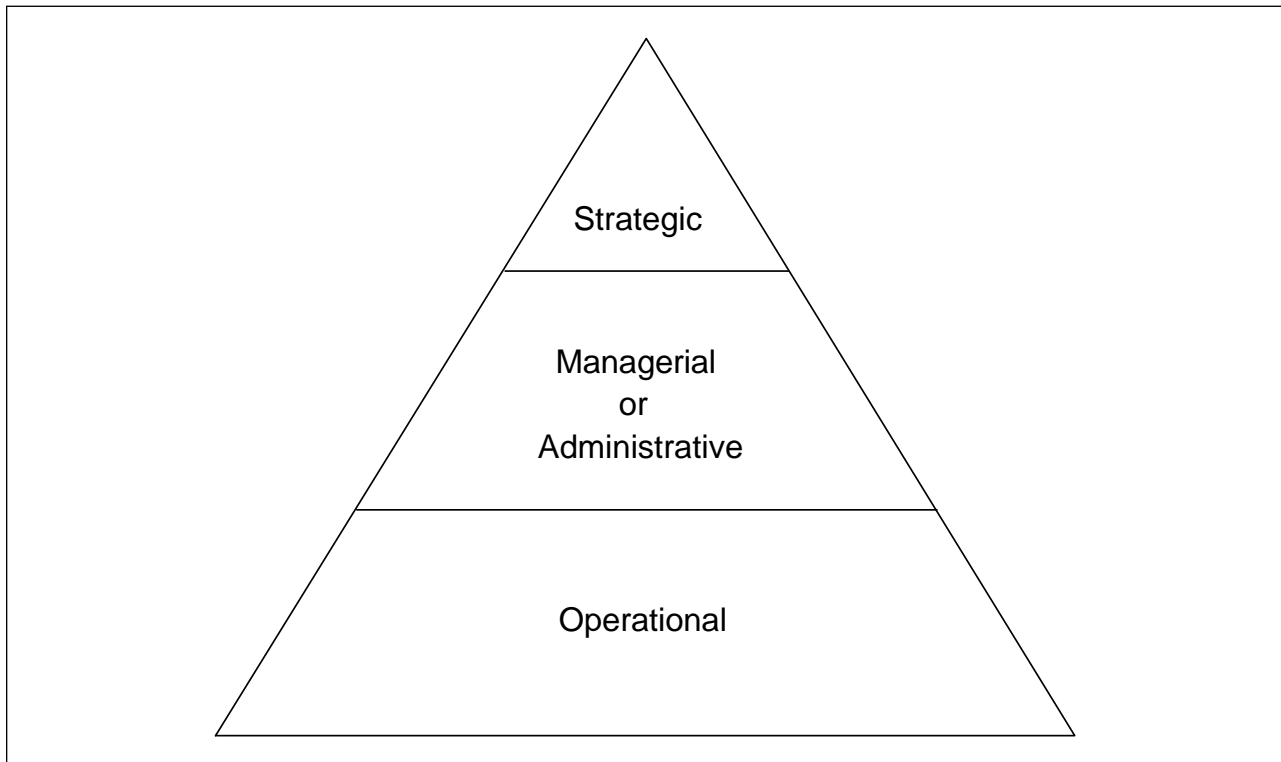


Figure 68. Robert Anthony's Organizational Pyramid

An organization is an entity with multiple levels. Each level is responsible for a different level of control. They have different data requirements and different views of the organization.

From the analysis point of view, we have to look at all levels because the part of business we have to analyze touches them all. However, they all have different requirements. Usually these requirements change from informational on the top and middle levels to operational on the lowest level. These requirements have to be seen from a wide angle at the top level, but a closer attention to the detailed context is needed at the operational level. This can be achieved with a top-down approach to analysis. This is of course not a task to be done only once; it is an iterative process. The resulting model can be shown like a data model, in three levels. From top to bottom, each level shows the business requirements in a more detailed format than the preceding level.

With its implemented functions, a CICS BTS process is used to support one or more business processes. In this sense, a business process is an internal economic illustration of time and location dependent actions required to achieve a certain goal (for example, personal finances).

Within a business process, there may be business functions. In terms of systems analysis, a business function is a decomposite view (a part of) the business process. On the different levels of our model, we then have:

1. Process entity set (business process)

2. Process entity subset (business function)
3. Business activity

A simplified example may be a bank department that deals with personal finances. This could be described as a business process. See Figure 69 on page 132 for a brief overview.

A business process can have several business functions. Each of these business functions can have several business activities. For example, if the business process is "Personal Finances", two of its business functions might be "Savings Contracts" and "Credit Contracts".

Within each business function, there may be several business activities. A business activity represents several steps (manual or automated) to achieve this business function's goal. Each step (business task) within a business activity, which is able to be automated, is subject to act as part of a CICS BTS process.

In such a business process, the advance of a mortgage, for example, is part of a business function. One of this function's business activities is mortgage selling. Another one may be the maintenance of an investment account. Therefore, each business activity represents one of several variations in this business process.

On this level, it is necessary to decide whether such a business activity can be automated (hence, supported by information technology). If so, this business activity can be represented by one or more CICS BTS processes.

A CICS BTS activity implements a specific (atomized) function (business task) of the business logic. The result of this step, together with those of other steps, may be combined to form a consistent result for business use. Such a result only originates after modelling to a process. This means that a step is, in most cases, useless unless it is executed in the context of a business activity. Because this step fulfills a specific task concerning the business activity, it may be reusable in other business activities of the same or another business process.

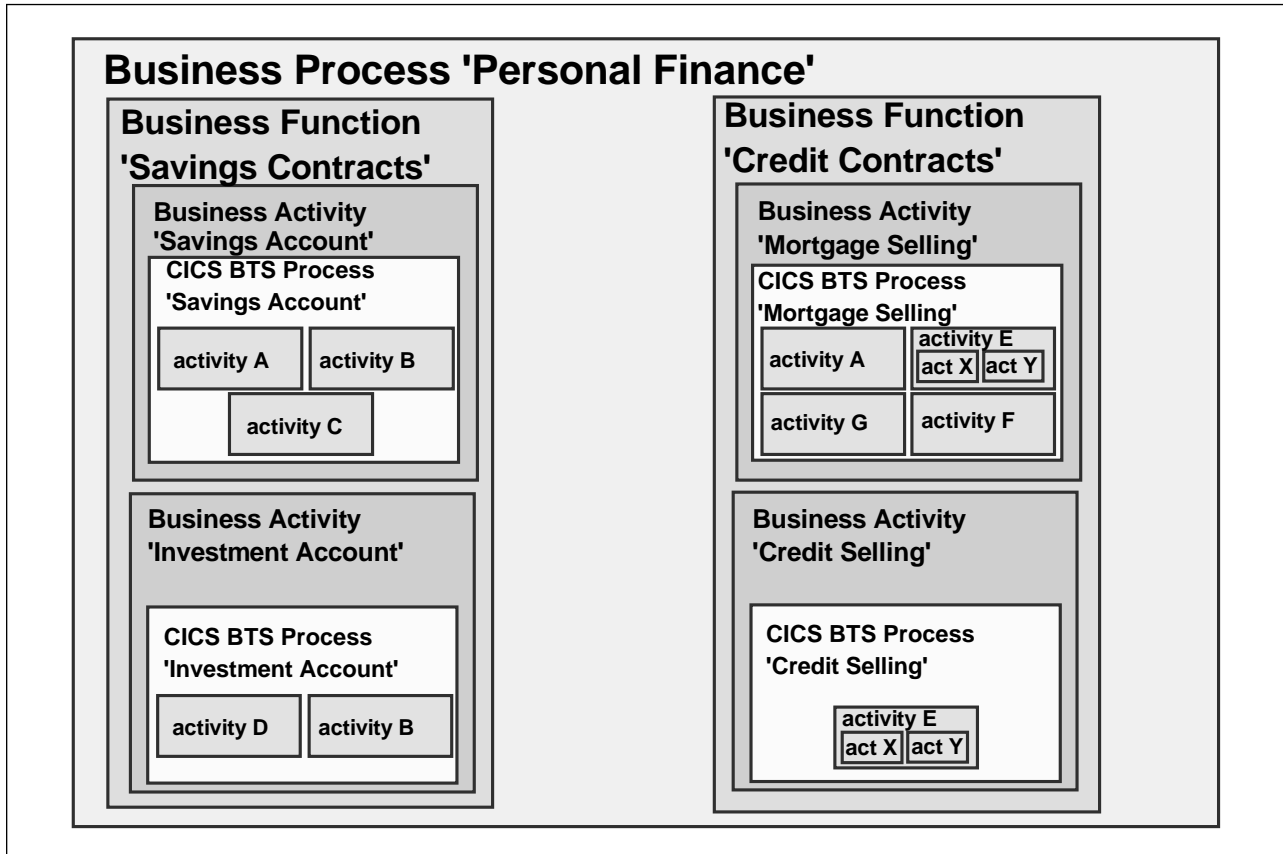


Figure 69. Illustration of a Business Process and Its Subsequent Elements

Typically, a CICS BTS process is the technical implementation of a business activity. For the calling program, it is not necessary to know how many CICS BTS activities the process consists of. However, it has access to the different CICS BTS activities if the application is designed this way.

## 6.2 Business Process Analysis

We do not claim to introduce business process analysis or even the different approaches of modelling. These are well explained in numerous publications. However, to provide a better understanding of the methodology, we give a simplified example of how these things work. During our analysis, we used *A Professional's Guide to Systems Analysis*, ISBN 0-07-042948-0. This book provides an extensive introduction into business process analysis.

### 6.2.1 Analysis of Requirements

For software developers, it becomes more important to examine the business requirements for a new application. The quality of the product arises from the quality of the analysis and the resulting structure of the business requirements. Market pressure and management requirements encourage use of a methodology that enables you to reuse your programs. A prerequisite is a well-structured model of the entities involved in the business function. Such a model is useful for the developer to understand the requirements. For business people, it is useful because it helps to analyze the business in a structured manner.

Reuse of programs at least means that you are able to use your program, or even parts of it, for several tasks. The minimum requirement is encapsulation of data (information hiding). At this point, we are not talking about object oriented programming. We can use the advantages of object oriented analysis and design; however, we develop the CICS BTS programs as procedural applications.

## 6.2.2 Example of a Business Process

We chose, as an example, a part of a business process model used in a bank. From this enterprise level part, we went into more detail (scoping) and examined the part *organization*. Within this part, a business process handles all organizational relationships. These relationships, for example, explain:

- Which groups form a department.
- Which tasks are performed by a group.
- Which roles can be found in this department.
- Which competencies are necessary to perform the tasks.
- Which employee has what competencies.
- How many employees work in group X.

With this example, we show how to develop a CICS BTS process model. We developed one process in this CICS BTS process model, adding a new employee and the job description to the database.

In Figure 70 on page 134 we use a part of a business process model. On the strategic level (enterprise level), the model knows the *business data* and *organization* business entities. Notice that this is not a data model. In fact, the entities are more or less the same, but we look at them from the business process analyst's point of view. It is a business process entity model that will be expanded to a process entity relationship model.

The entity relationship approach was first described in 1976 by Dr. Peter Chen in the first issue of *Transactions on Database Systems* (ACM publication).

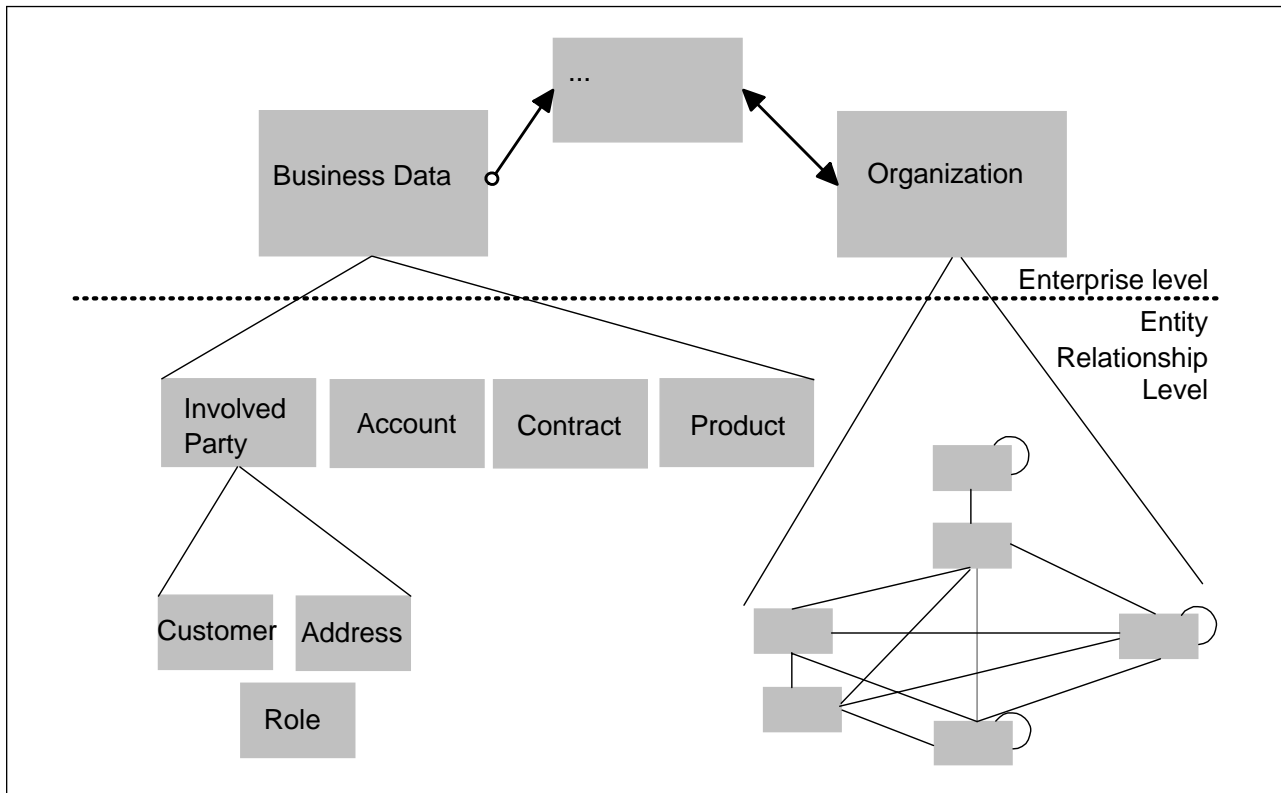


Figure 70. Part of the Business Process Model (1)

To work on a specific business function, scoping is necessary. Therefore, we define *organization* as the entity we work on and analyze this entity on the next level of the organizational pyramid. We assume the task is to develop an application that shows our business organization with all organizational units, the job descriptions, and all employees. We want to see:

- Who works on which job description.
- Which groups form a given department.
- How many job descriptions are in a given group and who works there.
- Which role is necessary to do this job.
- Which tasks do we want to be done when someone has a given role.
- Which competence is necessary to fulfil this task.
- Which employee has what competencies

The list, although not complete, provides an overview of what this application is to be used for.

Starting with the meta-model, we have analyzed the business requirements. The result is shown in the next level of the model (see Figure 71 on page 135 ). We show these analyzing activities in a simplified way, and we do not show the functional analysis process as well as the data analysis process. Our entity-relationship-model shows the entities we have to work on, their relationship to each other, and what this relationship describes.

Also, we added potential business activities for this model. To get these business activities, you have to discuss with the business people. To steer this discussion,



you may begin with identifying all possible entry points in this model. This means you must find which entity you need to start with when working on a specific business activity. We assumed that we have entry points on every single entity. To explain our example, we only use two entry points (job description and employee).

First, we defined all single entity steps (steps that only affect one entity). Then we defined all steps concerning relationships. Remember, we are not talking about a data model. However, you can build your data model from this business function model. It is useful to define all related business activities even when they are not necessary at this time, because when you have to expand your application, it is helpful to have all business activities defined. In this phase, you may have defined business activities that are not subject to be automated.

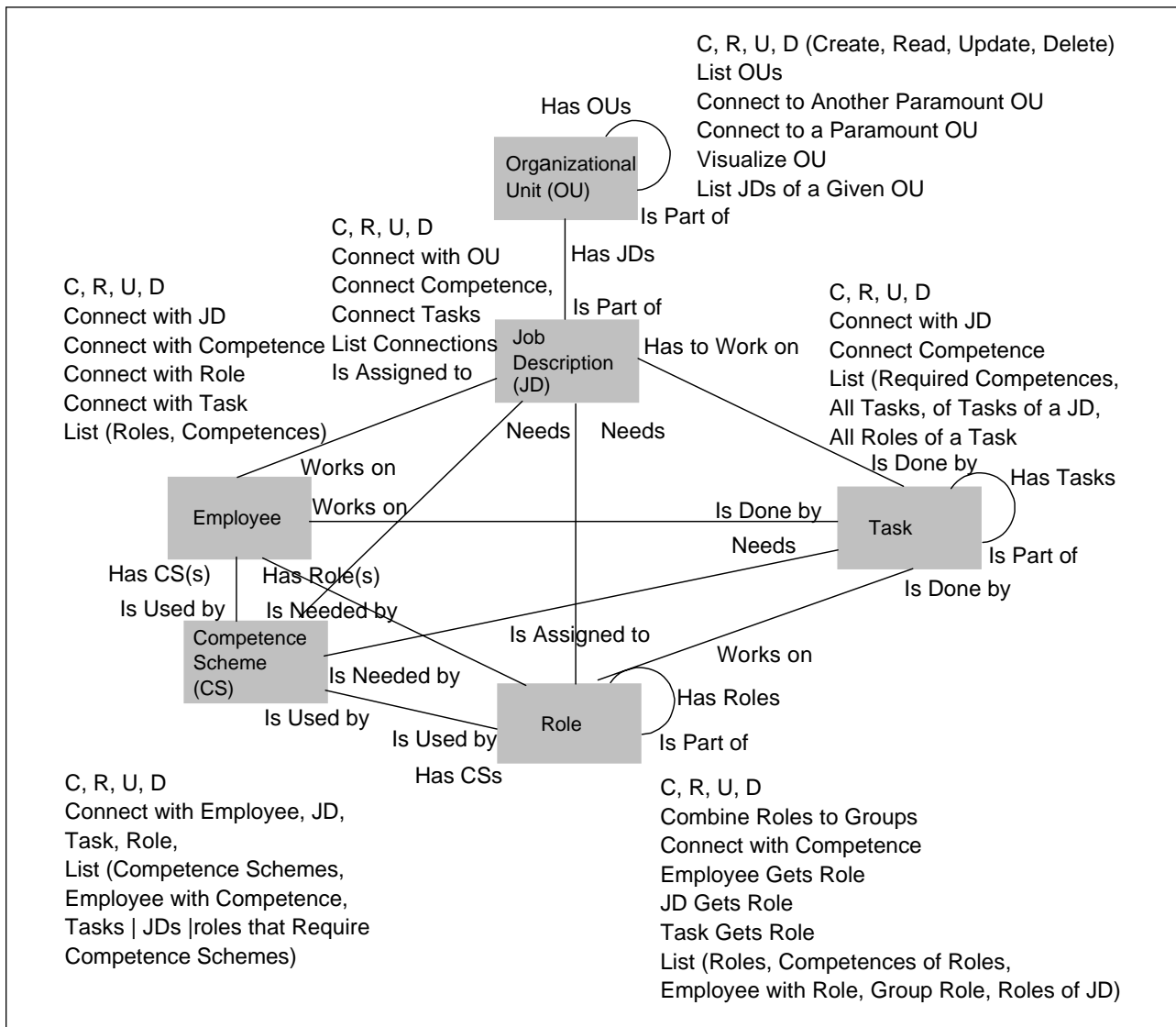


Figure 71. Part of the Business Process Model (2)

The example is very much simplified. Nevertheless, you can see how it works. Each defined business activity, or even business task, is a candidate to be implemented in an activity or process. You have to decide whether you prefer very

small activities (one activity for each step) or larger activities that cover entities and relationships.

We decided to show the implementation of the business activity needed to add a new employee to an organization (see Figure 72).

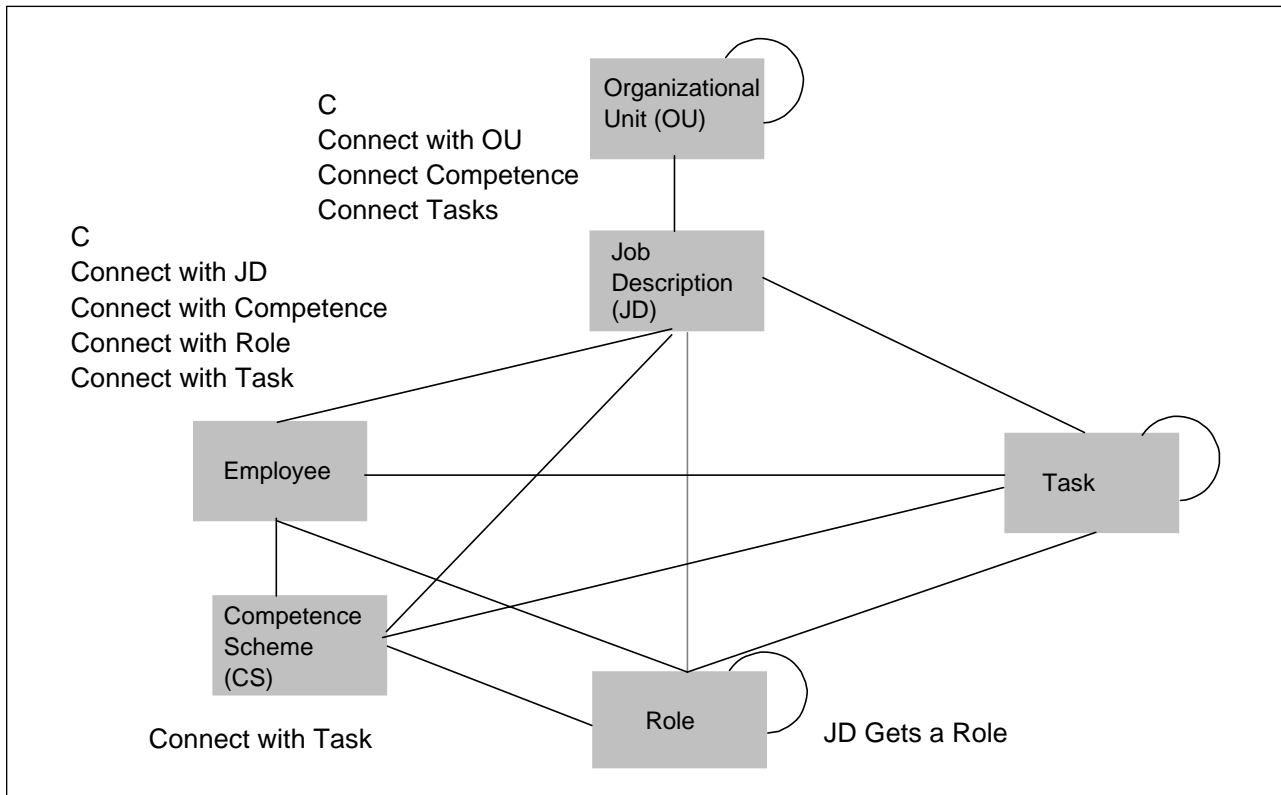


Figure 72. Transactions Needed to Add a New Employee

### 6.3 Application Design Extracted from a Business Process Model

Notice that we have not described the different steps of analysis, such as function analysis and data analysis, and we have not provided "what-if" scenarios. For more information, refer to the books mentioned earlier.

To identify the business activity, you need refer to the results of your discussion with the business people. The combination of business processes, business information (from the discussion), and business rules lead you to an information collection that is to be used to define the CICS BTS processes you need to support your business. Usually, this is done in an iterative process while developing your business process model, which becomes more and more detailed. The more detailed your model is, the more you need to define a scope. On the first level, it may be possible to develop the entire model. The subsequent levels need scoping. Therefore, it is necessary to define a communication model showing all communication paths of the scope used. These paths define all internal (scope) and external (entire model) relationships concerning the business entities.

As mentioned earlier, we have decided to define a CICS BTS process to add a new employee to our organization. We also have some assumptions:

- The organizational unit that the job description is to be connected to exists.
- The tasks exist.
- The competence scheme to be used exists.
- The roles needed exist.
- The roles are already connected to competence schemes.
- All data needed is provided by the transaction EMPM.

From this requirement we developed a CICS BTS process. The logical work and event flow of this process is shown in Figure 73 through Figure 77 on page 141.

The business control logic in our CICS BTS process is implemented in the root activity that runs with the transaction ID EMPR. This activity is the parent of three child activities. These are executed with the activity names NEWJD (asynchronous), NEWEMPL (asynchronous), and NEWEMJD (synchronous).

The child activities NEWJD and NEWEMPL run two subsequent child activities each.

### 6.3.1 Process NEWEMPLOYEE, Root Activity, Event DFHINITIAL

Figure 73 illustrates the event logic with the reattachment event DFHINITIAL of the root activity of our process to add a new employee.

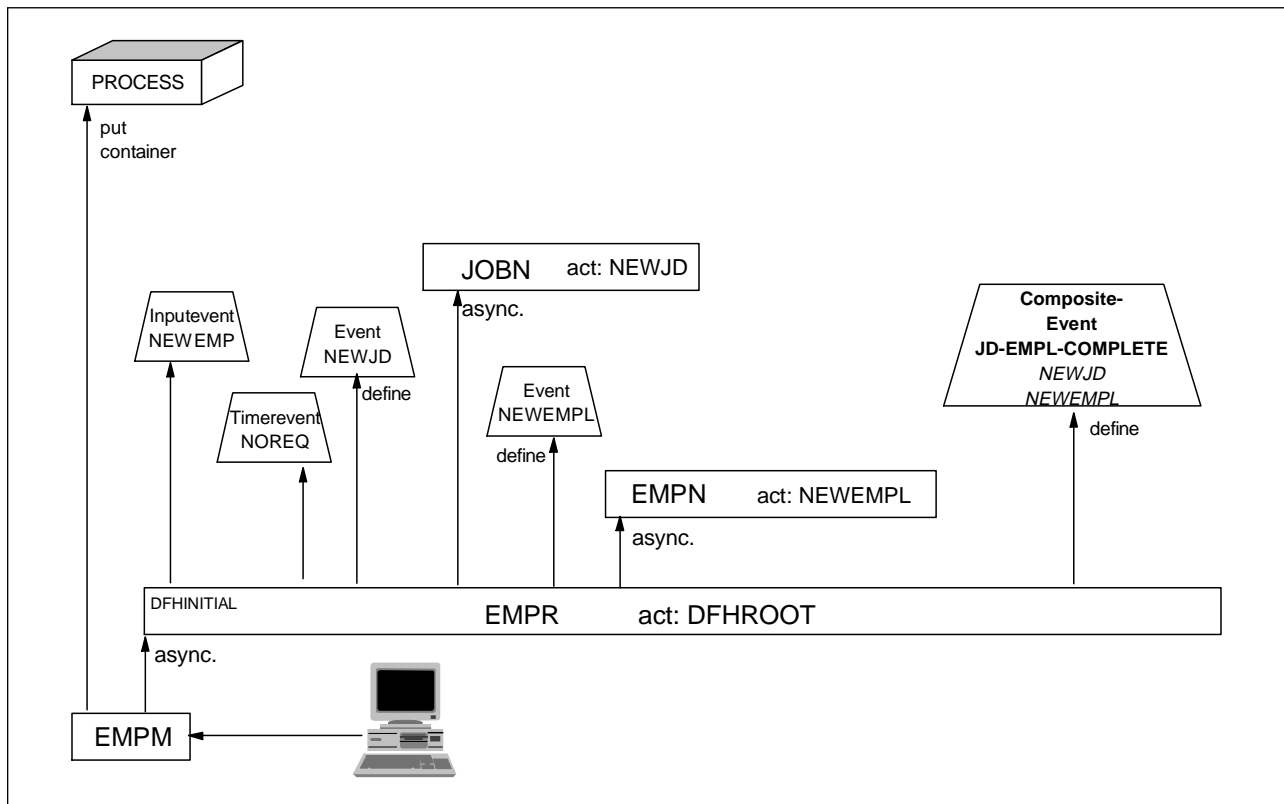


Figure 73. Process NEWEMPLOYEE, Event DFHINITIAL

- The terminal-oriented transaction EMPM writes the container PROCESS. Then it defines and runs the process EMPL asynchronously. This is a generic name. At run-time, a unique identifier (for example, tasknumber) has to be added to

this name. With the RETURN command of the transaction EMPM CICS BTS starts the process.

- The process is executed with transaction ID EMPR.
- The root activity first defines the input event NEWEMP to insure that the process result has been read before the process ends, and CICS deletes the container. To be sure that the process ends when nobody fires the input event, a timer is defined that expires after 31 days.
- It defines the activity NEWJD. With this definition, it also defines the event NEWJD (atomic event).
- This activity is executed asynchronously by the transaction JOBN.
- Next, the activity NEWEMPL is defined and it runs as transaction EMPN.
- Before returning to CICS, the composite event JD-EMPL-COMPLETE is defined. This event is an event with the boolean expression AND, and the subevents NEWJD and NEWEMPL are added.

### 6.3.2 Process NEWEMPLOYEE, Activity NEWJD, Event DFHINITIAL

Figure 74 illustrates the event logic with the reattachment event DFHINITIAL of the activity NEWJD of our process defined and started by the root activity.

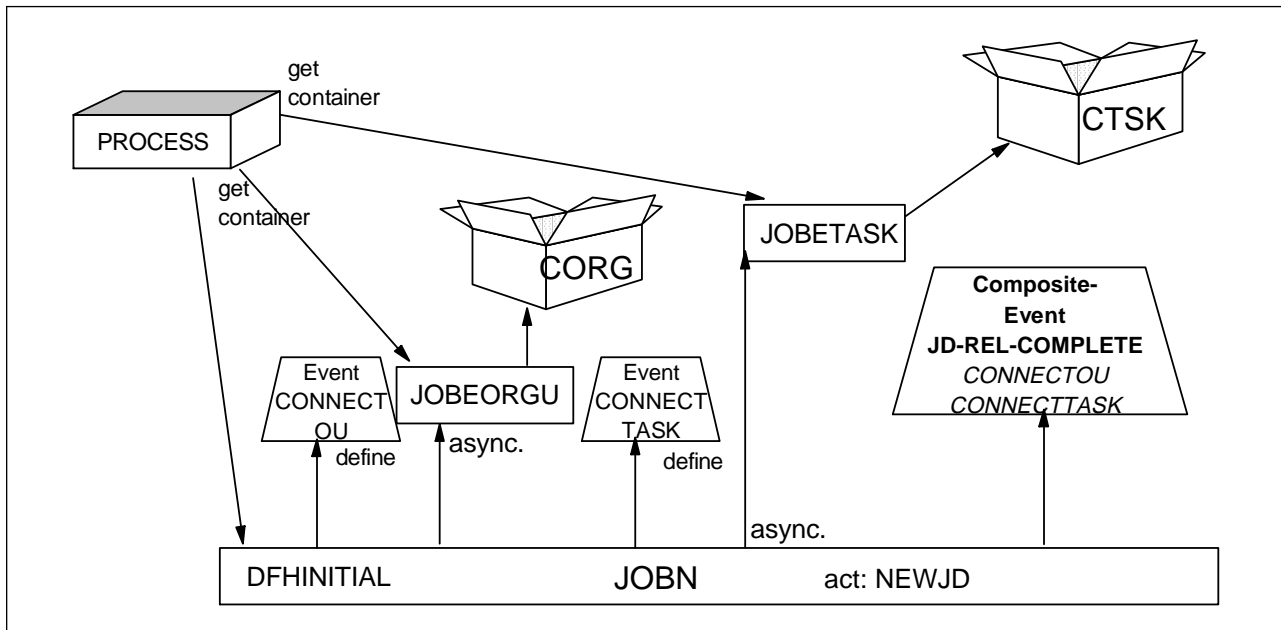


Figure 74. Process NEWEMPLOYEE, Activity NEWJD, Event DFHINITIAL

- The activity NEWJD reads the container PROCESS.
- It defines and runs the activities JOBEORGU and JOBETASK asynchronously.
- Finally, it defines the composite event JD-REL-COMPLETE.
- Each of the activities on the last level reads the container PROCESS and writes data to its activity container to specify the execution result.



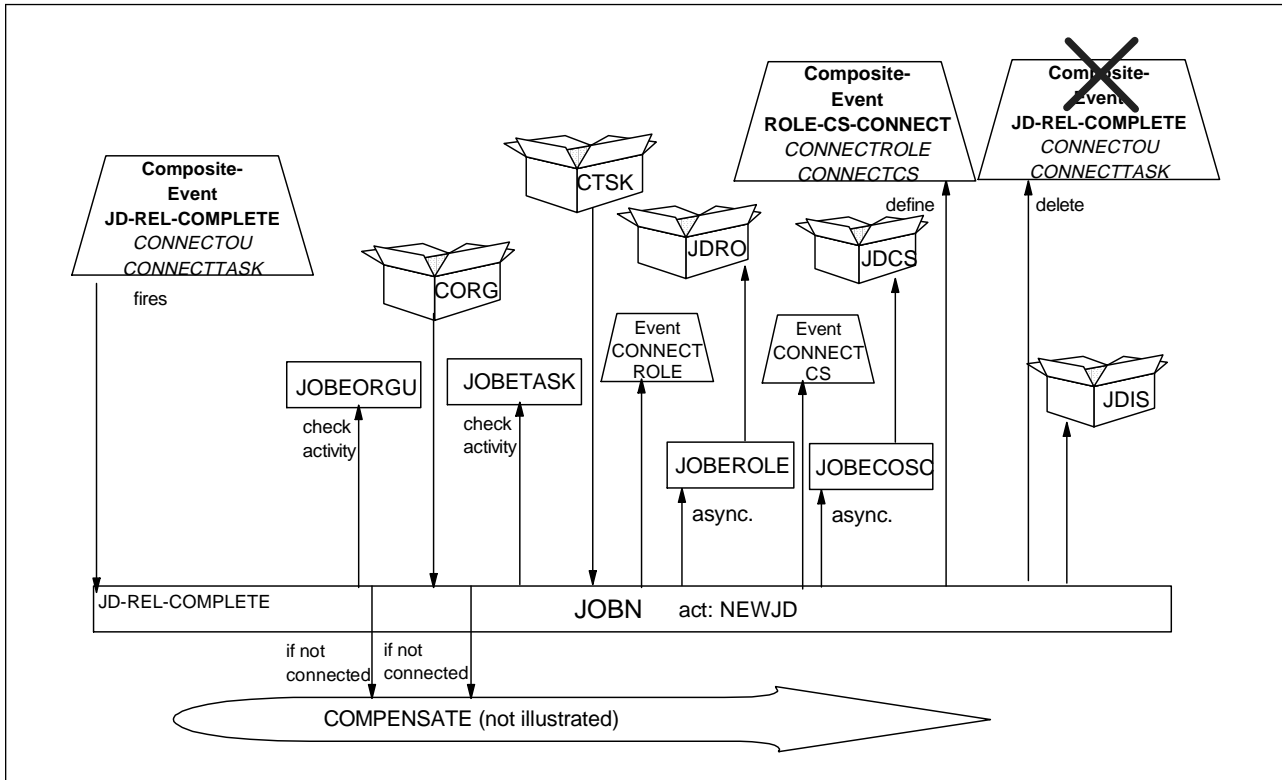


Figure 76. Process NEWEMPLOYEE, Event JD-REL-COMPLETE

- When both activities JOBEORGU and JOBETASK complete, the activity NEWJD becomes reactivated by CICS BTS.
- It checks whether the activities were successfully executed. If not, we start a compensation routine that cancels all changes. This routine is not illustrated.
- Then the activities JOBEROLE and JOBECOSC are defined and executed asynchronously.
- For reattachment, the composite event ROLE-CS-CONNECT is defined with the AND parameter.
- Before returning to CICS, the activity deletes the composite event JD-REL-COMPLETE and stores the result in the container JDIS.

### 6.3.5 Process NEWEMPLOYEE, Activity NEWEMPL, Event EMPLREL-COMPLETE

When the children have finished, the composite event EMPLREL-COMPLETE fires and causes the activity NEWEMPL to be reactivated. Figure 77 on page 141 shows the logic during this activation.

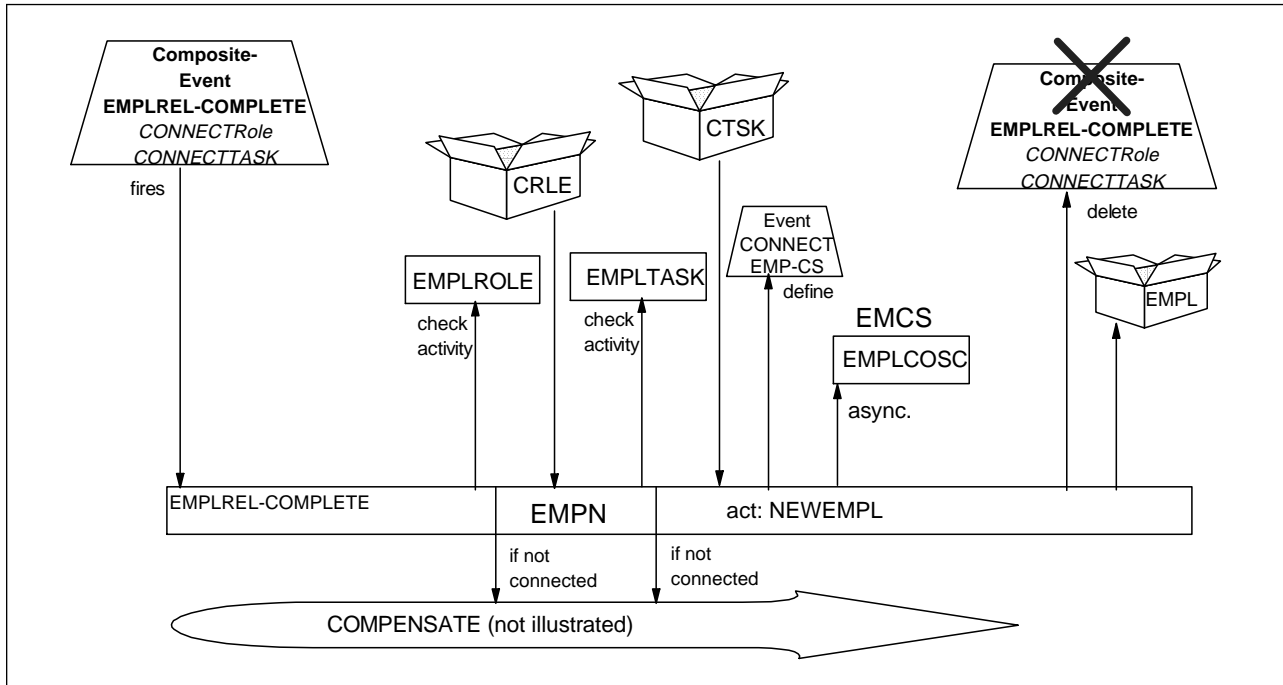


Figure 77. Process NEWEMPLOYEE, Event EMPLREL-COMPLETE

- When both activities EMPLROLE and EMPLTASK complete, the activity NEWEMPL becomes reactivated by CICS BTS.
- It checks the activities to discover whether they finished successfully. If not, we start a compensation routine, which is not illustrated.
- If OK, the activity reads the children's containers.
- Then the activity EMPLCOSC is defined and executed asynchronously.
- Before returning to CICS, the activity deletes the composite event EMPLREL-COMPLETE and stores the result in the container EMPL.

### 6.3.6 Process NEWEMPLOYEE, Activity NEWJD, Event ROLE-CS-CONNECT

When the children JOBEROLE and JOBECOSC have finished and executed an EXEC CICS RETURN ENDACTIVITY command, the composite event ROLE-CS-COMPLETE fires and causes the activity NEWJD to be reactivated. Figure 78 on page 142 shows the logic during this activation.

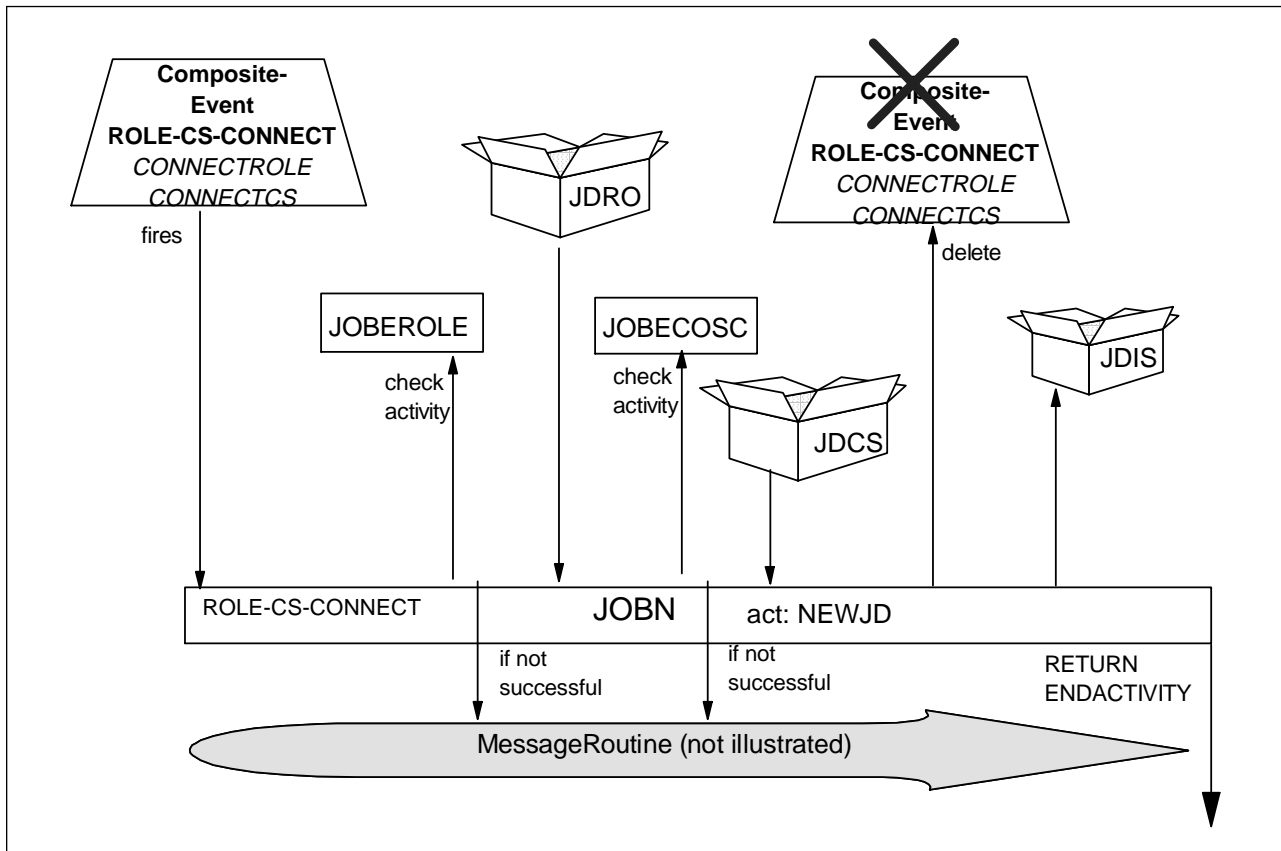


Figure 78. Process NEWEMPLOYEE, Event ROLE-CS-CONNECT

- This child activity is reattached when its children JOBBEROLE and JOBECOSC have finished.
- Both activities are checked. Because their successful execution is not so important, the process does not stop if they were not successful. In this case, an additional message routine is called to translate the program specific messages into process specific messages considering the process context.
- Then the composite event is deleted and the activity's result stored in the container JDIS.

### 6.3.7 Process NEWEMPLOYEE, Activity NEWEMPL, Event CONNECT-EMP-CS

When the child EMPLCOSC has finished and executed an EXEC CICS RETURN ENDACTIVITY command, the completion event CONNECT EMP-CS fires and causes the activity NEWEMPL to be reactivated. Figure 79 on page 143 shows the logic during this activation.





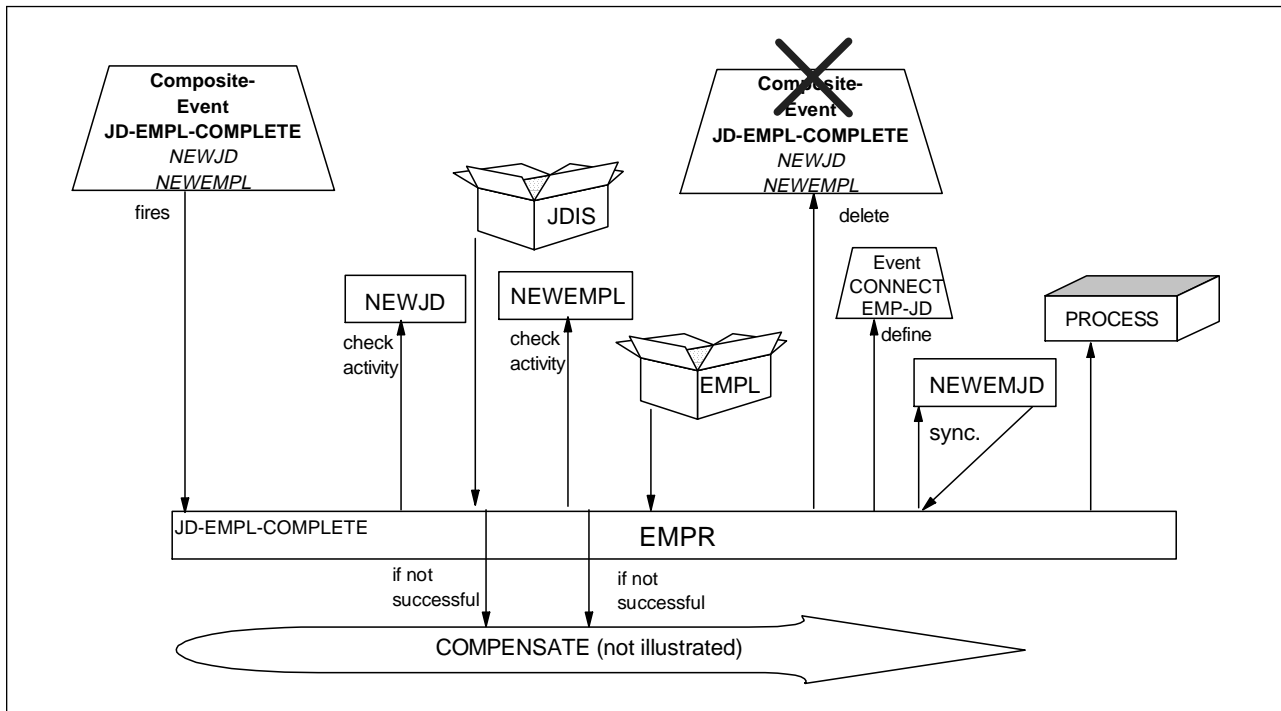


Figure 80. Process NEWEMPLOYEE, Event JD-EMPL-COMPLETE

- The root activity becomes reactivated by CICS BTS when all subevents of the composite event JD-EMPL-COMPLETE have fired.
- It checks the activities for successful execution.
- If OK the containers are read.
- Next, the composite event is deleted.
- Then it defines and runs the activity NEWEMJD synchronously.
- Finally, the container PROCESS is written.

### 6.3.9 Process NEWEMPLOYEE, Root Activity, Input Event NEWEMP or Timer Event NOREQ

Finally, the timer event NOREQ, or the input event NEWEMP, fires and causes the root activity to be reactivated. Figure 81 on page 145 shows the logic during this activation.

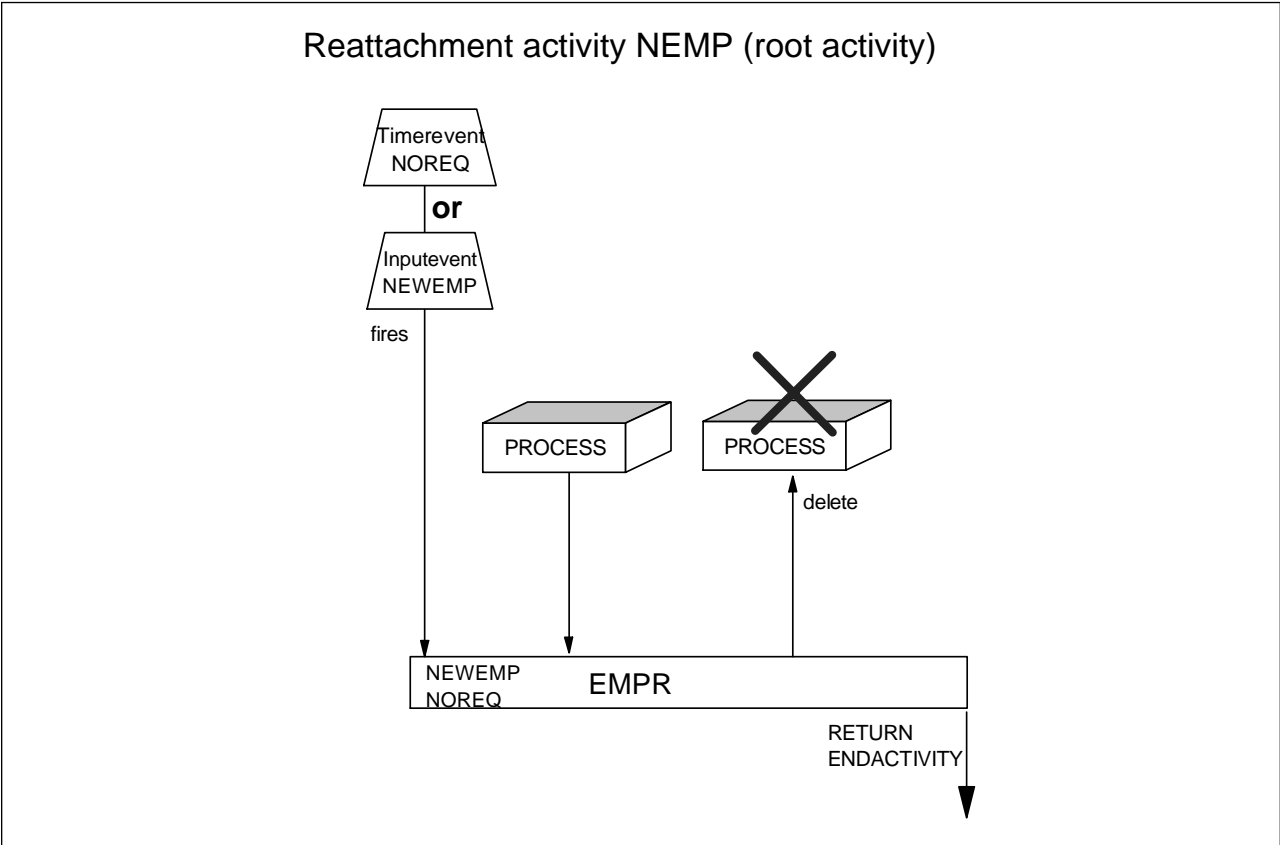


Figure 81. Process NEWEMPLOYEE, Input Event NEWEMP

- When a non-CICS BTS transaction has read the process result from the container PROCESS, it fires the input event NEWEMP by running the process.
- The root activity then reads the container to examine whether the process can be finished.
- If yes, the container is deleted, and the process ends with a RETURN ENDACTIVITY command.



---

## Chapter 7. Developing Your Application

Developing a CICS BTS application is just like developing any other application. It includes analysis, design, coding, and testing steps. Program coding development can vary from other applications regarding language, company standard, and even more importantly, the facility provided by the product itself. For CICS BTS applications, we recommend the following steps in the program development.

---

### 7.1 Program Development

Because CICS BTS applications are event driven, we recommend starting with the event-flow diagram. From this diagram, we identify the activities and the data shared or passed between the activities in the containers.

- Start with the event-flow diagram.
- Identify the activities.
- Assign the functions to be implemented to activities.
- Develop basic root activity logic, but do not implement all activities.
- Start testing only a few activities.
- Identify the information shared and passed in the containers.
- Increase the number of activities used.

---

### 7.2 Pseudo-Code of the Company Organization Employee Application

The following sections contain pseudo-code of the application designed in Chapter 6, “Designing Your Application” on page 129. We illustrate the program development part of the whole development cycle. We cover four major programs in this chapter for the application:

- EMPLMAIN - Screen-handling and process definition program
- EMPLROOT - Root Activity business logic control program
- EMPLACTM - New Employee business activity program
- JOBEACTM - New Job description business activity program

You will find the detailed development log in Appendix A, “Company Organization Application Development Log” on page 239. This log is like a diary of the application development based on the business process model in the previous chapter. Here, you will also find JCL, resource definitions, CBAM screen displays, error message codes, and short descriptions of how we approached the problem and solved it.

Note that event names in the pseudo-code of this chapter are in long format (xxxxyy-complete or xxx-yyy-event), while the pictures in Chapter 6, “Designing Your Application” on page 129 are labeled with short names that fit.

Note that we develop the application, as in real life, using an iterative approach. The figures can be slightly different from the pseudo-code shown in this chapter. The main objective of the exercise is to show the transformation from a business transaction process model to CICS BTS implementation. We describe CICS BTS

functions in more detail when we describe HOLIDAY, MORTGAGE, and FINANCE applications.

Table 8 shows the short and long event names.

<i>Table 8. Short and Long Event Names</i>	
<b>Short Names on Figures</b>	<b>Long Names in Programs</b>
NEWEMPL	EMPL-EVENT
CONNECTRole	EMPL-ROLE-EVENT
CONNECTTask	EMPL-TASK-EVENT
EMP-CS	EMPL-COSC-EVENT
NEWJD	JOB-EVENT
CONNECTOU	JOBE-ORGU-EVENT
CONNECTTASK	JOBE-TASK-EVENT
CONNECTROLE	JOBE-ROLE-EVENT
CONNECTCS	JOBE-COSC-EVENT
NEWEMP	NEW-EMPL-IEVENT

## 7.2.1 Pseudo-Code of the Main Program EMPLMAIN

The initial request is to start a Company Organization Employment menu for users to key in the employee details. After clearing the screen, type EMPM to invoke the program EMPLMAIN, which will send and receive a map, and define and run the process.

Figure 82 shows, in COBOL pseudo-code, how EMPLMAIN creates and starts the new employee process.

```
Identification Division.
Program-id. EMPLMAIN.
Environment Division.
Data Division.
Working-Storage Section.
*
  05 TASK-NUMBER-COMP          PIC 9(8) COMP.
  05 PROCESS-NAME.
    10 PROCESS-ID             PIC X(4) VALUE 'EMPL'.
    10 TASK-NUMBER            PIC 9(7).
    10 FILLER                  PIC X(25) VALUE spaces.
  05 MESSAGE-TEXT.
    10 MESSAGE-RECEIPT        PIC X(30) Value
        'Your confirmation number is:'.
    10 MESSAGE-PROCESS-NAME    PIC X(16).
*
* GET MAPSET COPY BOOK
COPY COMPBOOK.
*
* GET 3270 keyboard COPY BOOK
COPY DFHAID.
*
LINKAGE SECTION.

PROCEDURE DIVISION.

  IF EIBCALEN = ZERO
    EXEC CICS SEND MAP('COMPANY') . . . . .
    EXEC CICS RETURN TRANSID('EMPM') COMMAREA(WE-COMMAREA) . . .
  ELSE
    EXEC CICS RECEIVE MAP('COMPANY') . . . . .
  END-IF.
  MOVE EIBTASKN          TO TASK-NUMBER-COMP.
  MOVE TASK-NUMBER-COMP TO TASK-NUMBER.

  EXEC CICS DEFINE PROCESS(PROCESS-NAME)
        PROCESSTYPE('COMP-ORG')
        TRANSID('EMPR')
        PROGRAM('EMPLROOT')
  END-EXEC.

  EXEC CICS PUT CONTAINER(PROCESS-NAME)
        FROM(PROCESS-CONTAINER)
        FLENGTH(PC-LENGTH)
        ACQPROCESS
  END-EXEC.

  EXEC CICS RUN ACQPROCESS
        SYNCHRONOUS
        RESP(RESP-AREA)
        RESP2(RESP2-AREA)
  END-EXEC.
```

Figure 82 (Part 1 of 2). Pseudo-Code for the EMPLMAIN Program

```

* RETURN FROM PROCESS
* CHECK RESP AREAS FOR NORMAL COMPLETION

EXEC CICS GET CONTAINER(PROCESS-NAME)
           INTO(PROCESS-CONTAINER)
           FLENGTH(PC-LENGTH)
           ACQPROCESS

END-EXEC

IF RESP-AREA = DFHRESP(NORMAL) THEN
  MOVE PROCESS-NAME      TO MESSAGE-PROCESS-NAME.
  MOVE MESSAGE-TEXT      TO MSGO.
  EXEC CICS SEND
           MAP('COMPANY')
           MAPSET('COMPEMS')
           FROM(COMPANYO)
           DATAONLY
END-EXEC
ELSE
* PUT CODE HERE TO HANDLE NON NORMAL RESP CODE.
  PERFORM TERM-BAD-RETURN
END-IF.

EXEC CICS RETURN
           TRANSID('EMPM')
           COMMAREA(WO-COMMAREA)
END-EXEC.
End Program.

```

Figure 82 (Part 2 of 2). Pseudo-Code for the EMPLMAIN Program

## 7.2.2 Pseudo-Code of the Root Activity Program of the NEWEMPLOYEE Process

The root activity program EMPLROOT is designed to be reattached by CICS BTS when events in which it is interested are triggered. EMPLROOT requests CICS BTS to run asynchronously activities NEWEMPL and NEWJD. NEWEMPL creates employee personal related information, while NEWJD creates job-related information. It will be triggered again when both NEWEMPL and NEWJD finish successfully.

Figure 83 on page 151 shows, in COBOL pseudo-code, the employment root activity program, EMPLROOT.

We put the code that deals with containers and events in the third part of the figure and the code that deals with the definition of the activities in the fourth part of the figure. This code relates to Figure 73 on page 137.

We put the code that deals with the completion of the execution of the activities (NEWEMPL and NEWJD) in the fifth part of the figure which relates to Figure 80 on page 144 in Chapter 6, "Designing Your Application" on page 129.



```

IDENTIFICATION DIVISION.
PROGRAM-ID.  EMPLROOT.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
*
05  WS-STATUS                PIC S9(8) comp.
05  EVENT-TYPE               PIC S9(8) comp.
05  RESP-AREA                PIC S9(8) comp.
05  RESP2-AREA               PIC S9(8) comp.
05  ABEND-CODE                PIC X(4).
05  PROCESS-NAME.
    10  PROCESS-ID            PIC X(4) VALUE 'EMPL'.
    10  TASK-NUMBER           PIC 9(7).
    10  FILLER                 PIC X(25) VALUE SPACES.
*
*  EVENTS
*
05  EVENT-NAME                PIC x(16).
    88  DFH-INITIAL            value 'DFHINITIAL'.
    88  JD-EMPL-COMPLETE       value 'JD-EMPL-COMPLETE'.
    88  NEW-EMPL-IEVENT        value 'NEW-EMPL-IEVENT'.

05  SUB-EVENT-TYPE            PIC x(16).
    88  JOB-EVENT              value 'JOB-EVENT'.
    88  EMPL-EVENT             value 'EMPL-EVENT'.
*
*  CONTAINERS
*
COPY EMPLCONT.
*
*  ERROR AND TRACE MESSAGES
*
05  COMMON-MSG-AREA.
    10  COMMON-MSG             PIC X(40).
05  COMMON-MSG-LEN            PIC S9(4) VALUE +44 COMP.

05  INVALID-EVENT.
    10  IE-MESSAGE             PIC X(40) VALUE
        'INVALID EVENT RECEIVED      '.

05  INVALID-SUBEVENT.
    10  IE-MESSAGE             PIC X(40) VALUE
        'INVALID SUB EVENT RECEIVED   '.

05  INVALID-RETRIEVE-SUBEVENT.
    10  IE-MESSAGE             PIC X(40) VALUE
        'INVALID Retrieve subEVENT    '.

05  BAD-RETRIEVE-MESSAGE.
    10  BAD-RETRIEVE-MSG       PIC X(40) VALUE
        'UNKNOWN EVENT FORM RETRIEVE IN EMPLROOT '.

LINKAGE SECTION.

```

Figure 83 (Part 1 of 6). Pseudo-Code for the EMPLROOT Program

```

PROCEDURE DIVISION.
BEGIN-PROCESS.

    EXEC CICS RETRIEVE REATTACH EVENT(EVENT-NAME)
        RESP(RESP-AREA) RESP2(RESP2-AREA)
    END-EXEC.
    IF RESP-AREA NOT = DFHRESP(NORMAL) THEN
        MOVE BAD-RETRIEVE-MESSAGE TO COMMON-MSG-AREA
        PERFORM ABEND-PROCESS
    END-IF.

    EVALUATE TRUE
        WHEN DFH-INITIAL
            PERFORM INITIAL-ACTIVITY
            PERFORM NEW-EMPLOYEE-ACTIVITY
            PERFORM NEW-JOBDESCR-ACTIVITY
            PERFORM PUT-ROOT-CONTAINER
        WHEN JD-EMPL-COMPLETE
            PERFORM JD-EMPL-CHECK
        WHEN NEW-EMPL-IEVENT
            EXEC CICS DELETE EVENT('NEW-EMPL-IEVENT')
            END-EXEC
        WHEN OTHER
            * UNEXPECTED EVENT
            * WRITE ERROR MESSAGE TO CICS LOG
            MOVE INVALID-EVENT TO COMMON-MSG-AREA
            PERFORM ABEND-PROCESS

    END-EVALUATE.

    EXEC CICS RETURN END-EXEC.

JD-EMPL-CHECK.
* CHECK SUB EVENTS FOR THE COMPOSITE EVENT

    EXEC CICS RETRIEVE SUBEVENT(SUB-EVENT-TYPE)
        EVENT(EVENT-NAME)
        EVENTTYPE(EVENT-TYPE)
        RESP(RESP-AREA)
        RESP2(RESP2-AREA)

    END-EXEC.

    IF RESP-AREA NOT = DFHRESP(NORMAL)
        MOVE INVALID-RETRIEVE-SUBEVENT TO COMMON-MSG-AREA
        PERFORM ABEND-PROCESS
    END-IF.

    EVALUATE TRUE
        WHEN EMPL-EVENT
            PERFORM EMPL-CHECK
        WHEN JOB-EVENT
            PERFORM JOB-CHECK
        WHEN OTHER
            * UNEXPECTED SUBEVENT
            MOVE INVALID-SUBEVENT TO COMMON-MSG-AREA
            PERFORM ABEND-PROCESS
    END-EVALUATE.

```

Figure 83 (Part 2 of 6). Pseudo-Code for the EMPLROOT Program

```

* ALL COMPOSITE SUB-EVENTS OKAY DELETE COMPOSITE EVENT

PERFORM GET-ROOT-CONTAINER.

IF  EMPL-OKAY = 'Y' and JOB-OKAY = 'Y'
    EXEC CICS DELETE EVENT('JD-EMPL-COMPLETE')
        END-EXEC
    PERFORM EMPL-JOB-CONNECT
ELSE
    IF  EMPL-OKAY = 'Y' and JOB-OKAY = 'O'
        EXEC CICS DELETE EVENT('JD-EMPL-COMPLETE')
            END-EXEC
    ELSE
        IF  EMPL-OKAY = 'N' and JOB-OKAY = 'N'
            .
        END-IF
    END-IF
END-IF.

INITIAL-ACTIVITY.
* GET PROCESS NAME AND PROCESS CONTAINER
EXEC CICS ASSIGN PROCESS(PROCESS-NAME)
        RESP(RESP-AREA)
        RESP2(RESP2-AREA)

END-EXEC.

EXEC CICS GET CONTAINER(PROCESS-NAME)
        INTO(PROCESS-CONTAINER)
        FLENGTH(PC-LENGTH)
        PROCESS

END-EXEC.

EXEC CICS DEFINE INPUT EVENT('NEW-EMPL-IEVENT')
        RESP(RESP-AREA)
        RESP2(RESP2-AREA)

END-EXEC.

EXEC CICS DEFINE COMPOSITE EVENT('JD-EMPL-COMPLETE') AND
        RESP(RESP-AREA)
        RESP2(RESP2-AREA)

END-EXEC.

* DEFINE ONE COMPOSITE THEN TWO COMPLETION EVENTS BEING
* THE SUB-EVENTS.
* THEN RUN TWO ASYNCHRONOUS ACTIVITIES AS SUB-EVENTS

```

Figure 83 (Part 3 of 6). Pseudo-Code for the EMPLROOT Program

NEW-EMPLOYEE-ACTIVITY.

- \* NEW-EMPLOYEE-ACTIVITY RUN WITH A DIFFERENT USERID
- \* FOR SECURITY REASONS. THIS WILL CAUSE A CONTEXT SWITCH.
- \* THE NEW USERID IS SPECIFIED ON THE DEFINE ACTIVITY STATEMENT.

```
EXEC CICS DEFINE ACTIVITY('NEWEMPL')
      TRANSID('EMPN')
      PROGRAM('EMPLACTM')
      EVENT('EMPL-EVENT')
      USERID('CY-KJ-UC')
      RESP(RESP-AREA)
      RESP2(RESP2-AREA)
```

END-EXEC.

```
EXEC CICS ADD SUBEVENT('EMPL-EVENT')
      EVENT('JD-EMPL-COMPLETE')
      RESP(RESP-AREA)
      RESP2(RESP2-AREA)
```

END-EXEC.

```
EXEC CICS RUN ACTIVITY('NEWEMPL')
      ASYNCHRONOUS
      RESP(RESP-AREA)
      RESP2(RESP2-AREA)
```

END-EXEC.

NEW-JOBDESCR-ACTIVITY.

- \* NEW-JOBDESCR-ACTIVITY RUN WITH A DIFFERENT USERID
- \* FOR SECURITY REASONS. THIS WILL CAUSE A CONTEXT SWITCH.
- \* THE NEW USERID IS SPECIFIED ON THE DEFINE ACTIVITY STATEMENT.

```
IF WS-JOB-DESCRIPTION = SPACES or LOW-VALUE
  MOVE 'O' TO JOB-OKAY
```

Else

```
EXEC CICS DEFINE ACTIVITY('NEWJD')
      TRANSID('JOBN')
      PROGRAM('JOBEACTM')
      EVENT('JOB-EVENT')
      USERID('CY-KJ-UC')
      RESP(RESP-AREA)
      RESP2(RESP2-AREA)
```

END-EXEC

```
EXEC CICS ADD SUBEVENT('JOB-EVENT')
      EVENT('JD-EMPL-COMPLETE')
      RESP(RESP-AREA)
      RESP2(RESP2-AREA)
```

END-EXEC

```
EXEC CICS RUN ACTIVITY('NEWJD')
      ASYNCHRONOUS
      RESP(RESP-AREA)
      RESP2(RESP2-AREA)
```

END-EXEC.

Figure 83 (Part 4 of 6). Pseudo-Code for the EMPLROOT Program

```

JOB-CHECK.

    EXEC CICS CHECK ACTIVITY('NEWJD')
                COMPSTATUS(WS-STATUS)
                ABCODE(ABEND-CODE)

    END-EXEC.

    IF WS-STATUS NOT = DFHVALUE(NORMAL) THEN
        PERFORM JOB-NOT-COMPLETE
    ELSE
        PERFORM JOB-COMPLETE
    END-IF.

EMPL-CHECK.

    EXEC CICS CHECK ACTIVITY('NEWEMPL')
                COMPSTATUS(WS-STATUS)
                ABCODE(ABEND-CODE)

    END-EXEC.

    IF WS-STATUS NOT = DFHVALUE(NORMAL) THEN
        PERFORM EMPL-NOT-COMPLETE
    ELSE
        PERFORM EMPL-COMPLETE
    END-IF.

EMPL-JOB-CONNECT.

    EXEC CICS DEFINE ACTIVITY('NEWEMJD')
                    TRANSID('EMJD')
                    PROGRAM('EMPLJOBE')
                    RESP(RES-AREA)
                    RESP2(RES2-AREA)

    END-EXEC.

    EXEC CICS RUN ACTIVITY('NEWEMJD')
                SYNCHRONOUS
                RESP(RES-AREA)
                RESP2(RES2-AREA)

    END-EXEC.

    EXEC CICS CHECK ACTIVITY('NEWEMJD')
                COMPSTATUS(WS-STATUS)
                ABCODE(ABEND-CODE)

    END-EXEC.

    IF WS-STATUS NOT = DFHVALUE(NORMAL) THEN
        PERFORM EMPLJOB-NOT-COMPLETE
    ELSE
        MOVE EMPLJOB-OKAY-MESSAGE TO TRACE-MSG
        PERFORM DISPLAY-TRACE-MESSAGE
        PERFORM EMPLJOB-COMPLETE
    END-IF.

```

Figure 83 (Part 5 of 6). Pseudo-Code for the EMPLROOT Program

```

JOB-NOT-COMPLETE.
  PERFORM GET-ROOT-CONTAINER.
  MOVE 'N' TO JOB-OKAY.
  PERFORM PUT-ROOT-CONTAINER.
JOB-COMPLETE.
  PERFORM GET-ROOT-CONTAINER.
  MOVE 'Y' TO JOB-OKAY.
  PERFORM PUT-ROOT-CONTAINER.
EMPL-NOT-COMPLETE.
  PERFORM GET-ROOT-CONTAINER.
  MOVE 'N' TO EMPL-OKAY.
  PERFORM PUT-ROOT-CONTAINER.
EMPL-COMPLETE.
  PERFORM GET-ROOT-CONTAINER.
  MOVE 'Y' TO EMPL-OKAY.
  PERFORM PUT-ROOT-CONTAINER.
EMPLJOB-NOT-COMPLETE.
  PERFORM GET-ROOT-CONTAINER.
  MOVE 'N' TO EMPL-JOB-OKAY.
  PERFORM PUT-ROOT-CONTAINER.
EMPLJOB-COMPLETE.
  PERFORM GET-ROOT-CONTAINER.
  MOVE 'Y' TO EMPL-JOB-OKAY.
  PERFORM PUT-ROOT-CONTAINER.

GET-ROOT-CONTAINER.
  EXEC CICS GET CONTAINER('ROOT-CONTAINER')
    INTO(ROOT-CONTAINER)
  END-EXEC.

PUT-ROOT-CONTAINER.
  EXEC CICS PUT CONTAINER('ROOT-CONTAINER')
    FROM(ROOT-CONTAINER)
  END-EXEC.
End Program.

```

Figure 83 (Part 6 of 6). Pseudo-Code for the EMPLROOT Program

### 7.2.3 Pseudo-Code of the Activity NEWEMPL

The employee activity program EMPLACTM is designed to be reattached by CICS BTS when events in which it is interested are triggered. EMPLACTM requests CICS BTS to run asynchronously activities EMPLTASK and EMOLROLE. When they are completed normally, it requests to run activity EMPLCOSC.

Figure 84 on page 157 shows, in COBOL pseudo-code, the employee activity program, EMPLACTM.

We put the code that deals with the definition of the activities and events in parts two and three of the figure, which relate to Figure 75 on page 139.

We put the code that deals with the results of the execution of the activities (EMPLROLE and EMPLTASK) in part four of the figure, which relates to Figure 77 on page 141. The code that deals with the connection between employee and competency scheme relates to Figure 79 on page 143.

```

IDENTIFICATION DIVISION.
PROGRAM-ID.  EMPLACTM.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
*
05 WS-STATUS                PIC S9(8) comp.
05 EVENT-TYPE              PIC S9(8) comp.
05 RESP-AREA              PIC S9(8) comp.
05 RESP2-AREA            PIC S9(8) comp.
05 ABEND-CODE             PIC X(4).
05 PROCESS-NAME.
   10 PROCESS-ID          PIC X(4) VALUE 'EMPL'.
   10 TASK-NUMBER        PIC 9(7).
   10 FILLER              PIC X(25) VALUE SPACES.
*
*   EVENTS
*
05 EVENT-NAME              PIC x(16).
   88 DFH-INITIAL         value 'DFHINITIAL'.
   88 EMPL-REL-COMPLETE   value 'EMPLREL-COMPLETE'.
   88 EMPL-COSC-EVENT     value 'EMPL-COSC-EVENT'.

05 SUB-EVENT-TYPE         PIC x(16).
   88 EMPL-ROLE-EVENT    value 'EMPL-ROLE-EVENT'.
   88 EMPL-TASK-EVENT    value 'EMPL-TASK-EVENT'.
*
*   CONTAINERS
*
COPY EMPLCONT.

*
*   ERROR AND TRACE MESSAGES
*
05 COMMON-MSG-AREA.
   10 COMMON-MSG         PIC X(40).
05 COMMON-MSG-LEN        PIC S9(4) VALUE +44 COMP.

05 INVALID-EVENT.
   10 IE-MESSAGE         PIC X(40) VALUE
      'INVALID EVENT RECEIVED          '.

05 BAD-RETRIEVE-MESSAGE.
   10 BAD-RETRIEVE-MSG   PIC X(40) VALUE
      'UNKNOWN EVENT FORM RETRIEVE IN EMPLACTM '.

05 EYE-CATCHER-END       PIC x(32) Value
      'EMPLACTM working storage ends'.

LINKAGE SECTION.

```

Figure 84 (Part 1 of 5). Pseudo-Code for the EMPLACTM Program

```

PROCEDURE DIVISION.
BEGIN-PROCESS.

    EXEC CICS RETRIEVE REATTACH EVENT(EVENT-NAME)
        RESP(RESP-AREA) RESP2(RESP2-AREA)
    END-EXEC.
    IF RESP-AREA NOT = DFHRESP(NORMAL) THEN
        MOVE BAD-RETRIEVE-MESSAGE TO COMMON-MSG-AREA
        PERFORM ABEND-PROCESS
    END-IF.

    EVALUATE TRUE
        WHEN DFH-INITIAL
            PERFORM INITIAL-ACTIVITY
        WHEN EMPL-REL-COMPLETE
            PERFORM EMPL-REL-CHECK
        WHEN EMPL-COSC-EVENT
            PERFORM EMPL-COSC-CHECK
        WHEN OTHER
            MOVE INVALID-EVENT TO COMMON-MSG-AREA
            PERFORM ABEND-PROCESS
    END-EVALUATE.

    IF TASK-OKAY = 'Y' AND
        ROLE-OKAY = 'Y' THEN
        PERFORM EMPL-CS
        EXEC CICS RETURN END-EXEC
    ELSE
        IF COSC-OKAY = 'Y'
            EXEC CICS DELETE CONTAINER('EMPL-LEVEL1-CON')
            END-EXEC
            EXEC CICS RETURN ENDACTIVITY END-EXEC
        ELSE
            EXEC CICS RETURN END-EXEC
        END-IF
    END-IF.

INITIAL-ACTIVITY.
* GET PROCESS NAME AND PROCESS CONTAINER
    EXEC CICS ASSIGN PROCESS(PROCESS-NAME)
        RESP(RESP-AREA)
        RESP2(RESP2-AREA)
    END-EXEC.

    EXEC CICS GET CONTAINER(PROCESS-NAME)
        INTO(PROCESS-CONTAINER)
        FLENGTH(PC-LENGTH)
        PROCESS
    END-EXEC.

    PERFORM PUT-EMPL-LEVEL-1-CONTAINER.

    EXEC CICS DEFINE ACTIVITY('EMPLROLE')
        TRANSID('EMPO')
        PROGRAM('EMPLACTR')
        EVENT('EMPL-ROLE-EVENT')
        RESP(RESP-AREA)
        RESP2(RESP2-AREA)
    END-EXEC.

    EXEC CICS DEFINE ACTIVITY('EMPLTASK')
        TRANSID('EMPT')
        PROGRAM('EMPLACTT')
        EVENT('EMPL-TASK-EVENT')
        RESP(RESP-AREA)
        RESP2(RESP2-AREA)
    END-EXEC.

```

Figure 84 (Part 2 of 5). Pseudo-Code for the EMPLACTM Program



```

EXEC CICS DEFINE COMPOSITE EVENT('EMPLREL-COMPLETE') AND
      RESP(RES-AREA)
      RESP2(RES-AREA)
END-EXEC.

EXEC CICS ADD SUBEVENT('EMPL-ROLE-EVENT')
      EVENT('EMPLREL-COMPLETE')
      RESP(RES-AREA)
      RESP2(RES-AREA)
END-EXEC.

EXEC CICS ADD SUBEVENT('EMPL-TASK-EVENT')
      EVENT('EMPLREL-COMPLETE')
      RESP(RES-AREA)
      RESP2(RES-AREA)
END-EXEC.

EXEC CICS RUN ACTIVITY('EMPLROLE')
      ASYNCHRONOUS
      RESP(RES-AREA)
      RESP2(RES-AREA)
END-EXEC.

EXEC CICS RUN ACTIVITY('EMPLTASK')
      ASYNCHRONOUS
      RESP(RES-AREA)
      RESP2(RES-AREA)
END-EXEC.

EMPL-REL-CHECK.
* CHECK SUB EVENTS FOR THE COMPOSITE EVENT

EXEC CICS RETRIEVE SUBEVENT(SUB-EVENT-TYPE)
      EVENT(EVENT-NAME)
      EVENTTYPE(EVENT-TYPE)
      RESP(RES-AREA)
      RESP2(RES-AREA)
END-EXEC.

IF RESP-AREA NOT = DFHRESP(NORMAL)
  MOVE INVALID-EVENT TO COMMON-MSG-AREA
  PERFORM ABEND-PROCESS
END-IF.

EVALUATE TRUE
  WHEN EMPL-ROLE-EVENT
    PERFORM EMPL-ROLE-CHECK
  WHEN EMPL-TASK-EVENT
    PERFORM EMPL-TASK-CHECK
  WHEN OTHER
* UNEXPECTED SUBEVENT
  MOVE INVALID-EVENT TO COMMON-MSG-AREA
  PERFORM ABEND-PROCESS
END-EVALUATE.

* ALL COMPOSITE SUB-EVENTS OKAY DELETE COMPOSITE EVENT

PERFORM GET-EMPL-LEVEL-1-CONTAINER.

IF TASK-OKAY = 'Y' AND
  ROLE-OKAY = 'Y' THEN
  EXEC CICS DELETE EVENT('EMPLREL-COMPLETE')
  END-EXEC
ELSE
*
END-IF.

```

Figure 84 (Part 3 of 5). Pseudo-Code for the EMPLACTM Program

```

EMPL-ROLE-CHECK.

EXEC CICS CHECK ACTIVITY('EMPLROLE')
                COMPSTATUS(WS-STATUS)
                ABCODE(ABEND-CODE)

END-EXEC.

IF WS-STATUS NOT = DFHVALUE(NORMAL) THEN
  PERFORM ROLE-NOT-COMPLETE
ELSE
  PERFORM ROLE-COMPLETE
END-IF.

EMPL-TASK-CHECK.

EXEC CICS CHECK ACTIVITY('EMPTASK')
                COMPSTATUS(WS-STATUS)
                ABCODE(ABEND-CODE)

END-EXEC.

IF WS-STATUS NOT = DFHVALUE(NORMAL) THEN
  PERFORM TASK-NOT-COMPLETE
ELSE
  PERFORM TASK-COMPLETE
END-IF.

EMPL-CS.

EXEC CICS DEFINE ACTIVITY('EMPLCOSC')
                TRANSID('EMPS')
                PROGRAM('EMPLACTS')
                EVENT('EMPL-COSC-EVENT')
                RESP(RES-AREA)
                RESP2(RES2-AREA)

END-EXEC.

EXEC CICS RUN ACTIVITY('EMPLCOSC')
            ASYNCHRONOUS
            RESP(RES-AREA)
            RESP2(RES2-AREA)

END-EXEC.

EMPL-COSC-CHECK.

EXEC CICS CHECK ACTIVITY('EMPLCOSC')
                COMPSTATUS(WS-STATUS)
                ABCODE(ABEND-CODE)

END-EXEC.

IF WS-STATUS NOT = DFHVALUE(NORMAL) THEN
  PERFORM COSC-NOT-COMPLETE
ELSE
  PERFORM COSC-COMPLETE
END-IF.

```

Figure 84 (Part 4 of 5). Pseudo-Code for the EMPLACTM Program

```

ROLE-NOT-COMPLETE.
    PERFORM GET-EMPL-LEVEL-1-CONTAINER.
    MOVE 'N' TO ROLE-OKAY.
    PERFORM PUT-EMPL-LEVEL-1-CONTAINER.
ROLE-COMPLETE.
    PERFORM GET-EMPL-LEVEL-1-CONTAINER.
    MOVE 'Y' TO ROLE-OKAY.
    PERFORM PUT-EMPL-LEVEL-1-CONTAINER.
TASK-NOT-COMPLETE.
    PERFORM GET-EMPL-LEVEL-1-CONTAINER.
    MOVE 'N' TO TASK-OKAY.
    PERFORM PUT-EMPL-LEVEL-1-CONTAINER.
TASK-COMPLETE.
    PERFORM GET-EMPL-LEVEL-1-CONTAINER.
    MOVE 'Y' TO TASK-OKAY.
    PERFORM PUT-EMPL-LEVEL-1-CONTAINER.
COSC-NOT-COMPLETE.
    PERFORM GET-EMPL-LEVEL-1-CONTAINER.
    MOVE 'N' TO COSC-OKAY.
    PERFORM PUT-EMPL-LEVEL-1-CONTAINER.
COSC-COMPLETE.
    PERFORM GET-EMPL-LEVEL-1-CONTAINER.
    MOVE 'Y' TO COSC-OKAY.
    PERFORM PUT-EMPL-LEVEL-1-CONTAINER.
GET-EMPL-LEVEL-1-CONTAINER.
    EXEC CICS GET CONTAINER('EMPL-LEVEL1-CON')
        INTO(EMPL-LEVEL-1-CONTAINER)

    END-EXEC.
PUT-EMPL-LEVEL-1-CONTAINER.
    EXEC CICS PUT CONTAINER('EMPL-LEVEL1-CON')
        FROM(EMPL-LEVEL-1-CONTAINER)

    END-EXEC.

ABEND-PROCESS.
    EXEC CICS WRITEQ TD
        QUEUE('CSMT')
        FROM(COMMON-MSG-AREA)
        LENGTH(COMMON-MSG-LEN)

    END-EXEC.

    EXEC CICS ABEND ABCODE('U001') NODUMP
    END-EXEC.

```

End Program.

Figure 84 (Part 5 of 5). Pseudo-Code for the EMPLACTM Program

## 7.2.4 Pseudo-Code of the Activity NEWJD

The employee activity program JOBEACTM is designed to be reattached by CICS BTS when events in which it is interested are triggered. JOBEACTM requests CICS BTS to run activities JOBETASK and JOBEORGU. When they are completed normally, it requests to run activities JOBECOSC and JOBEROLE.

Figure 85 on page 162 shows, in COBOL pseudo-code, the Job activity program, JOBEACTM.

We put the code that deals with the definition of the activities and events in the part three of the figure that relates to Figure 74 on page 138.

We put the code that deals with the results of the execution of the activities (JOBEORGU and JOBETASK) in part four of the figure that relates to Figure 76 on page 140.

We put the code that deals with the definitions of the activities (JOBBEROLE and JOBECOSC) in part five of the figure that relates to Figure 76 on page 140. The code that deals with the results of the execution of the activities relates to Figure 78 on page 142.

```

IDENTIFICATION DIVISION.
PROGRAM-ID. JOBEACTM.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
*
05 WS-STATUS PIC S9(8) comp.
05 EVENT-TYPE PIC S9(8) comp.
05 RESP-AREA PIC S9(8) comp.
05 RESP2-AREA PIC S9(8) comp.
05 ABEND-CODE PIC X(4).
05 PROCESS-NAME.
10 PROCESS-ID PIC X(4) VALUE 'EMPL'.
10 TASK-NUMBER PIC 9(7).
10 FILLER PIC X(25) VALUE SPACES.
*
* EVENTS
*
05 EVENT-NAME PIC x(16).
88 DFH-INITIAL value 'DFHINITIAL'.
88 JOB-REL-COMPLETE value 'JD-REL-COMPLETE'.
88 JOB-ROLE-EVENT value 'JOB-ROLE-EVENT'.
88 JOB-COSC-EVENT value 'JOB-COSC-EVENT'.

05 SUB-EVENT-TYPE PIC x(16).
88 JOB-ORGU-EVENT value 'JOB-ORGU-EVENT'.
88 JOB-TASK-EVENT value 'JOB-TASK-EVENT'.
*
* CONTAINERS
*
COPY EMPLCONT.

*
* ERROR AND TRACE MESSAGES
*
05 COMMON-MSG-AREA.
10 COMMON-MSG PIC X(40).
05 COMMON-MSG-LEN PIC S9(4) VALUE +44 COMP.

05 INVALID-EVENT.
10 IE-MESSAGE PIC X(40) VALUE
'INVALID EVENT RECEIVED '.

05 BAD-RETRIEVE-MESSAGE.
10 BAD-RETRIEVE-MSG PIC X(40) VALUE
'UNKNOWN EVENT FORM RETRIEVE IN jobeactm '.

LINKAGE SECTION.

```

Figure 85 (Part 1 of 6). Pseudo-Code for the JOBEACTM Program

```

PROCEDURE DIVISION.

BEGIN-PROCESS.

    EXEC CICS RETRIEVE REATTACH EVENT(EVENT-NAME)
        RESP(RES-AREA) RESP2(RES2-AREA)
    END-EXEC.
    IF RESP-AREA NOT = DFHRESP(NORMAL) THEN
        MOVE BAD-RETRIEVE-MESSAGE TO COMMON-MSG-AREA
        PERFORM ABEND-PROCESS
    END-IF.

    EVALUATE TRUE
        WHEN DFH-INITIAL
            PERFORM INITIAL-ACTIVITY
        WHEN JOB-REL-COMplete
            PERFORM JOB-REL-CHECK
        WHEN JOB-COSC-EVENT
            PERFORM JOB-COSC-CHECK
        WHEN JOB-ROLE-EVENT
            PERFORM JOB-ROLE-CHECK
        WHEN OTHER
            MOVE INVALID-EVENT TO COMMON-MSG-AREA
            PERFORM ABEND-PROCESS
    END-EVALUATE.

    IF JTASK-OKAY = 'Y' AND
        ORGU-OKAY = 'Y' THEN
        PERFORM JOB-ROLE-COSC
        EXEC CICS RETURN END-EXEC
    ELSE
        IF JROLE-OKAY = 'Y' AND
            JCOSC-OKAY = 'Y' THEN
            EXEC CICS DELETE CONTAINER('JOB-LEVEL1-CON')
            END-EXEC
            EXEC CICS RETURN ENDACTIVITY END-EXEC
        ELSE
            EXEC CICS RETURN END-EXEC
        END-IF
    END-IF.

```

Figure 85 (Part 2 of 6). Pseudo-Code for the JOBEACTM Program

```

INITIAL-ACTIVITY.
* GET PROCESS NAME AND PROCESS CONTAINER
  EXEC CICS ASSIGN PROCESS(PROCESS-NAME)
                RESP(RESP-AREA)
                RESP2(RESP2-AREA)
  END-EXEC.

EXEC CICS GET CONTAINER(PROCESS-NAME)
          INTO(PROCESS-CONTAINER)
          FLENGTH(PC-LENGTH)
          PROCESS
  END-EXEC.

PERFORM PUT-JOB-LEVEL-1-CONTAINER.

EXEC CICS DEFINE ACTIVITY('JOBEORGU')
                TRANSID('JOBU')
                PROGRAM('JOBEACTU')
                EVENT('JOB-ORGU-EVENT')
                RESP(RESP-AREA)
                RESP2(RESP2-AREA)
  END-EXEC.

EXEC CICS DEFINE ACTIVITY('JOBETASK')
                TRANSID('JOBT')
                PROGRAM('JOBEACTT')
                EVENT('JOB-TASK-EVENT')
                RESP(RESP-AREA)
                RESP2(RESP2-AREA)
  END-EXEC.

EXEC CICS DEFINE COMPOSITE EVENT('JD-REL-COMPLETE') AND
                RESP(RESP-AREA)
                RESP2(RESP2-AREA)
  END-EXEC.

  EXEC CICS ADD SUBEVENT('JOB-ORGU-EVENT')
                EVENT('JD-REL-COMPLETE')
                RESP(RESP-AREA)
                RESP2(RESP2-AREA)
  END-EXEC.

  EXEC CICS ADD SUBEVENT('JOB-TASK-EVENT')
                EVENT('JD-REL-COMPLETE')
                RESP(RESP-AREA)
                RESP2(RESP2-AREA)
  END-EXEC.

EXEC CICS RUN ACTIVITY('JOBEORGU')
                ASYNCHRONOUS
                RESP(RESP-AREA)
                RESP2(RESP2-AREA)
  END-EXEC.

  EXEC CICS RUN ACTIVITY('JOBETASK')
                ASYNCHRONOUS
                RESP(RESP-AREA)
                RESP2(RESP2-AREA)
  END-EXEC.

```

Figure 85 (Part 3 of 6). Pseudo-Code for the JOBEACTM Program

```

JOB-REL-CHECK.
* CHECK SUB EVENTS FOR THE COMPOSITE EVENT

    EXEC CICS RETRIEVE SUBEVENT(SUB-EVENT-TYPE)
                EVENT(EVENT-NAME)
                EVENTTYPE(EVENT-TYPE)
                RESP(RESP-AREA)
                RESP2(RESP2-AREA)

    END-EXEC.

    IF RESP-AREA NOT = DFHRESP(NORMAL)
        MOVE INVALID-EVENT TO COMMON-MSG-AREA
        PERFORM ABEND-PROCESS
    END-IF.

    EVALUATE TRUE
        WHEN JOB-ORGU-EVENT
            PERFORM JOB-ORGU-CHECK
        WHEN JOB-TASK-EVENT
            PERFORM JOB-TASK-CHECK
        WHEN OTHER
* UNEXPECTED SUBEVENT
        MOVE INVALID-EVENT TO COMMON-MSG-AREA
        PERFORM ABEND-PROCESS
    END-EVALUATE.

* ALL COMPOSITE SUB-EVENTS OKAY DELETE COMPOSITE EVENT

    PERFORM GET-JOB-LEVEL-1-CONTAINER.

    IF JTASK-OKAY = 'Y' AND
        ORGU-OKAY = 'Y' THEN
        EXEC CICS DELETE EVENT('JD-REL-COMPLETE')
        END-EXEC
    ELSE
*
    END-IF.

JOB-ORGU-CHECK.

    EXEC CICS CHECK ACTIVITY('JOBEORGU')
                COMPSTATUS(WS-STATUS)
                ABCODE(ABEND-CODE)

    END-EXEC.

    IF WS-STATUS NOT = DFHVALUE(NORMAL) THEN
        PERFORM ORGU-NOT-COMPLETE
    ELSE
        PERFORM ORGU-COMPLETE
    END-IF.

JOB-TASK-CHECK.

    EXEC CICS CHECK ACTIVITY('JOBETASK')
                COMPSTATUS(WS-STATUS)
                ABCODE(ABEND-CODE)

    END-EXEC.

    IF WS-STATUS NOT = DFHVALUE(NORMAL) THEN
        PERFORM TASK-NOT-COMPLETE
    ELSE
        PERFORM TASK-COMPLETE
    END-IF.

```

Figure 85 (Part 4 of 6). Pseudo-Code for the JOBEACTM Program

```

JOB-ROLE-COSC.

EXEC CICS DEFINE ACTIVITY('JOBROLE')
      TRANSID('JOB0')
      PROGRAM('JOBACTR')
      EVENT('JOB-ROLE-EVENT')
      RESP(RESP-AREA)
      RESP2(RESP2-AREA)

END-EXEC.

EXEC CICS DEFINE ACTIVITY('JOBECOSC')
      TRANSID('JOBS')
      PROGRAM('JOBACTS')
      EVENT('JOB-COSC-EVENT')
      RESP(RESP-AREA)
      RESP2(RESP2-AREA)

END-EXEC.

EXEC CICS RUN ACTIVITY('JOBROLE')
      ASYNCHRONOUS
      RESP(RESP-AREA)
      RESP2(RESP2-AREA)

END-EXEC.

EXEC CICS RUN ACTIVITY('JOBECOSC')
      ASYNCHRONOUS
      RESP(RESP-AREA)
      RESP2(RESP2-AREA)

END-EXEC.

JOB-COSC-CHECK.

EXEC CICS CHECK ACTIVITY('JOBECOSC')
      COMPSTATUS(WS-STATUS)
      ABCODE(ABEND-CODE)

END-EXEC.

IF WS-STATUS NOT = DFHVALUE(NORMAL) THEN
  PERFORM COSC-NOT-COMPLETE
ELSE
  PERFORM COSC-COMPLETE
END-IF.

JOB-ROLE-CHECK.

EXEC CICS CHECK ACTIVITY('JOBROLE')
      COMPSTATUS(WS-STATUS)
      ABCODE(ABEND-CODE)

END-EXEC.

IF WS-STATUS NOT = DFHVALUE(NORMAL) THEN
  PERFORM ROLE-NOT-COMPLETE
ELSE
  PERFORM ROLE-COMPLETE
END-IF.

```

Figure 85 (Part 5 of 6). Pseudo-Code for the JOBEACTM Program



```

ORGU-NOT-COMPLETE.
  PERFORM GET-JOB-LEVEL-1-CONTAINER.
  MOVE 'N' TO ORGU-OKAY.
  PERFORM PUT-JOB-LEVEL-1-CONTAINER.
ORGU-COMPLETE.
  PERFORM GET-JOB-LEVEL-1-CONTAINER.
  MOVE 'Y' TO ORGU-OKAY.
  PERFORM PUT-JOB-LEVEL-1-CONTAINER.
TASK-NOT-COMPLETE.
  PERFORM GET-JOB-LEVEL-1-CONTAINER.
  MOVE 'N' TO JTASK-OKAY.
  PERFORM PUT-JOB-LEVEL-1-CONTAINER.
TASK-COMPLETE.
  PERFORM GET-JOB-LEVEL-1-CONTAINER.
  MOVE 'Y' TO JTASK-OKAY.
  PERFORM PUT-JOB-LEVEL-1-CONTAINER.
COSC-NOT-COMPLETE.
  PERFORM GET-JOB-LEVEL-1-CONTAINER.
  MOVE 'N' TO JCOSC-OKAY.
  PERFORM PUT-JOB-LEVEL-1-CONTAINER.
COSC-COMPLETE.
  PERFORM GET-JOB-LEVEL-1-CONTAINER.
  MOVE 'Y' TO JCOSC-OKAY.
  PERFORM PUT-JOB-LEVEL-1-CONTAINER.
ROLE-NOT-COMPLETE.
  PERFORM GET-JOB-LEVEL-1-CONTAINER.
  MOVE 'N' TO JROLE-OKAY.
  PERFORM PUT-JOB-LEVEL-1-CONTAINER.
ROLE-COMPLETE.
  PERFORM GET-JOB-LEVEL-1-CONTAINER.
  MOVE 'Y' TO JROLE-OKAY.
  PERFORM PUT-JOB-LEVEL-1-CONTAINER.

GET-JOB-LEVEL-1-CONTAINER.
  EXEC CICS GET CONTAINER('JOB-LEVEL1-CON')
           INTO(JOB-LEVEL-1-CONTAINER)
END-EXEC.
PUT-JOB-LEVEL-1-CONTAINER.
  EXEC CICS PUT CONTAINER('JOB-LEVEL1-CON')
           FROM(JOB-LEVEL-1-CONTAINER)
END-EXEC.

ABEND-PROCESS.
  EXEC CICS WRITEQ TD
           QUEUE('CSMT')
           FROM(COMMON-MSG-AREA)
           LENGTH(COMMON-MSG-LEN)
END-EXEC.

  EXEC CICS ABEND ABCODE('U001') NODUMP
END-EXEC.

```

End Program.

Figure 85 (Part 6 of 6). Pseudo-Code for the JOBEACTM Program

---

## 7.3 Hints and Tips

In this section, we share some of our experience in using CICS BTS in the course of this project.

### 7.3.1 Activities

These are our comments regarding activities in general:

- A program can acquire only one process per UOW. The process remains acquired until the next syncpoint. If you try to run the acquired activity the second time, you will get the following message:

```
EIBRESP    = 108 PROCESSERR  
EIBRESP2   = 14
```

If you try to acquire the process the second time, you will get the following message:

```
EIBRESP    = 16 INVREQ  
EIBRESP2   = 1
```

- We recommend to issue a RETURN ENDACTIVITY command at the end of the final activation of an activity to avoid an activity going dormant forever through a program logic error.

If the parent issues a RETURN ENDACTIVITY command with an outstanding activity completion event, the business transaction will abend. The child with the outstanding completion event will complete with FORCED status (see Figure 150 on page 261).

- Always issue a CHECK ACTIVITY command. This will delete the activity completion event.

### 7.3.2 Root Activity

These are our comments regarding root activities:

- The root activity should only contain business control logic. Its child activity contains the business logic. In other words, all the file or database interface logic is in child activities.
- The root activity does not have a completion event.

### 7.3.3 User Syncpoints

A program that is running as the activation of a BTS process or activity cannot issue user syncpoints (EXEC CICS SYNCPOINT, or, EXEC CICS SYNCPOINT ROLLBACK commands).

Only programs that are running outside any BTS process may issue user syncpoints, or the transaction will abend with an ASP9 abend code. Some examples of programs that may issue user syncpoints are:

- A top-level transaction that defines and runs a BTS process
- A program executing outside the BTS environment that acquires a process or activity by means of an ACQUIRE command.

### 7.3.4 Administration Add-Ons to Your Application

These are some comments regarding timer event management.

- If a piece of processing (for example, at midnight on June 30th, half-yearly customer statements are prepared) could result in a large number of timers being set to expire at the same time, we recommend to put the timers in groups and have different expiry times. This spreads the load on CICS and improves performance.
- If you shut down CICS at regular times, and know beforehand that at certain times it will be unavailable, try not to set a large number of timers to expire at these times. The timer events will all fire when CICS is restarted, which could affect CICS startup elapse time.
- Deleting a composite event explicitly by issuing a DELETE EVENT command does not delete the composite's sub-events.
- A suspended task is different from a dormant activity.

A suspended task, either after issuing the EXEC CICS WAIT or DELAY commands, still holds resources that it acquired before. The task with its unique task number is still under CICS task management.

When an activity is dormant, even if it has its own transaction code (with its task number allocated by CICS before), you cannot find the task (the activity) in CICS environment by issuing the CEMT INQUIRE TASK command.

### 7.3.5 A Useful Tool

While developing our applications, we found that we were left with several processes on our repository that we were unable to complete. These processes are referred to as *stuck processes*. Please see 12.1, "Application Design Errors" on page 215 for discussion about stuck processes and the ways to avoid them.

We decided to implement a routine to remove these stuck processes from the repository. We used another CICS BTS command, CANCEL ACQPROCESS, to help us achieve this goal.

The routine first asks the user which PROCESSTYPE to eliminate from the repository. It inquires on the process type to ensure that it exists, and then performs a series of ACQUIRE PROCESS and CANCEL ACQPROCESS commands until all processes of the requested process type have disappeared. Note that the program has to issue a user syncpoint, because only one process can be acquired in a UOW.

```
Send a map to find out which processtype to kill
Inquire processtype start
Inquire processtype next
loop until no more processes of processtype
  startbrowse process browsetoken processtype
  getnext process
  acquire process
  cancel acqprocess
  syncpoint
  endbrowse
  write to td queue - 'process xxx - cancelled'
end-loop
send a map saying all was great
```



---

## Chapter 8. Reuse of Elements

The word *reuse* is popular and important today. CICS BTS provides means to reuse existing CICS programs and transactions. Do not misunderstand reuse in terms of CICS BTS by comparing it with the object oriented approach of reuse. CICS BTS, for example, does not support inheritance. With CICS BTS, reuse has another meaning. That is, you can reuse the following parts of your existing applications:

- 3270 transactions
- Non-terminal transactions
- Programs
- CICS BTS elements

---

### 8.1 Reuse of Existing Applications

CICS is installed at many sites, and the existing CICS API is used to run millions of programs. Vendors offer their applications using CICS as a general solution or even as a standard.

Now, CICS offers a set of additional commands that extend the existing API dramatically. The new API implements a new programming paradigm - Event programming. But nobody who runs a CICS environment wants to start over from the beginning. Whenever possible, you want to continue using your existing programs, but, on the other hand, it is advisable to begin using the new CICS BTS as soon as possible. Therefore, an integration of existing CICS programs and transactions is a must.

#### 8.1.1 Reuse of 3270 Transactions

A CICS BTS application can use the 3270 bridge function. CICS BTS applications can be integrated with, and make use of, existing 3270-based transactions.

Even though CICS BTS activities are not terminal related (they are never started directly from a terminal), a CICS BTS activity can be defined and executed by a 3270-based transaction. Additionally, the bridge exit program can be used to put a *CICS BTS wrapper* around the original 3270 transaction.

It is possible to execute both conversational and pseudo-conversational 3270 based programs using the bridge mechanism. Refer to *CICS Business Transaction Services, SC34-5268* and to *CICS Internet Guide, SC34-5445* for more information.

#### 8.1.2 Reuse of Non-Terminal Transactions and Programs

As far as it is useful, CICS BTS supports the classic CICS API also. Syncpoint processing or BMS commands, for example, are not supported. Program Control commands (for example, LINK, XCTL) are supported. You can reuse existing CICS transactions and programs in your new CICS BTS applications.

---

## 8.2 Reuse of CICS BTS Elements

The reuse of CICS BTS elements depends primarily on your application design. Using a business process model, as a result of systems analysis on the enterprise level, and using scopes to analyze in depth and developing a process entity model, you will have a collection of activities and CICS BTS processes that you can reuse whenever possible.

This approach leads you to the issue of reuse management.

---

## 8.3 Reuse Management

When you start to build your business using CICS BTS business transactions and develop your activities as a result of business analysis, you will need some kind of reuse management.

We can compare reuse management of CICS BTS processes and activities with data management using a data repository. The same approach is valid for reuse management of CICS BTS elements. A CICS BTS element repository can be developed for that purpose.

### 8.3.1 Process

A process is an encapsulated unit. Using the CICS BTS API, you can start a process from any program you use. When your processes are designed to work on the smallest entities of your business model, you can build the next level of complexity by combining processes within a process.

### 8.3.2 Activities

CICS BTS starts each activity the first time using the reattachment event DFHINITIAL. No data is provided to the program; therefore, the only way to communicate is to use other CICS resources to pass data to the program. When using CICS BTS activities, the preferred way is to get and put containers.

When you want to reuse your activities, it is best to develop a concept of how these containers are used. Each activity should use a container in the same way. An example may be to use two containers, a *structure* container and a *data* container. By adding a field definition to the CICS BTS element repository, you will be able to generate the data interfaces to use these containers.

Additionally, it may be useful to define naming conventions with these containers.

Such a concept guarantees that each activity can communicate with another activity simply by looking in the structure container to help un-pick the data in the data container.

On the other hand, it is important to store enough information in the CICS BTS element repository to ensure that you will be able to find all activities when looking for an already developed business element. Therefore, it is useful to describe each activity in your repository based on the results of your business process analysis. Using keywords to describe the activities eases recovery. When you describe additionally, with each activity, which fields are used, you can more easily find the activities you need when analyzing the communication model of your business elements. A business process communication model is the description of the data

flow needed to run the different elements of your business process. To add a new employee, for example, to an organizational unit, you have to provide the employee's key data, at least, the employee number or another unique identifier.

You can now start thinking about other advantages the reuse repository is able to provide. For example, you can use it as:

- A field repository
- An interface generation and documentation tool
- A generator for program shells to call the functions packaged in modules

---

## 8.4 Mapping Messages to Context

If *reuse* is an important word in your company, you will face the problem of message mapping.

CICS BTS activities can easily be seen as candidates for reusable elements. It is essential that each element provides return information in terms of reporting errors that occur.

When interfacing with an end user, you usually write error messages that are interpreted by the end user. The end user's interpretation depends on the context of the application. In this case, the end user acts as part of the control logic and stops his activity when the error message necessitates such action.

To make this concept more clear, we use this example:

An activity reads basic customer data and receives the following response:  
Customer data is not available.

The information provided by an activity has to be interpreted in context of the usage.

- Used in a business process where no actions are executed following the request for this information, the response means that there is no data stored in the database concerning this customer.
- In a business process that adds contract data for a credit to the database, the same message is very important. When the customer cannot be read from your database, you cannot add a credit.

We recommend the use of a mapping algorithm for messages. This algorithm uses a message ID that has to be unique. Additionally, it is helpful if this ID can be used to identify the program that caused the message.

In context with the process the activity is used with, the algorithm provides:

- A unique message ID relating to the process.
- Information regarding the action required by the message.
- Replacement of variables.

To achieve this goal, it might be helpful to develop a single CICS BTS-wide valid message mapping program. This program uses a table to map the messages when they occur. Keep in mind that a process does not necessarily finish

processing when an error occurs; therefore, your program should be prepared to handle several messages.

We suggest transforming the messages using a translation table.

Table 9 shows the table that can be used to translate program messages.

<i>Table 9. Message Mapping Table</i>	
<b>Column name</b>	<b>Description</b>
Program name	Program that caused the message.
Input message ID	Message identification provided by the program that detected the error.
Processtype	Name of the processtype.
Process	Generic name of the process which provides the message. This should only be a generic name of a process. You can use, for example, the first 10 characters to identify the process. The rest has to be dynamic because CICS BTS needs a unique process name.
Message class	A number to react on, for example: <ul style="list-style-type: none"> <li>• 4 - minor error, process continues</li> <li>• 8 - important error, process continues</li> <li>• 12 - important error, process stops</li> <li>• 16 - major error, process stops</li> </ul>
Message number	Unique process dependent message ID
MessageText	Original text of the message

This table can be used by each program. It translates the program message into a context dependent process message and gives information as to whether the process can continue or has to stop.

When you use processes whose results have to be presented to your end users, we recommend that you use a process container to collect all messages. The root activity can sort all messages by message class and time they were generated to ensure that the most important message is the first.

You can add a message translation routine and a message communication routine to each activity. The communication routine stores all messages that were generated during the current activation in a container (probably, a special message container). The communication routine in the parent activity reads all containers and stores the messages in its message container. Finally, the root activity writes the messages to the process container.

Another way to provide these messages is to store them in a TS queue.



---

## Chapter 9. Programming Paradigms

In this chapter, we describe the application design when using CICS BTS and compare several techniques in the pre-CICS BTS and CICS BTS era.

Application design is a broad topic to discuss. Here, we only concentrate on the following topics relevant to CICS BTS.

1. Traditional transaction model and business transaction model
2. Sharing data in and across transactions
  - TWA, COMMAREA, CWA, TCTUA and Containers in CICS BTS
3. Synchronization among transactions
  - RETRIEVE WAIT/START, DELAY/CANCEL REQID, POST/CANCEL REQID and SYNCHRONOUS/ASYNCHRONOUS, EVENT in CICS BTS
4. Recovery considerations
  - Compensation implementation in CICS BTS
5. Dealing with exception conditions
  - CHECK API in CICS BTS
6. Access to system information
  - PROCESSTYPE and INQUIRE TASK related to CICS BTS
7. Program Structure

---

### 9.1 Traditional Transaction Model and Business Transaction Model

Before CICS TS 1.3, the largest transaction processing unit that CICS understood was terminal-related conversation or pseudo-conversation.

#### 9.1.1 Conversational and Pseudo-Conversational

Before CICS BTS, applications were mainly designed and developed around the the interface with a terminal, such as 3270 terminal, Automatic Teller Machine (ATM), or Point Of Sale (POS) terminal. The basic building blocks used by CICS applications are the CICS transaction and the UOW. Usually, the choice is between conversational and pseudo-conversational design. The contention and exclusive use of resources are usually the factors to determine the solution. Though conversational approach incurs less processor overhead, and CICS will handle any recovery and integrity implications, pseudo-conversational design will mostly be adopted to develop the applications with as little resource constraint as possible. Usually, a transaction would contain one UOW, but if a user program issues SYNCPOINT commands, there might be several UOWs in a transaction. Typically, a UOW is short-lived and it does not hold locks for long periods of time.

Figure 86 on page 176 shows the program structure in the conversational design.

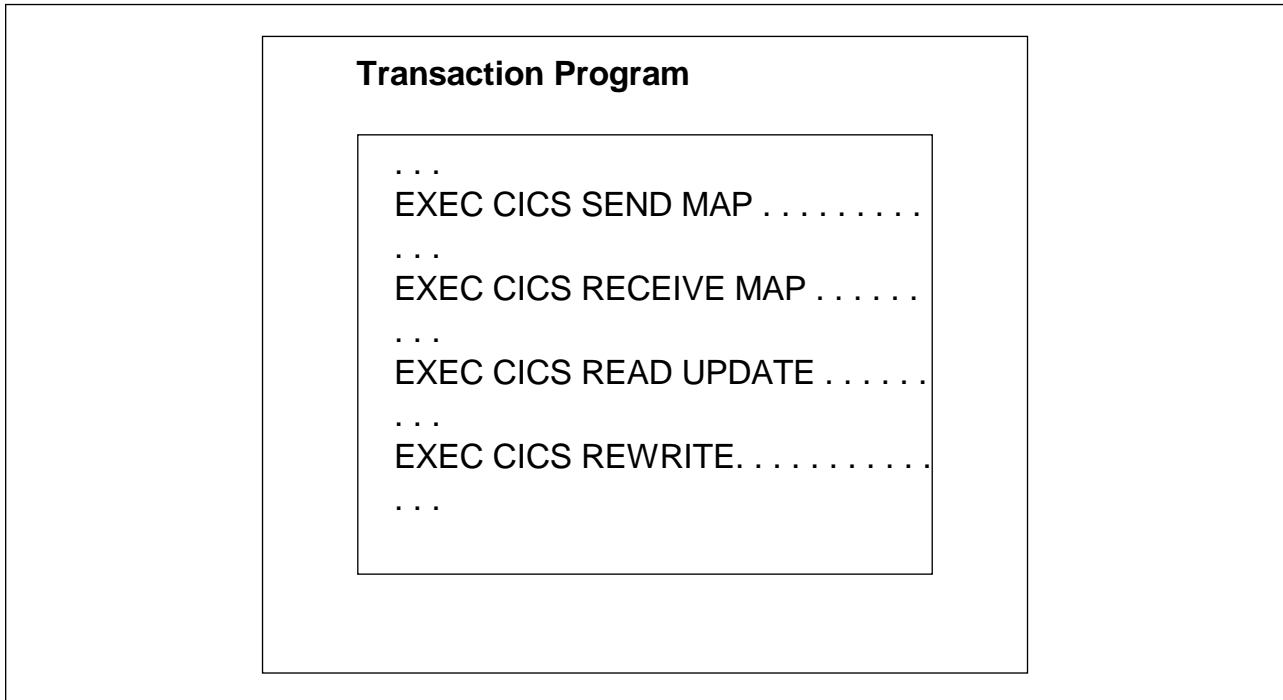


Figure 86. Program Structure in a Single-Program Transaction

In a typical conversational design, we see the EXEC CICS statements in the following sequence of execution in a program:

- SEND MAP is to send a map with some initial data to prompt and guide the user to enter the input data.
- RECEIVE MAP is to receive the map with user's input into the symbolic structure storage generated by Basic Mapping Support (BMS).
- READ UPDATE is to obtain a record from an external file.
- REWRITE is to update the record with its latest information.
- RETURN is to return to a CICS program at a higher level or to CICS.

Figure 87 on page 177 shows the program structure in the pseudo-conversational design.

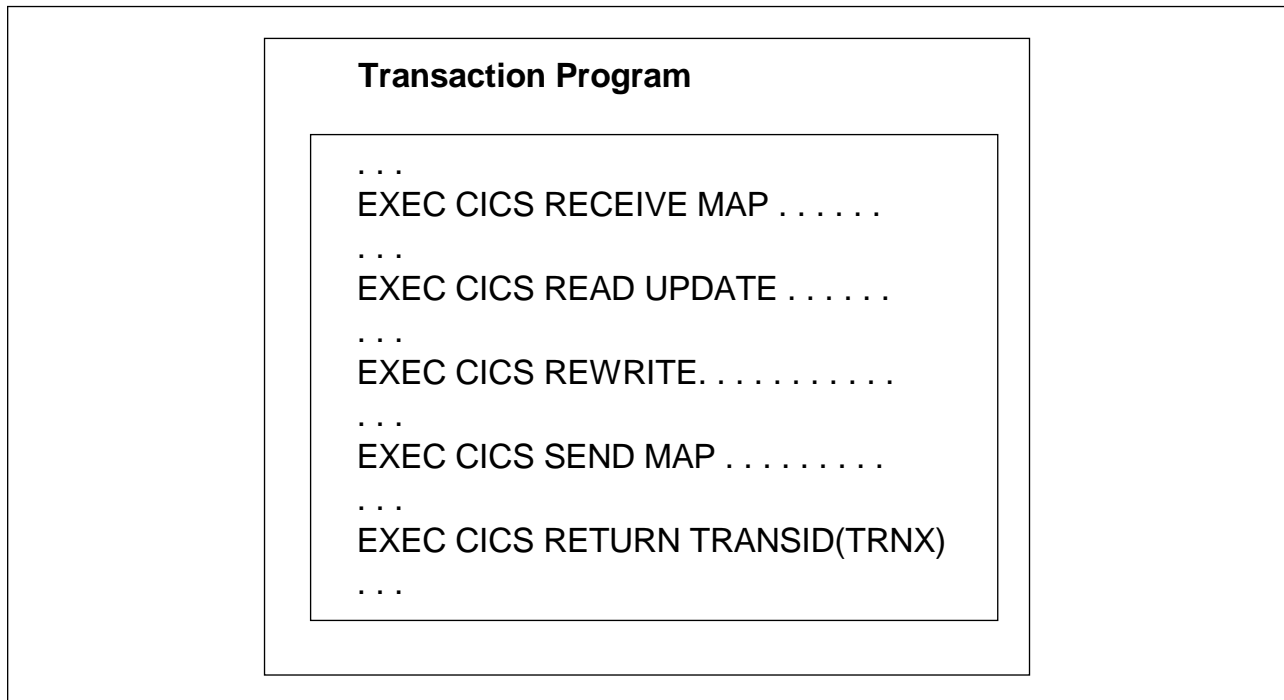


Figure 87. Program Structure in a Pseudo-conversational Transaction

In a typical pseudo-conversational design, we see the EXEC CICS statements in the following sequence of execution in a program:

- RECEIVE MAP is to receive the map with user's input into the symbolic structure storage generated by BMS.
- READ UPDATE is to obtain a record from an external file.
- REWRITE is to update the record with its latest information.
- SEND MAP is to send the map to confirm the user's latest input either in a confirmation message in the same map (screen) or a higher level menu.
- RETURN TRANSID is to specify the next transaction to be initiated when the user presses a control key which is usually an Enter key.

### 9.1.2 Multi-Tier Model

As the number of CICS applications grows in your company, the number of user interfaces with different terminal devices grows too. In order to reuse the business logic as much as possible, a better programming technique is to use a two-tier or even multi-tier programming model, that is, to separate the Front End processing that contains the presentation logic (dealing with external interface, such as terminal, MQ, Web input) from the business logic.

Figure 88 on page 178 shows a two-tier model of isolating business logic.

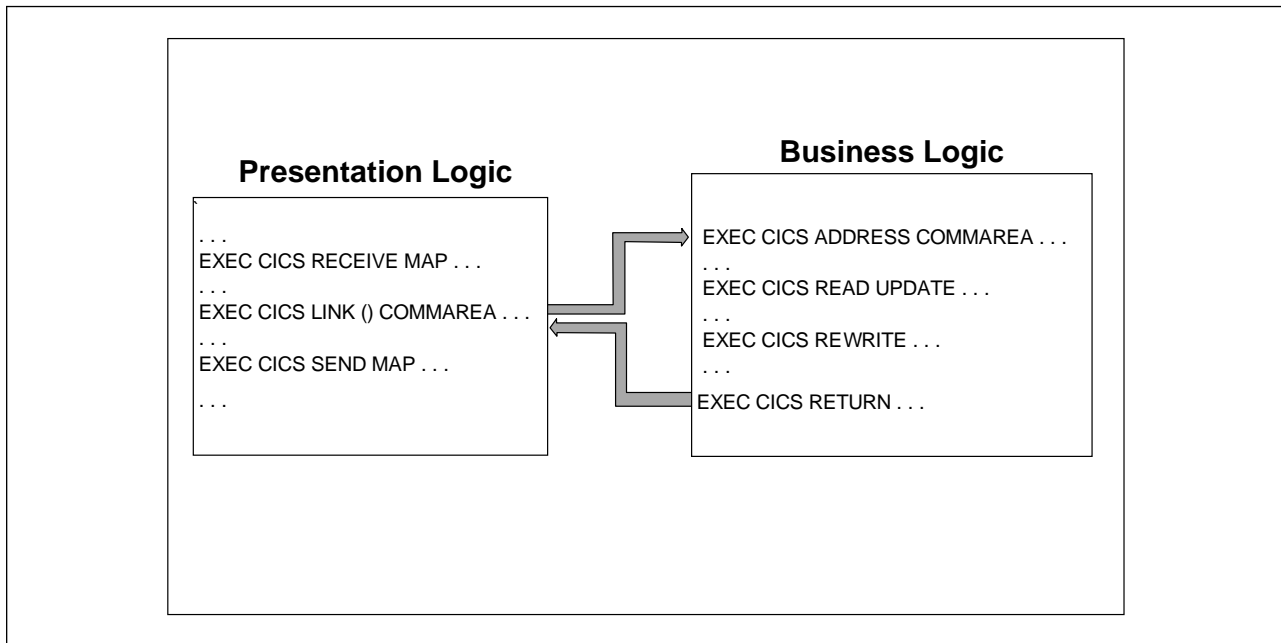


Figure 88. Two Tier Model with Separation of the Presentation Logic from the Business Logic

In a typical two-tier design, we see two sets of EXEC CICS statements in two separate programs:

- Presentation logic contains RECEIVE MAP and SEND MAP statements with an additional EXEC CICS LINK() COMMAREA statement to pass control to a program to handle application data. COMMAREA is a means to pass data to a LINKed program.
- Business logic contains all the file I/O statements, such as READ UPDATE and REWRITE. The ADDRESS COMMAREA is to point to the area with the passing data, while RETURN is to get back to the calling program; in this example, the presentation logic program.

In a multi-tier model, the presentation logic can invoke a control program that controls the sequence of business activities implemented in more than one CICS business logic program. The sequence of these business activities can be implemented in a table. That is, a CICS transaction contains a series of CICS programs in sequence. This will certainly provide a way to reuse business logic code across applications. For example, a credit card cash withdrawal transaction consists of a CICS program to check against the card status and the currently available credit limit. When the bank is going to support a cash withdrawal transaction in ATM, an application analyst can include this credit card program as an entry of the new ATM transaction in the table to support this business function.

Figure 89 on page 179 shows the control, presentation (pre and post processing), table driven, and the business logic models.

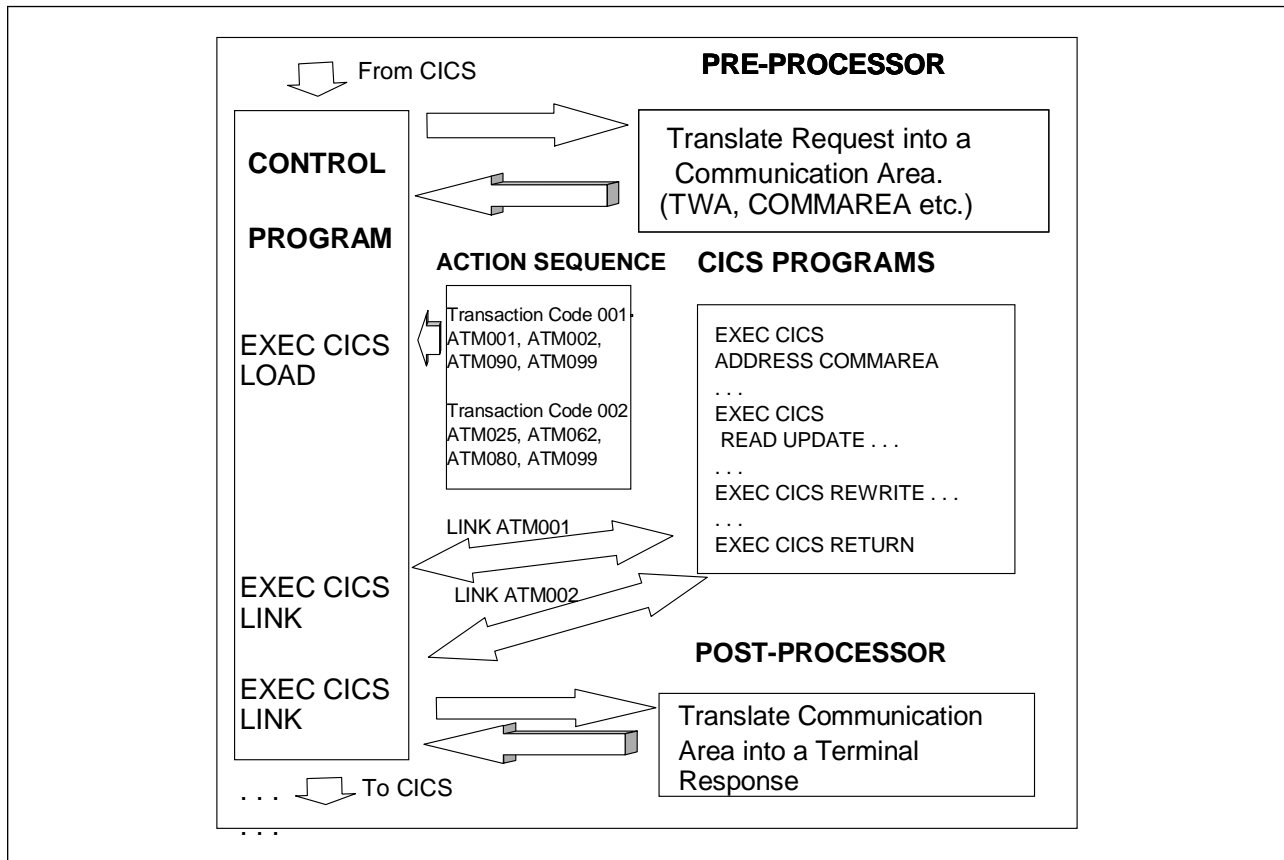


Figure 89. Control, Presentation, Table Driven, and Business Logic Model

In a typical multi-tier design, we see several sets of EXEC CICS statements in separate programs:

- Control Program contains the following EXEC CICS statements:  
LINK to pre/post processors, LOAD PROGRAM (table), LINK to business logic programs
- Pre-Processor contains the following EXEC CICS statements:  
ADDRESS COMMAREA/TWA, RECEIVE {MAP}, RETURN
- Post-Processor contains the following EXEC CICS statements:  
ADDRESS COMMAREA/TWA, SEND {MAP}, RETURN
- Table contains business transaction code and its related programs in sequence.
- Business Logic contains all the file I/O statements, such as READ UPDATE and REWRITE. ADDRESS COMMAREA is to point to the area with the passing data, while RETURN is to get back to the calling program, in this example, the control program.

However, this design has three limitations:

- The processing is serial following the table-defined rules.
- The application will probably have an affinity to a certain CICS region.
- The design is valid for a short-lived business transaction. Otherwise, you have to create permanent output such as extrapartition transient data, VSAM file, or

database for batch processing relating to any business activities that will happen in the future, such as payment in the SALES sample application.

### 9.1.3 Business Transaction Model

In earlier CICS releases, a pseudo-conversational application could be used only to simulate a single conversation with a terminal. A typical long-lived business transaction needs a model that will be reinvoked whenever a new stage of the business transaction is ready to run. Each time it is invoked, the top-level control will run a transaction that implements a particular activity of the business action. This resolves the limitation of the serial processing of the multi-tier table-driven design approach.

---

## 9.2 Sharing Data in and across Transactions

We used to choose CWA or table (either CICS Maintained Table or User Maintained Table) for static information for all transactions during the CICS session. We use TCTUA for the terminal specific information and TWA to hold information for CICS programs sharing data in a particular transaction. We use TS queues to pass information between transactions but have to create and delete queues appropriately. Alternatively, we use a TD queue to trigger action and pass control to a particular transaction. The most common practice is to use the COMMAREA as the means to share data inside and across transactions, that is, data passed from transaction to transaction across a CICSplex. However, as the number of applications grows, the size of COMMAREA is getting larger. CICS holds a COMMAREA in CICS main storage until the terminal user responds with the next transaction. This may be an important consideration if you are using large COMMAREAs because the number of COMMAREAs held by CICS relates to terminal usage and not to the maximum number of tasks in a region at any one time.

In CICS BTS, we can use data-containers, that are named areas of storage and maintained by CICS BTS. In other words, you do not have to code GETMAIN and FREEMAIN for the container. It is like a named COMMAREA. Data-containers are recoverable resources that are being written to repository data set as necessary, and restored at system restart. There are two types of data-containers:

- Process containers

A process container can be read by every activity in the process by issuing a GET CONTAINER command. However, it can be updated only by the root activity, or by a program that has acquired the process by issuing the ACQUIRE PROCESS(data-value) PROCESSTYPE(data-value) command. It is quite similar to CWA or COMMAREA within the scope of a process, but child activities cannot update it. The process container is written and updated by issuing the PUT CONTAINER(data-value) PROCESS/ACQPROCESS FROM(data-value) FLENGTH(data-value) command and can be deleted only by the root activity or by a program that has acquired the process (DELETE CONTAINER command).

- Activity containers

An activity container can be read and updated by the activity itself, by the activity's parent, or by a program that has acquired the activity. It is similar to CWA or COMMAREA within the scope of an activity, which can be a

communication vehicle between the parent and child to pass data, such as status information and is read or updated frequently.

The root activity's container is a special kind of activity container, which can only be accessed by the root activity or a program that has acquired the process. Other children in the process cannot access the container. It is not the same as the process container. It is quite similar to the TWA which is task-specific.

It is worthwhile to mention a new feature introduced in CICS TS 1.3 for data sharing, namely, Coupling Facility Data Tables (CFDT) support. CICS CFDT support is designed to provide rapid sharing of working data within a Parallel Sysplex with update integrity.

CICS data table support has evolved over the years to provide optimized access to different types of data. There are two kinds of data tables that are supported: CICS Maintained Tables (CMTs) and User Maintained Tables (UMTs). Any changes made to the data in CMT are reflected in the underlying VSAM source data set; any changes made to the data in UMT are available only for the lifetime of the data table and are not reflected in the underlying VSAM source data set.

Shared data table support (which is an integral part of CICS/ESA V4 and CICS TS) is optimized for shared access, both sequential and direct, to keyed data. It achieves this using cross-memory sharing so that all CICS regions within an MVS image can read and browse a data table without the overhead of function shipping.

CFDT support provides a means of sharing data with update integrity across a Parallel Sysplex by storing the data in a coupling facility list structure and accessing it through the coupling facility services provided by MVS. The data is in a table that behaves like a sysplexwide form of UMT. However, unlike UMT, a CFDT does not have to be associated with a VSAM source data set and can be loaded directly by user application programs.

The CICS file control commands can access CFDTs. With the API that supports data integrity (READ UPDATE) and imprecise key access (READ GTEQ, READ GENERIC) and browsing (STARTBR, READNEXT), applications that use CFDT support within a Parallel Sysplex can also run unchanged on other releases of CICS or on other CICS platforms.

CFDT support is tailored for the sharing of *informal* data that is usually not a large amount of data, short-term in nature, needs fast access, occasional loss tolerable by user and requiring update integrity. The term *informal shared data* encompasses a variety of data, including scratchpad data, control records, keyed data, shared queues, and a variety of data access modes including read-only, single or multiple updaters, sequential or random access, and random insertion and deletion.

Here are some examples of informal shared data.

- A unique number to be allocated each time an application performs a particular function.
- A look-up table of items to which an application needs access, such as telephone numbers or the numbers of stolen credit cards.
- Working data consisting of a few items, such as the set of customers from the full customer list who are currently of interest.

- Small amounts of data, taken from some larger file or data base, that need further processing.
- Information that *belongs* to the user of the application or relates to the terminal where the application is running

CFDT support offers particular advantages to customers who have already been using UMTs to share informal data with key lengths of up to 16 bytes, because the existing applications can run in a Parallel Sysplex simply by changing the file definition to specify a CFDT rather than a UMT.

You can consider using CFDT in CICS BTS application to share small amounts of informal data when occasional data loss is tolerable but update integrity is required.

For more details about setup, operation, and server command of CFDT, refer to the *CICS TS for OS/390: V1R3 Implementation Guide, SG24-5274*.

---

## 9.3 Task Synchronization

You may have designed your first application in CICS as a pilot application when you first installed CICS in your company. Since it has been implemented, you may have designed a lot more applications and grouped them in different CICS regions under the consideration of performance, operation, and business factors. With the introduction of multiregion operation (MRO) environment, you may have several CICS regions in place. As the applications grow in your company, there is very likely a need to interface between applications. Under some circumstances, time or an event may become involved in the synchronization of two tasks in two different applications in separate CICS regions. The CICS API contains several commands that help you achieve synchronization between tasks.

### 9.3.1 Non-CICS BTS Commands

#### 9.3.1.1 WAIT EVENT Command

The WAIT EVENT command is used to synchronize a task with the completion of an event performed by some other CICS tasks. The completion of the event is signalled (posted) by the setting of a bit pattern into the event control block (ECB). Both the waiting task and the posting task must have direct addressability to the ECB; hence, both tasks must execute in the same AOR. A dynamic transaction routing program, therefore, must ensure that a posting task executes in the same AOR as the waiting task to preserve the application design.

#### 9.3.1.2 RETRIEVE WAIT and START Commands

The RETRIEVE WAIT and START commands are used to synchronize two tasks. A task suspends itself by issuing the EXEC CICS RETRIEVE WAIT command. It needs to be waked up by another task that will restart the waiting task by issuing the EXEC CICS START command. In order to ensure that the second task can wake up the first one, the START command issued by the second task must:

- Be issued in same AOR where the first task waits, or
- Specify the SYSID of the AOR where the first task waits, or
- Specify a TRANSID (the waiting task) that is defined as a remote transaction residing on the AOR where the first task waits.



An application design based on the remote TRANSID technique only works for two AORs. An application design that uses the SYSID option on the START command only works for multiple AORs if all AORs have connections to all other AORs (which may not be desirable). There is no acceptable way to use these programming techniques in a dynamic transaction routing environment except by imposing restrictions on the routing program.

### 9.3.1.3 DELAY and CANCEL REQID Commands

With this technique, one task can resume another task that has been suspended by a DELAY command. For example, an ATM program initiates a credit card program with a START command and then issues a DELAY command with some value in the INTERVAL parameter. When it resumes, the ATM program can check whether the interval has expired or it is CANCEL REQID by another task. If it resumes by CANCEL, the ATM application will perform its normal business logic.

A DELAY request can only be cancelled by a different task from the one issuing the DELAY command, and the CANCEL command must specify the REQID associated with DELAY command. Both the DELAY and CANCEL commands must specify the REQID option to use this technique. A TS storage queue is one way that can be used to pass the REQID of the POST request from task to task.

However, if the CANCEL command is issued by a transaction that is initiated from a terminal, it is possible that the transaction could be dynamically routed to the wrong AOR.

### 9.3.1.4 POST/CANCEL REQID Commands

The POST command is used to request notification that a specified time has expired. Another transaction can force notification, as if the specified time has expired, by issuing a CANCEL of the POST request. The time limit is signalled (posted) by CICS by setting a bit pattern in the event control block (ECB). To determine whether notification has been received, the requesting transaction has either to test the ECB periodically or to issue a WAIT command on the ECB. A TS storage queue is one way that can be used to pass the REQID of the POST request from task to task.

However, like the other CICS commands discussed above, the CANCEL command with REQID will cause an inter-transaction affinity problem. There is no acceptable way to use these programming techniques in a dynamic transaction routing environment except by imposing restrictions on the routing program.

## 9.3.2 CICS BTS Tools

In CICS BTS, there are events, including timer events, to handle the issues we have to face in the classic CICS environment. A CICS BTS event is a means by which CICS BTS signals the progress in a process. Furthermore, an event can be composed of several logical expressions connected by boolean AND or OR operators (*composite event*). This powerful facility will definitely help application designers to handle the more complex business rules of the real world.

---

## 9.4 Recovery Considerations

If a single CICS transaction fails, uncommitted changes to recoverable resources are automatically backed out by CICS. However, a CICS BTS transaction is usually a long-lived business transaction that spans across more than one CICS transaction. If any activity with its own transaction code fails, any committed changes in the previous CICS activities (transactions) may need to be reversed or even modified. Reversing the actions of completed activities is often referred to as *compensation*.

Compensation means undoing the actions that an activity has taken under normal circumstances. Compensation for authorizing a credit card purchase might be to reverse this business transaction. Compensation for accepting an order might be to cancel the order. Typically, this transaction or program to reverse the result of the normal processing has already been implemented due to a business need. It provides a means to reverse a business transaction done by mistake.

There are two ways to implement compensation.

- A single program to handle the normal flow of a business transaction and its reversal (compensation)

In CICS BTS, you must first issue a RESET ACTIVITY command to bring the activity to its initial stage.

- A different program to handle the reversal business action.

In CICS BTS, you must define and run a new activity with the same input data that was passed to the activity for the normal processing.

---

## 9.5 Dealing with Exception Conditions

Every time you process an EXEC CICS command in one of your applications, CICS automatically raises a condition, or return code, to tell you what happened. You can choose to have this condition, which is usually NORMAL, passed back by the CICS EXEC interface in the RESP and RESP2 options of the command. Alternatively, you may obtain this value by reading it from the EXEC interface block (EIB).

If something out of the ordinary happens, you get an exception condition, which simply means a condition other than NORMAL. By testing this condition, you can find out what has happened and, possibly, why.

There are HANDLE CONDITION ERROR, IGNORE CONDITION and HANDLE ABEND commands with RESP and NOHANDLE options in the existing CICS.

With CICS BTS, there are additional commands to handle this new application architecture. These are CHECK ACTIVITY and CHECK ACQACTIVITY commands. CHECK ACTIVITY returns the completion status of an activity. They are typically used to check the success of a previous RUN ACTIVITY or LINK ACTIVITY command. It allows you to discover whether an activity completed successfully or needs to be reactivated to complete its processing.

CHECK ACTIVITY is issued by a parent activity or a program that has acquired an activity through the ACQUIRE ACTIVITYID command. It can be used to check its child's activities:

- That have completed
- That have not completed
- That were requested to run asynchronously
- That were requested to run synchronously

The COMPSTATUS option returns a CVDA value indicating the completion status of the activity: NORMAL, FORCED (by CANCEL ACTIVITY), INCOMPLETE and ABEND. The MODE options returns the processing state of the activity: ACTIVE, CANCELLING, COMPLETE, DORMANT, and INITIAL.

The parent activity issues the CHECK ACTIVITY command to see if its child's activity has completed (either successfully or unsuccessfully). This command causes CICS to delete the activity-completion event. CICS deletes the completion event only if the parent issues a CHECK ACTIVITY command. A program that executes outside a CICS BTS process and acquires an activity through the ACQUIRE ACTIVITYID command will not cause CICS to delete its child's completion event if it issues a CHECK ACTIVITY command.

In case of ABEND, CHECK ACTIVITY will give you further information about how the activity failed. The CHECK ACTIVITY command gives you an abend code, ABCODE, and the name of the program in which the abend occurred, ABPROGRAM.

In CICS BTS, the parent activity can retry the child activity in case of its failure. The parent activity issues a RESET ACTIVITY command to bring its child to initial state and then issues a RUN ACTIVITY command.

---

## 9.6 Access to System Information

There are five new system programming interface (SPI) commands introduced by CICS BTS. Four of them deal with process types: CREATE PROCESSTYPE, DISCARD PROCESSTYPE, INQUIRE PROCESSTYPE, and SET PROCESSTYPE. The INQUIRE TASK command has been enhanced to include ACTIVITY, ACTIVITYID, PROCESS, and PROCESSTYPE parameters. If you are using CICS BTS in a Parallel Sysplex, it is strongly recommended to use CICSplex SM instead of issuing the SET and DISCARD PROCESSTYPE commands.

---

## 9.7 Program Structure

We have discussed briefly typical program structures of the pre-CICS BTS era. We now concentrate on the CICS BTS application program.

Since CICS BTS is event driven, the program structure differs dramatically from the previous models. Within the CICS BTS model, we distinguish two sub-models in terms of program structure:

- Parent and child model
- Client/server model

## 9.7.1 Parent and Child Model

The parent or the root activity manages the ordering, inter-relationship, and execution of the child activities.

A typical root activity program should start with an EXEC CICS RETRIEVE REATTACH EVENT statement to analyze which event has triggered it to execute. If it is the first time (with system event DFHINITIAL), it performs some initialization functions, such as finding out its process name, process type, or getting a process container. A typical activity program determines which of the possible events caused it to be attached and what to do as a result. Therefore, the mainline in the root activity program should contain a subroutine to prepare each child activity and kick it off. Each subroutine could contain the following CICS BTS API statements:

- EXEC CICS DEFINE ACTIVITY() TRANSID() with EVENT() as one of the optional parameters.
- EXEC CICS GET CONTAINER() INTO() if there is one.
- EXEC CICS PUT CONTAINER() FROM()
- EXEC CICS RUN ACTIVITY() Synchronous | Asynchronous

A typical sequence (somewhat simplified) is:

1. The root activity requests BTS to run a child activity (possibly several child activities) and to notify it when the child has completed.
2. The root activity becomes dormant while waiting for the child activity to complete.
3. BTS reattaches the root activity because the child activity has completed.
4. The root activity requests the next child activity to run.
5. Steps 1 through 4 are repeated until the business transaction is complete.

The root activities of the processes we have investigated have all followed a similar pattern. In view of this fact, we propose a Universal Process Driver (see Chapter 10, "Universal Process Driver Model" on page 195).

Figure 90 on page 187 shows, in COBOL pseudo-code, the root activity.

```

      .
Data Division.
Working-Storage Section.
      .
01  Event-Name                pic x(16).
   88  DFH-Initial            value 'DFHINITIAL'.
   88  Delivery-Complete      value 'Delivry-Complete'.
   88  Delivery-Confirmed     value 'Delivry-Confirmd'.
   88  Invoice-Complete        value 'Invoice-Complete'.
   88  Payment-Due            value 'Payment-Due&apos'.
   88  Payment-Complete       value 'Payment-Complete'.
   88  Reminder-Expired       value 'Remindr-Expired'.
   88  Reminder-Complete      value 'Remindr-Complete'.
      .
Procedure Division.
      .
EXEC CICS RETRIEVE REATTACH EVENT(Event-Name)
      RESP(RC) END-EXEC
      .
If RC NOT = DFHRESP(NORMAL)
      .
End-If.
      .
Evaluate True
  When DFH-Initial
    Perform Initial-Activity
    Perform Order-Activity
    Perform Order-Response
    Perform Delivery-Activity
  When Delivery-Complete
    Perform Delivery-Response
    Perform Delivery-Confirmation
  When Delivery-Confirmed
    Perform Confirm-Response
    Perform Invoice-Activity
  When Invoice-Complete
    Perform Invoice-Response
    Perform Payment-Activity
  When Payment-Due
    Perform Payment-Due-Response
  When Payment-Complete
    Perform Payment-Response
  When Reminder-Expired
    Perform Reminder-Expired-Response
  When Reminder-Complete
    Perform Reminder-Response
  When Other
      .
End Evaluate.
      .
EXEC CICS RETURN END-EXEC
      .

```

Figure 90. Pseudo-code of a Root Activity

Within the DFHINITIAL event, the last subroutine Perform Delivery-Activity defines an activity with the event Delivery-Complete, which happens to be the next event trigger in the Evaluate True COBOL statement. That is, the name of the subroutine that defines the event, and the *When event* in the Evaluate True COBOL statement are next to each other.

In case of payment-activity and payment-due-response activity, they contain time events also. We put the timer event handling before the activity event handling subroutine in the Evaluate True COBOL statement.

Table 10 on page 188 shows the subroutine name that creates the corresponding event in the pseudo-code. If there is no event handling in the subroutine, CICS BTS API is shown instead.

<i>Table 10. Subroutine Name Which Creates the Event in the Pseudo-code</i>	
<b>Subroutine name</b>	<b>Event name (or *API or *function)</b>
Delivery-Activity	Delivery-Complete
Delivery-Confirmation	Delivery-Confirmed
Invoice-Activity	Invoice-Complete
Payment-Activity	Payment-Due(timer) and Payment-Complete
Payment-Due-Response	Reminder-Expired(timer) and Reminder-Complete
Payment-Response	*Check Activity(payment)
Reminder-Expired-Response	*Compensation
Reminder-Response	*Check Activity(reminder)

If you use a different name of the event in the Evaluate True COBOL statement and the DEFINE ACTIVITY EVENT parameter, you have to have a table for their corresponding entries in the working-storage section of Data Division in the program. In our example, the 88 level name corresponds to the Evaluate True COBOL statement, and the 88 level value corresponds to the name in the EVENT parameter defined to the CICS BTS. The administration of events is crucial in developing the business transaction by using CICS BTS.

## 9.7.2 Client/Server Model

Most business transactions require human involvement. They cannot be restarted automatically by CICS BTS because the user may not be ready to process the work. For example, the user cannot invoke payment process without receiving the check from the customer.

To permit interaction with the outside world, CICS BTS allows a program executing outside a process to acquire access to an activity within the process. In this way, CICS BTS supports client/server processing.

In the handshake process (first time communication between client and server), the server defines some input events for which it may be reinvoked and returns to the client. The client invokes the server with a named input event and puts some input data in a data-container. From this input event, the server determines what actions it needs to take. It returns any output for a client in a data-container.

After the client has got the output from the server, it releases the server process that has been acquired. At this moment, the server process is maintained only by CICS BTS. The client can acquire the server process again and request it to perform another business action. Eventually, the client tells the server to finish this service, and the server informs itself that it should not be reinvoked in this client/server transaction.

Figure 91 on page 189 shows, in COBOL pseudo-code, the mainline of the client program.

```

.
Data Division.
Working-Storage Section.
.
01 Unique-Reference          pic s9(8) comp.
.
01 Process-Type              pic x(8)  value 'Servers'.
.
01 Event-Name                pic x(16) value low-values.
.
01 Work-Buffer.
.
01 Work-request              Pic x.
88 Work-hAndshake           value 'A'.
88 Work-Continue             value 'C'.
88 Work-End                  value 'E'.
.
Procedure Division using DFHEIBLK DFHCOMMAREA.
.
EXEC CICS SEND MAP ...
    RESP(data-area) END-EXEC
.
EXEC CICS RECEIVE MAP ...
    RESP(data-area) END-EXEC
.
Move ..unique.. TO Unique-Reference
Move ..request.. TO Work-Request
.
Evaluate True
  When Work-hAndshake
    Perform Define-Process-Put-Container-Run-Process
  When Work-Continue
    Move 'INPUT-EVENT-1' to Event-Name
    Perform Acquire-Process-Put-Container-Run-Process
  When Work-End
    Move 'INPUT-EVENT-2' to Event-Name
    Perform Acquire-Process-Put-Container-Run-Process
  When Other
.
End Evaluate.
.
EXEC CICS GET CONTAINER('Server-Out')
    ACQPROCESS INTO(Work-Buffer)
    RESP(data-area) RESP2(data-area) END-EXEC
.
EXEC CICS SEND MAP ...
    RESP(data-area) END-EXEC
.
EXEC CICS RETURN END-EXEC
.

```

Figure 91. Pseudo-code of the Mainline of a Client Program

Figure 92 on page 190 shows, in COBOL pseudo-code, the mainline of a server program.

```

.
Data Division.
Working-Storage Section.
01 Event-Name          pic x(16).
   88 DFH-Initial      value 'DFHINITIAL'.
   88 SRV-Request      value 'SRV-REQUEST'.
01 Sub-Event-Name     pic x(16).
   88 SRV-Work         value 'INPUT-EVENT-1'.
   88 SRV-Shutdown    value 'INPUT-EVENT-2'.
01 Input-Buffer.
.
01 Output-Buffer.
.
01 State-Buffer.
.
Procedure Division.
Begin-Process.
.
EXEC CICS RETRIEVE REATTACH EVENT(Event-Name)
      RESP(data-area) RESP2(data-area) END-EXEC
.
Evaluate True
  When DFH-Initial
    Perform Define-Input-Events
    Perform Server-work
  When SRV-Request
    Perform Server-Event
  When Other
.
End Evaluate.
.
EXEC CICS RETURN END-EXEC
.
Server-Event.
.
EXEC CICS RETRIEVE SUBEVENT(Sub-Event-Name) EVENT(Event-Name)
      RESP(data-area) RESP2(data-area) END-EXEC
.
Evaluate True
  When SRV-Work
    Perform Server-Work
  When SRV-Shutdown
    Perform Server-Shutdown
  When Other
.
End Evaluate.
.
Define-Input-Events.
.
EXEC CICS DEFINE INPUT EVENT('INPUT-EVENT-1')
      RESP(data-area) RESP2(data-area) END-EXEC
.
EXEC CICS DEFINE INPUT EVENT('INPUT-EVENT-2')
      RESP(data-area) RESP2(data-area) END-EXEC
.
EXEC CICS DEFINE COMPOSITE EVENT('SRV-REQUEST') OR
      SUBEVENT1('INPUT-EVENT-1')
      SUBEVENT2('INPUT-EVENT-2')
      RESP(data-area) RESP2(data-area) END-EXEC
.

```

Figure 92 (Part 1 of 2). Pseudo-code of the Mainline of a Server Program



```

Server-Work.
.
EXEC CICS GET CONTAINER('Server-In') INTO(Input-Buffer)
      RESP(data-area) RESP2(data-area) END-EXEC
.
If DFH-Initial
  EXEC CICS DEFINE ACTIVITY('Work')
        TRANSID('SWRK')
        PROGRAM('SWRKPGM')
        RESP(data-area) RESP2(data-area) END-EXEC
.
Else
  EXEC CICS GET CONTAINER('Previous-State') INTO(State-Buffer)
        RESP(data-area) RESP2(data-area) END-EXEC
.
End-If.
.
EXEC CICS PUT CONTAINER('Work-Input')
      ACTIVITY('Work') FROM(Input-Buffer)
      RESP(data-area) RESP2(data-area) END-EXEC
.
EXEC CICS RUN ACTIVITY('Work')
      SYNCHRONOUS
      RESP(data-area) RESP2(data-area) END-EXEC
EXEC CICS CHECK ACTIVITY('Work') COMPSTATUS(status)
      RESP(RC) RESP2(data-area) END-EXEC
.
If RC NOT = DFHRESP(NORMAL)
.
End-If.
.
If status NOT = DFHVALUE(NORMAL)
.
End-If.
.
EXEC CICS GET CONTAINER('Work-Output')
      ACTIVITY('Work') INTO(Output-Buffer)
      RESP(data-area) RESP2(data-area) END-EXEC
.
EXEC CICS PUT CONTAINER('Previous-State') FROM(State-Buffer)
      RESP(data-area) RESP2(data-area) END-EXEC
.
EXEC CICS PUT CONTAINER('Server-Output') FROM(Output-Buffer)
      RESP(data-area) RESP2(data-area) END-EXEC
.
Server-Shutdown.
EXEC CICS DELETE EVENT('INPUT-EVENT-1')
      RESP(data-area) RESP2(data-area) END-EXEC
.
EXEC CICS DELETE EVENT('INPUT-EVENT-2')
      RESP(data-area) RESP2(data-area) END-EXEC
.
EXEC CICS DELETE EVENT('SRV-REQUEST')
      RESP(data-area) RESP2(data-area) END-EXEC
.
EXEC CICS RETURN ENDACTIVITY
      RESP(data-area) RESP2(data-area) END-EXEC
End Program.

```

Figure 92 (Part 2 of 2). Pseudo-code of the Mainline of a Server Program



---

## Part 4. Universal Process Driver



---

## Chapter 10. Universal Process Driver Model

Looking at the various applications we have discussed in this book, you may have determined that each process is controlled by an all-pervasive root activity. In most cases, the dependent activities are only responsible for a single element within the business transaction automated with this process.

The root activity contains the major part of the event logic. It is responsible for executing the business control logic. Each root activity described in this book is developed using the same pattern.

- When the process starts the root activity with the DFHINITIAL event, several activities (children) are defined and executed. Additionally, in some cases, events are defined (input event, for example), or containers are written.
- When reattached because of the completion of an activity, the activity is checked, a container may be read, and a composite event is deleted.

These simple examples show that each CICS BTS action can be formally described using systems analysis methodologies.

Therefore, we are convinced that it is possible to develop a program driven only by data to execute most of our processes. The data can reside in a database or (for performance reasons) in storage. In this chapter, we introduce a possible design of such a universal process driver (UPD).

Our UPD is not complete in its design — it provides a brief overview of what is necessary to run parent activities in this universal form. We only introduce the logic and the control data needed; however, you will also need naming conventions for activities, processes, and containers. We strongly recommend that you develop an interface concept covering data exchange between modules and activities, and you might also generate elements of your application from a repository. There are many things to consider; therefore, this section should only be seen as a starting point for your discussion.

---

### 10.1 Database Process Control Catalogue

Our UPD uses a set of tables to control the process flow, a Process Control Catalogue (PCC). The database is not complete, but it shows how our driver can work using only data to control the logic.

Figure 93 on page 197 shows the database and the relationships among the tables.

Our database consists of the following tables:

- ProcessTable  
This is the entry table. It contains the process name, the process type, explains how the dynamic part of the process name is to be built, refers to the root activity, and assigns each process a unique process number. See Table 11 on page 199.
- ActivityTable

The activity table contains the program name, the transaction ID, and the activity name, and references the event table when an event name is necessary with the activity (see Table 12 on page 200).

- EventTable

Referenced from the activity table, this table gives information about all events. It contains the name, the type, and additional parameters for execution (date, time, duration). For event type COMPOSITE, it refers to the SubEventTable (see Table 13 on page 201).

- SubEventTable

This table is reconnected to the event table (see Table 14 on page 201).

- ProcessControl

This table is the major table at runtime. It contains the process number to identify each process and is unique in the database.

The table is used to control the process flow. Therefore, each activation can be identified by the column InActivity, which refers to the activity table.

To identify the reattachment event, the associated column refers to the event table.

Each activation consists of as many steps as necessary. In the column Call, the command to be executed is specified. This command can be:

- Any CICS BTS command
- CICS commands, such as LINK or XCTL
- A program call
- An internal UPD-command such as IF

The column Activity refers to the activity table and contains the activity's number. If an event is connected with a command, its number has to be specified in the column EventNumber. The columns Container, ProgramName, ProcessName, and UserId contain the appropriate names when necessary with a command.

The column Condition has several meanings. You could specify the conditions returned from the CICS commands, or it could contain self-defined values such as *OK* or *more*. The column GoTo shows which step has to be executed next when the Condition is true. Possible values are a step within the running reattachment event (for example, 300) or a step in another reattachment event (for example, 26/70). This means that the logic has not been duplicated in the table.

To implement parts in the process that are specific for only one process, it is useful to develop this logic in a separate module and to call this module. To implement modules like these, it is necessary to define an interface to be used by the modules. For example, it may be useful to call a program for each container operation. The interface must cover fields, such as:

- Address to work from
- Length of data
- Return code

See Table 15 on page 202 and Table 16 on page 203 for an example of a part of a process control table. For a full layout, refer to Appendix C, “UDP Process Control Catalogue” on page 267.

Our database is not complete — time did not permit the design of a complete Universal Process Driver. But, we are convinced that this basic design can be expanded into a total system that can help you to run processes without developing a separate root activity.

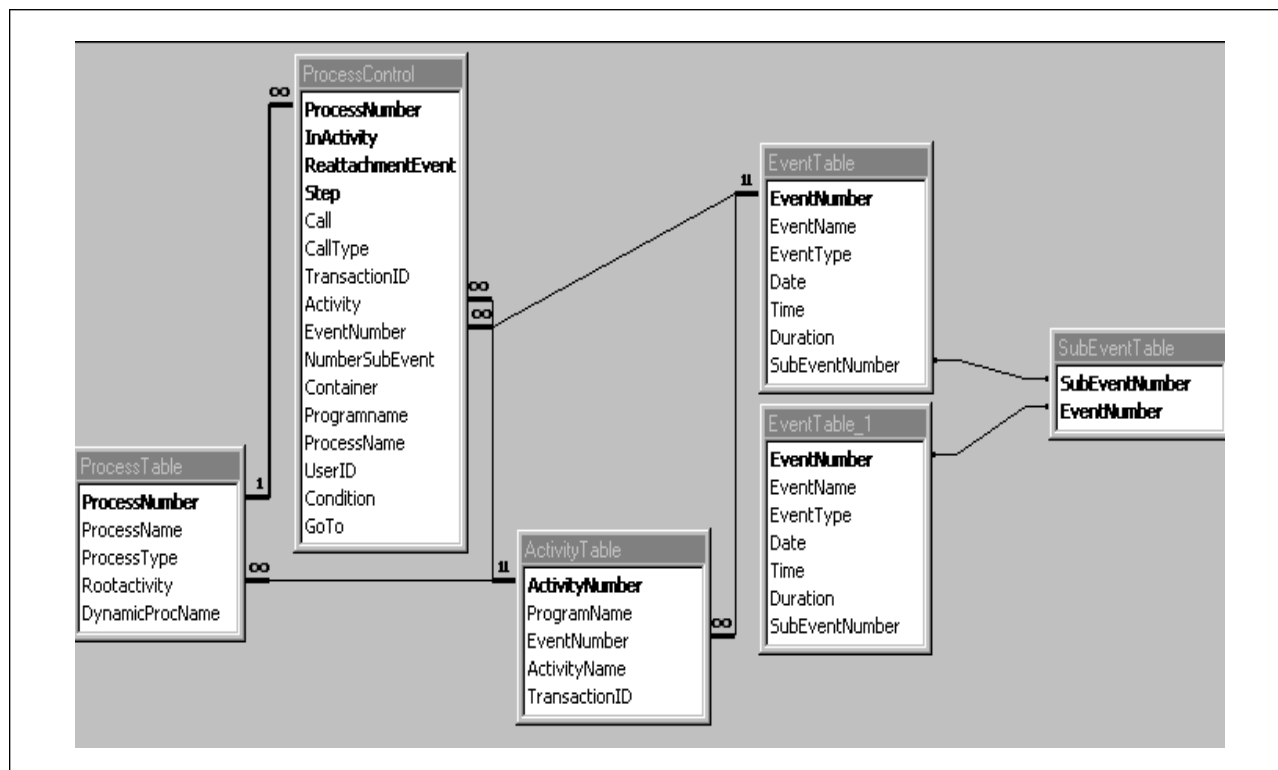


Figure 93. Process Control Catalogue for a Universal Process Driver

## 10.2 Process Definition

We recommend the use of a program that accesses the UPD-database (see Figure 93) to provide the unique process name. This program should be called by the program that wants to start the process. The process type and the generic process name are used to make the unique process name. Therefore, it is useful to have naming conventions for generic process names. The transaction ID to be used with this process is taken from the process table. With the unique process name, the process number, which is a database internal key, is provided. This number should be written to the process container.

## 10.3 Universal Process Driver Global Description

When started with a DFHINITIAL event, the program first reads the database to discover the process number and the root activity's number. If the process number is provided in the process container, this value will be used. Then it reads the event number of the reattachment event. Next, there is a request to read the first process step. The selection criteria are the process number, the current activity

number (that is, the root activity number from the first query), and the number of the reattachment event. See Figure 94 on page 198 for an illustration of the start logic.

At this point, the program has enough information to start working. Once the command to be executed with this step is known, the program branches to the different command execution areas.

When a command is executed, the database is read again until no more steps are required. Then the status data is stored (for example, in a TS queue).

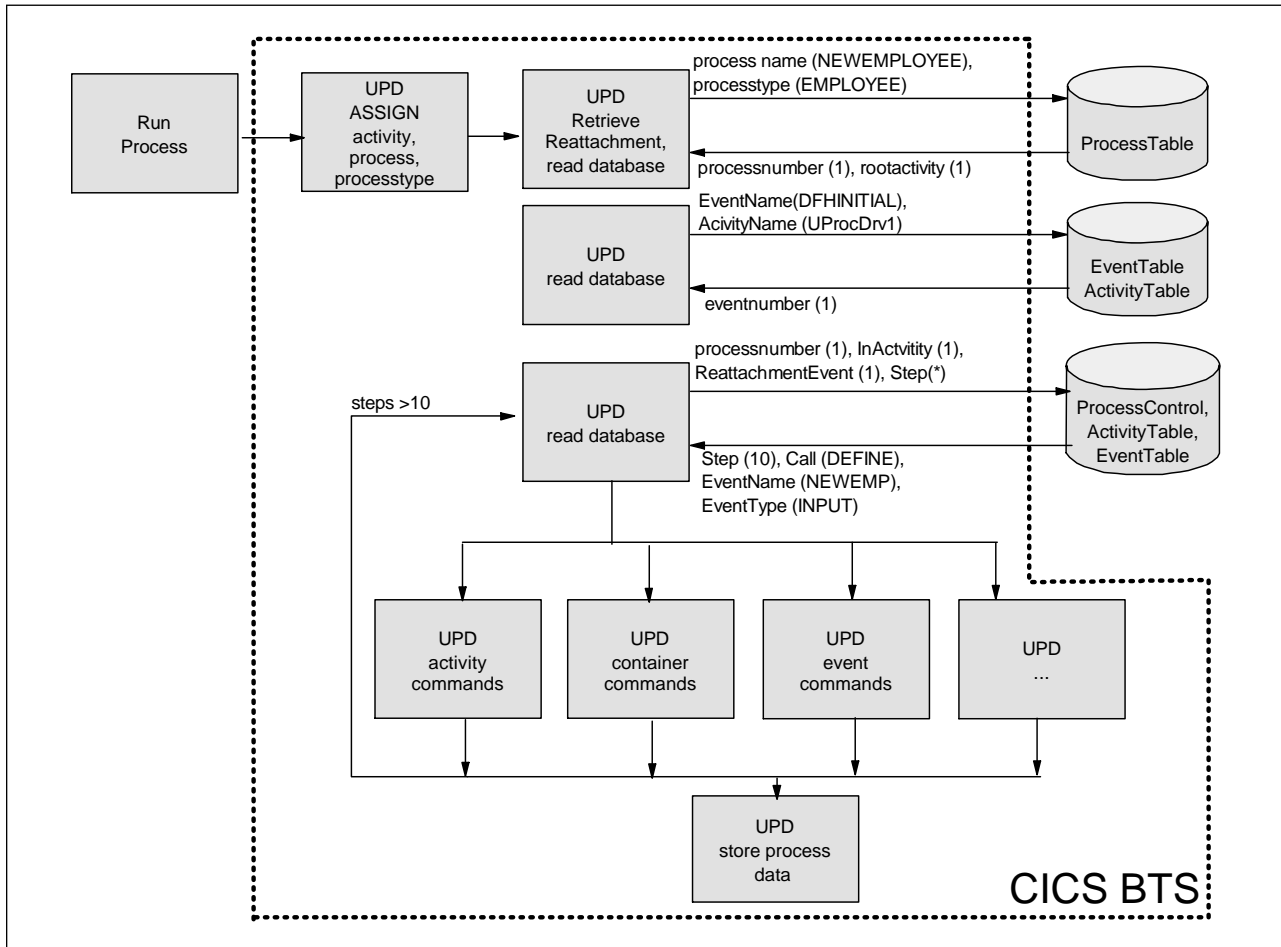


Figure 94. Universal Process Driver Logic with Event DFHINITIAL

When the reattachment event is not DFHINITIAL, UPD restores the status data, executes an ASSIGN command to get the current activity name, and reads the database to discover the current EventNumber and the ActivityNumber. This is illustrated in Figure 95 on page 199.

The logic continues, as previously described, to execute the requested commands.

We do not describe how to deal with data, although several concepts are possible. One method could be to decide that a child activity always has two containers. The first container stores structure data that describes each field (for example, type, length, current count) and an offset or a pointer to where the data is stored. In the second container, all the actual data are stored, possibly using a hash algorithm.



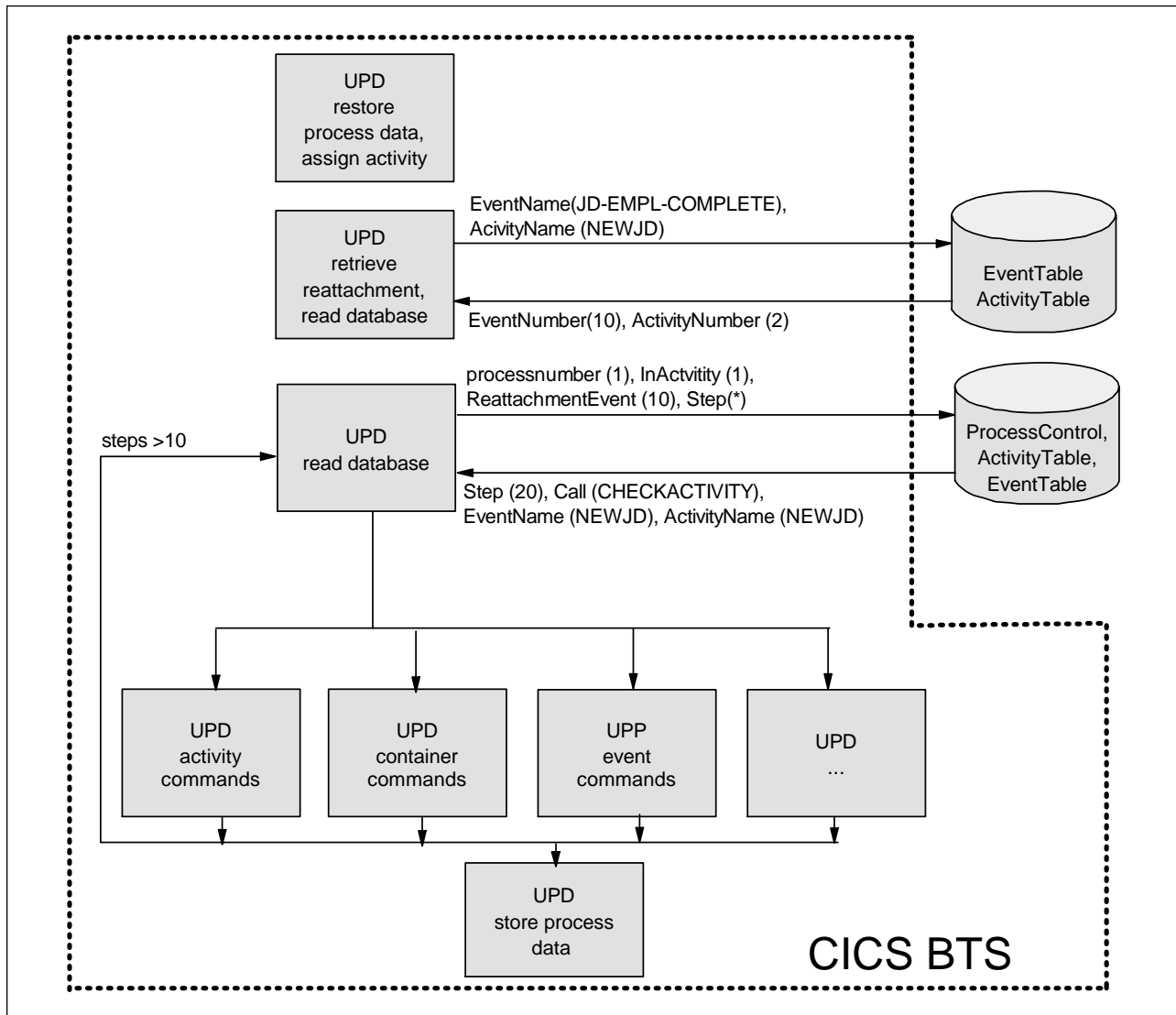


Figure 95. Universal Process Driver Logic with an Event Other Than DFHINITIAL

## 10.4 Example of a Process Description in Process Control Catalogue

We describe the sample process NEWEMPLOYEE, introduced in Figure 76 on page 140 in our PCC. Additionally, we describe a small second process called NEWJOBDESCRIPTION to show how you can start reuse with just two processes.

Table 11 shows the process table for this example.

Table 11. ProcessTable

ProcessNumber	ProcessName	ProcessType	Rootactivity	DynamicProcName	TransactionID
1	EMPL&	EMPLOYEE	1	EIBTASKN	EMPR
2	NEWJD&	ORG	1	EIBTASKN	NDNW

The ProcessTable contains only two processes to illustrate the example. The first process is NEWEMPLOYEE. The second process is NEWJOBDESCRIPTION, introduced to show how to separate several activities in a new process. The field

DynamicProcName is used to give the defining program the rule how to specify the process name. If the field ProcessName contains a substitution sign (&), this column is used to generate the unique process name. The field can contain:

- &taskn  
This entry says that the task number of the defining transaction must be added to the process name.
- &fieldname  
The entry is the name of a field whose content has to be added to the process name.
- &systeme  
With this example, the actual time is added to the process name.

Table 12 shows the activity table for this example.

*Table 12. ActivityTable*

ActivityNumber	PgmName	EventNumber	Activity	TrxID	DynActNme
1	UPD00001		UProcDrv1	EMPR	
2	JDIS0001	11 (NEWJD)	NEWJD	JOBN	
3	JDOU0001	2 (CONNECTOU)	JOBEORGU	JOBU	
4	JDTA0001	3 (CONNECTTASK)	JOBETASK	JOBT	
5	EMPL0001	4 (NEWEMPL)	NEWEMPL	EMPN	
6	EMR00001	5 (CONNECTRole)	EMPLROLE	EMPO	
7	EMTA0001	6 (CONNECTTASK)	EMPTASK	EMPT	
8	JDR00001	13 (CONNECTROLE)	JOBEROLE	JOB0	
9	JDCS0001	14 (CONNECTCS)	JOBECOSC	JOB5	
10	EMCS0001	16 (CONNECTEMP-CS)	EMPLCOSC	EMPS	
11	EMJD0001	17 (CONNECTEMP-JD)	NEWEMJD	EMJD	
12	TASK0001	19 (NewTaskComplete)	NewTask	TASK	

Each value in brackets in column EventNumber represents the associated value to the number in table EventTable, and would not normally appear in such a table.

This table contains all activities. They are described by their unique number. The name of the program that implements the activity, the transaction ID, the activity name and the number of the associated event are included in the table.

If the activity name has to be generic, you can use substitution-signs (for example, '&') to show where the dynamic part of the activity name has to be inserted.

For example, with the HOLIDAY process, we used a generic activity name to execute several excursion activities. The activity name in the table is EX&-BOOK. The column DynActNme contains 1++ to show that & has to be replaced by a number beginning with one that will be increased for a second definition of an excursion activity. Refer to Appendix C, "UDP Process Control Catalogue" on page 267 to pursue this example. Table 13 on page 201 shows the event table for this example.

Table 13. EventTable							
EventNumber	EventName	EventType	SubEventNumb	Date	Time	Duration	DynamicName
1	DFHINITIAL	System					
2	CONNECTOU	COMPLETION					
3	CONNECTTASK	COMPLETION					
4	NEWEMPL	COMPLETION					
5	CONNECTRo1e	COMPLETION					
6							
7	EMPLREL-COMPLETE	COMPOSITE	3				
8	JD-REL-COMPLETE	COMPOSITE	2				
10	JD-EMPL-COMPLETE	COMPOSITE	1				
11	NEWJD	COMPLETION					
12	NEWEMP	INPUT					
13	CONNECTROLE	COMPLETION					
14	CONNECTCS	COMPLETION					
15	ROLE-CS-CONNECT	COMPOSITE	4				
16	CONNECTEMP-CS	COMPLETION					
17	CONNECTEMP-JD	COMPLETION					
19	NewTaskCOMPLETE	COMPLETION					
20	NOREQ	TIMER				28days	

This table contains an entry for each event used with the two processes. In column SubEventNumb, you find the referral to the SubEvent table that contains all subevents a composite event is formed from.

The columns Date, Time, and Duration are used to define timers. It makes no sense to use absolute dates or time entries in this table. Therefore, you have to design an algorithm to define timers. For example, a date can be set depending on the content of a field in a specific container.

Table 14 shows the subevent table for this example.

Table 14. SubEventTable	
SubEventNumb	EventNumber
1	4 (NEWEMPL)
1	11 (NEWJD)
2	2 (CONNECTOU)
2	3 (CONNECTTASK)
3	5 (CONNECTRo1e)
3	3 (CONNECTTASK)
4	13 (CONNECTROLE)
4	14 (CONNECTCS)

The values in brackets in column EventNumber represent the associated value to the number in table EventTable, and would not normally be seen.

Table 15 on page 202 and Table 16 on page 203 show a part of the table to control the process flow (ProcessControlTable). Refer to Appendix C, "UDP Process Control Catalogue" on page 267 to see a complete printout of our PCC. The printout is made by joining the tables of our PCC. Therefore, you find all

activities, events, and processes named, rather than being referred to by their internal number.

The columns ProcessNumber, InActivity, Re.Event, and Step form a primary key.

*Table 15. ProcessControlTable, Part 1*

ProcessNumber	InActivity	Re.Event	Step	Call	CallType	TrxID	Activity	EventNumber
1	1	1	10	DEFINE				12
1	1	1	19	DEFINE		JDIS	2	11
1	1	1	20	RUN	ASYNC		2	
1	1	1	29	DEFINE		EMPL	5	4
1	1	1	30	RUN	ASYNC		5	
1	1	1	40	DEFINE				10
1	1	10	10	CHECK			2	
1	1	10	20	GET				
1	1	10	30	CHECK			5	
1	1	10	40	GET				
1	1	10	50	DELETE				10
1	1	10	60	RUN	SYNC		11	17
1	2	1	10	GET				
1	2	1	20	RUN	ASYNC		3	2
1	2	1	30	RUN	ASYNC		4	3
1	2	1	40	DEFINE				8

Table 16. ProcessControlTable, Part 2

ProcessNumber	InActivity	Re.Event	Step	Call	Container	Pgmname	ProcName	UserID	Condition	GoTo
1	1	1	10	DEFINE						
1	1	1	19	DEFINE						
1	1	1	20	RUN						
1	1	1	29	DEFINE						
1	1	1	30	RUN						
1	1	1	40	DEFINE						
1	1	10	10	CHECK						
1	1	10	20	GET	JDIS					
1	1	10	30	CHECK						
1	1	10	40	GET	EMPL					
1	1	10	50	DELETE						
1	1	10	60	RUN						
1	2	1	10	GET	PROCESS					
1	2	1	20	RUN						
1	2	1	30	RUN						
1	2	1	40	DEFINE						

## 10.5 Conclusion

The discussion in this chapter has been purely theoretical, but we are convinced that the UPD approach can be used. Similar applications, not using CICS BTS, of course, are available on the market.



---

## Part 5. System Considerations





---

## Chapter 11. CICS Business Transaction Services Setup

In this chapter, we describe the CICS BTS setup, the facilities available for problem determination, and the operational aspects of CICS BTS implementation.

Before you can use CICS BTS for your applications, you have to set up the CICS BTS environment in your CICS regions. The setup consists of the following tasks, some of which are optional:

1. The PROCESSTYPE resource definition
2. The local request queue data set
3. The CICS BTS repository data set
4. Workload Management Environment (optional)
5. The audit log (optional)

---

### 11.1 Process Resource Definition

One of the first definitions you provide is the process type. The process type is a means of categorizing your various business transactions to distinguish them from one another, and it is also useful for accounting or error tracking. It is the category to which a process belongs. All the activities in the process inherit the same process type attribute. You define the process type, using the new RDO object, PROCESSTYPE, as shown in Figure 96.

```
OBJECT CHARACTERISTICS                                CICS RELEASE = 0530
CEDA View PROCesstype( ORDER )
  PROCesstype   : ORDER
  Group         : CBTS
  Description   : PROCESS TYPE DEF. FOR TYPE SALES ORDER
INITIAL STATUS
  Status       : Enabled           Enabled | Disabled
DATA SET PARAMETERS
  File         : PTYPSALE
AUDIT TRAIL
  AUDITLog     : CBTSLOG
  AUDITLevel   : Full             Off | Process | Activity | Full
                                           SYSID=PAA1 APPLID=SCSCPAA1
```

Figure 96. CEDA View of a Process Type - Sales Sample Application

The file name defined here (PTYPSALE) names the repository file used for all CICS BTS processes of the type ORDER as in the SALES sample application. See 11.3, “CICS BTS Repository Data Set” on page 209 for more details.

Optionally, you can specify an audit log journal name on the CEDA panel (Figure 96). If you do, a corresponding journal model definition must be provided as well. Audit log entries are produced at the level you specify for AUDITLevel. The level can be:

- Off - If you do not want an audit trace to be produced.

- Process - Produces a trace entry only at a process level, for example, at invocation or reinvocation of a process or termination of a process.
- Activity - Produces trace entries for each invocation and termination of an activity.
- Full - You get the maximum trace entries in the audit log including such information as events that have fired.

For more details, see 11.5, “Audit Log” on page 211.

## 11.2 Local Request Queue Data Set

The local request queue data set should already be set up. It is one of the installation tasks for CICS TS 1.3. This data set is needed even if you do not use CICS BTS. The local request queue holds a copy of pending requests destined for this specific CICS region, thus, providing recoverability for such requests as CICS BTS timer events.

Figure 97 shows a copy of the JCL that creates the local request queue data set.

```
//DEFLRQDS JOB (999,POK),'CICSTS13',MSGCLASS=T,CLASS=A,
//          NOTIFY=&SYSUID
//*
//DEFINE EXEC PGM=IDCAMS,REGION=1M
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
/*          */
/* DEFINE A LOCAL REQUEST */
/* QUEUE DATASET          */
/*          */
DEFINE CLUSTER(NAME(CICSSYSF.CICS530.PAA1.DFHLRQ)-
              INDEXED-
              LOG(UNDO)-
              CYL(1 1)-
              VOLUME(TOTCI3)-
              RECORDSIZE( 2024 2400 )-
              KEYS( 40 0 )-
              FREESPACE ( 5 5 )-
              SHAREOPTIONS( 2 3 ))-
DATA (NAME(CICSSYSF.CICS530.PAA1.DFHLRQ.DATA))-
INDEX (NAME(CICSSYSF.CICS530.PAA1.DFHLRQ.INDEX))
/*
```

Figure 97. JCL to Create a Local Request Queue Data Set

The local request queue data set must be defined as a recoverable VSAM KSDS, and you have to define one per CICS TS 1.3 region. For this reason, we added the CICS SYSID as a data set qualifier, namely PAA1, in our testing environment.

## 11.3 CICS BTS Repository Data Set

The CICS BTS repository data set holds state information about the CICS BTS processes as well as application data (container contents) that is passed among activities or different reactivations of the same activity. You can define only one data set to hold information for all of your CICS BTS process types, or you can define a separate data set for each process type (see 11.1, “Process Resource Definition” on page 207), or any combination of these two options.

Figure 98 shows the JCL to create a CICS BTS repository data set.

```
//DEF#CBTS JOB (999,POK),'CICSTS13',MSGCLASS=T,CLASS=A,
//          NOTIFY=&SYSUID
//* *****
//*
//* DELETE/DEFINE THE CLUSTER FOR A CICS BTS REPOSITORY DATASET
//* NOTE:
//* THIS DS IS USED TO HARDEN PROCESS STATE DATA
//* IT CAN HOLD MORE THAN 1 PROCESSTYP, THEREFORE IT MIGHT
//* BE A GOOD IDEA TO CHOOSE A NAME THAT IS NOT RELEASE OR
//* SYSTEM DEPENDENT, BUT RATHER CATEGORIZES A CERTAIN PROCESS
//* TYPE.
//* *****
//DEFINE EXEC PGM=IDCAMS,REGION=1M
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
/* */
/* DEFINE A CICS BTS PROCESS REPOSITORY DATASET */
/* FOR PROCESS TYPE SALE */
/* */
DEFINE CLUSTER (NAME (CICSSYSF.CBTS.PTYP SALE) -
              LOG(UNDO) -
              CYL(2 1) -
              CISZ(4096) -
              SPANNED -
              VOLUMES(TOTCI4) -
              STORCLAS(CICSRLS) -
              KEYS(50 0) -
              INDEXED -
              RECORDSIZE(8192 16384) -
              FREESPACE( 5 5 ) -
              SHAREOPTIONS( 2 3 ) -
              ) -
DATA (NAME (CICSSYSF.CBTS.PTYP SALE.DATA) -
      ) -
INDEX (NAME (CICSSYSF.CBTS.PTYP SALE.INDEX) -
       )
/*
```

Figure 98. JCL to Create a CICS BTS Repository Data Set

Note that we defined the data set in an SMS storage class for RLS data sets. If you are planning to execute CICS BTS activities that belong to the same process, and you want to distribute these activities to several CICS regions, you must access the repository in RLS mode. This is true even if you run all of the CICS regions in the same MVS image. CICS BTS does not support remote file access to

a file owning region (FOR). An attempt for such an access fails with an AEZX abend. When you use CICS BTS from a single region, RLS is not required.

Next, you have to define the file control resource definition for the CICS BTS repository data set to CICS. Figure 99 shows the file control resource definition we need for the SALES sample application.

```

OBJECT CHARACTERISTICS                                CICS RELEASE = 0530
CEDA View File( PTYPSALE )
  File          : PTYPSALE
  Group         : CBTS
  DEscription   : CBTS REPOSITORY FILE
VSAM PARAMETERS
  DSName        : CICSSYSF.CBTS.PTYPSALE
  Password      :                               PASSWORD NOT SPECIFIED
  RLSaccess     : Yes                           Yes | No
  LSRpoolid    : 1                             1-8 | None
  READInteg    : Consistent                     Uncommitted | Consistent | Repeatable
  DSNSharing    : Allreqs                       Allreqs | Modifyreqs
  STRings      : 001                            1-255
  Nsrgr        :
REMOTE ATTRIBUTES
  REMOTESystem :
  REMOTENAME   :
REMOTE AND CFDATATABLE PARAMETERS
  RECORDSize   :                               1-32767
  Keylength    :                               1-255 (1-16 For CF Datatable)
INITIAL STATUS
  STatus       : Enabled                       Enabled | Disabled | Unenabled
  Opertime     : Firstref                      Firstref | Startup
  Disposition  : Share                        Share | Old
BUFFERS
  Databuffers  : 00002                         2-32767
  Indexbuffers : 00001                         1-32767
DATATABLE PARAMETERS
  TABLE       : No                           No | Cics | User | CF
  Maxnumrecs   : Nolimit                      Nolimit | 1-99999999
CFDATATABLE PARAMETERS
  Cfdtpool    :
  TABLEName  :
  UPDATEModel : Locking                       Contention | Locking
  LQad        : No                            No | Yes
DATA FORMAT
  RECORDFormat : V                            V | F
OPERATIONS
  Add          : Yes                           No | Yes
  BRrowse     : Yes                           No | Yes
  DELete      : Yes                           No | Yes
  READ        : Yes                           Yes | No
  UPDATE      : Yes                           No | Yes
AUTO JOURNALLING
  JJournal    : No                            No | 1-99
  JNLRead     : None                          None | Updateonly | Readonly | All
  JNLSYNRead  : No                            No | Yes
  JNLUpdate   : No                            No | Yes
  JNLAdd      : None                          None | Before | AFter | All
  JNLSYNWrite : Yes                           Yes | No
RECOVERY PARAMETERS
  RECOVery    : Backoutonly                   None | Backoutonly | All
  Fwdrecovlog : No                            No | 1-99
  BAcuptype   : Static                        Static | Dynamic
SECURITY
  RESsecnum   : 00                            0-24 | Public

```

Figure 99. File Control Resource Definition for the CICS BTS Repository Data Set

In our environment, we specify number of strings as 1, and RLsaccess equal Yes. In a non-RLS environment, the string number is recommended to be set to at least 10 to avoid a string wait problem on the CICS BTS repository data set.

If you intend to open the repository data set in RLS mode, you should define the read integrity to be consistent because access from different activities of a process to the same record must be serialized. It would be fatal if, for example, activity 1 updated a record while activity 2, which runs in parallel, reads the same record and receives old information (for example, event X fired / not fired).

The ADD, BRowse, DELeTe, READ and UPDATE attributes of OPERATIONS parameters have to be set to the YES value.

In a real production environment, it is probably also a good idea to define the LOG parameter of the IDCAMS repository data set definition as ALL and specify a forward recovery log, because the repository holds vital data for the process including application data (your containers).

---

## 11.4 Distributed Routing Program

If you plan to balance CICS BTS activities among several CICS regions, you need logic to provide the distribution. CICS TS 1.3 provides a new peer-to-peer routing mechanism that enables workload management of CICS BTS processes and activities. For further information about this new routing mechanism, refer to *CICS TS for OS/390 V1R3 Implementation Guide, SG24-5274*.

---

## 11.5 Audit Log

Optionally you can define an audit log for some or all of your CICS BTS process types. See 11.1, “Process Resource Definition” on page 207 for the corresponding attributes in the process type resource definition. CICS BTS provides audit points much like CICS provides trace points. However, the audit log entries for the same process can come from different CICS regions when you balance the workload of your processes in a Parallel Sysplex. Therefore, these entries include the CICS system ID in each record. The purpose of having an audit log is, for example, accounting or tracking processes that are stuck. Stuck processes typically indicate an application design problem, however, and usually should not occur.

### 11.5.1 Sample JCL to Update The Logger Policy for The Audit Log

The audit log is written to an MVS log stream, therefore, you need a log stream definition in your logger policy for the audit log. Figure 100 on page 212 shows the JCL we used to update our logger policy. The JCL defines a model log stream that will be used to dynamically create a log stream referenced in the JOURNALMODEL.

```

//DEFALOG JOB (999,POK),'CICSTS3',CLASS=A,MSGCLASS=T,
//          NOTIFY=&SYSUID
//*
//* JOB TO ADD LOGSTREAM DEFS FOR CICS BTS AUDIT LOG
//*
//* ONCE THIS IS DONE YOU SHOULD CEDA DEFINE A JOURNALMODEL
//* NAME(YOURNAME) LOGSTREAM=CICSTS13.CBTS.XNAME
//*          SAMPLE IN GROUP(CBTS) IS CICSTS13.CBTS.CBTSLOG
//*
//IXCMIAPU EXEC PGM=IXCMIAPU
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
    DATA TYPE(LOGR) REPORT(YES)
    DEFINE LOGSTREAM
        NAME(CICSTS13.CBTS.MODEL)
        STRUCTNAME(LOG_GENERAL_001)
        STG_DUPLEX(NO)
        LS_SIZE(2000)
        HLQ(CICS)
        HIGHOFFLOAD(60)
        LOWOFFLOAD(10)
        MODEL(YES)
/*

```

Figure 100. JCL to Update the Logger Policy for the Audit Log

Note that we define the log stream to be written to a coupling facility structure (LOG\_GENERAL\_001) that has already been defined. If you do not have an existing structure in the coupling facility, you have to update your CFRM policy as well to include a structure with a matching name.

If the CICS regions are in the same MVS image, you can define the log stream to use either a coupling facility structure or DASD-only logging. However, if the CICS regions are in different MVS images, the log stream should use a coupling facility structure rather than DASD-only logging. This is because CICS regions in different MVS images cannot access the same DASD-only log stream at the same time.

If the regions are in different MVS images, and you use DASD-only logging, you will not be able to use shared log streams for your BTS logs. This means that audit records for a single process may be split across several log streams; you will have to collate them yourself.

For more information about logger policy definitions, see *Setting Up a Sysplex, GC28-1799*.

Having defined the log stream, you now have to define a corresponding journal model in CICS. Figure 101 on page 213 shows the journal model for our sample application.

```
OBJECT CHARACTERISTICS                                CICS RELEASE = 0530
CEDA View Journalmodel( CBTSLOG )
Journalmodel   : CBTSLOG
Group         : CBTS
Description   : AUDIT LOG FOR CBTS PROCESSES
Journalname   : CBTSLOG
Type         : Mvs                Mvs | Smf | Dummy
Streamname    : CICSTS13.CBTS.CBTSLOG
```

Figure 101. Journal Model Resource Definition for the Audit Log

The Streamname we chose is rather generic and not specific for a CICS region or a process type. This way we could write audit logs from any process running in any CICS region to the same log stream. You may, of course, provide a more granular naming convention if you need separate audit logs and want to avoid any contention caused by the serialization when writing the log streams.

### 11.5.2 Printing The Audit Log Contents

CICS TS 1.3 provides a utility, DFHATUP, to print the contents of the audit log. It allows you to filter selected records. It formats the records to make them easier to interpret. This utility was used to print the audit logs shown in Figure 55 on page 98 and Figure 104 on page 223.

Refer to 13.1.2, “Sample JCL to Print the Audit Log Contents” on page 221 for a sample JCL.





---

## Chapter 12. Problem Determination in BTS Applications

This chapter suggests possible solutions for some of the more common problems that you might encounter when you run a CICS BTS application. The most common problem is a so-called *stuck process*.

A process is said to be stuck when it cannot proceed because it is waiting for an event that cannot, or does not, occur. These are some possible causes:

- A faulty application design.
- An activity cannot start on a remote system because the CICS region that has to run the activity is not running.

---

### 12.1 Application Design Errors

A stuck process may be caused by a program logic error. For example, consider the following scenarios:

1. Outstanding user events:

- The activity issues an EXEC CICS RETURN command without the ENDACTIVITY option in its final activation.
- There are no events on the activity's reattachment queue, but there is a user event in its event pool.
- The activity has forgotten to delete an input event after it has been retrieved.

In a case such as this, the activity becomes dormant, and there is no way for it to be reactivated. The process is stuck.

The recommended way to prevent this from happening is to add the ENDACTIVITY option to the EXEC CICS RETURN command that ends the final activation of the activity. Coding RETURN ENDACTIVITY deletes any outstanding events other than activity completion events for child activities, which the activity must deal with properly and allows the activity to complete normally.

2. Waiting for an external interaction:

A user-related activity returns from its initial activation and becomes dormant waiting for an external interaction to occur. However, the expected user input does not arrive. The process is stuck.

The recommended way to prevent this is to set a timer which, if the expected external interaction does not occur within a specified period, will cause the activity (or its parent) to be reactivated anyway.

3. Timer error:

A programming error results in a timer being set to expire in five days rather than five minutes. The process is stuck. It can also happen if there is a business rule change to handle a special VIP customer urgent request, and it will take some time to change the program to handle this situation.

**Note:** To force a timer to expire before its specified time, use the FORCE TIMER command.

---

## 12.2 Restarting Stuck Processes

There are several possible way to restart a stuck process.

### 12.2.1 Using Activity Timers

Since a business transaction in general exists longer than the traditional CICS application transaction, it is difficult to distinguish from the outside whether the process is merely dormant as designed, or the process is stuck. The best way to restart processes that are stuck is to use timers.

It is probably unnecessary to set a timer for a simple activity that completes its processing in one activation, has no children, and is to be run synchronously. On the other hand, you might want to set a timer for an activity that is long-lived, has multiple activations, running asynchronously, or involving external interaction.

### 12.2.2 Using Process Timers

You can set a timer for root activity with an expiry time some time after the whole process could reasonably be expected to have completed.

If the process is short-lived, you may decide not to set any activity timers, but to set a process timer instead. If the process is long-lived, do not set a process timer without also setting timers for at least some individual activities. This prevents the possibility of a delay in restarting the process. For example, if a process that is expected to last six months becomes stuck after one day while processing its first activity, and you have set only a process timer, the process could remain dormant for, say, seven months before the root activity is reactivated to deal with the problem.

If the root activity is activated by the process timer, it could, for example:

1. Browse and inquire on each of its descendant activities, checking completion status and mode.
2. If it succeeds in identifying the stuck activity, issue a CANCEL command to cancel it. (If the stuck activity is not a child but a lower-level descendant of the root activity, the root must first acquire the stuck activity.)
3. The stuck activity's completion event fires, causing the parent activity to be reactivated. The CHECK ACTIVITY command issued by the parent returns a completion status of FORCED. The parent should be coded to handle the abnormal completion of one of its children. The process is no longer stuck.

### 12.2.3 Using Status Containers

To make it easier for a root activity to identify which of its descendant activities are stuck, you could use status containers. Status containers are simply data-containers that contain information about what an activity is currently doing. A status container might, for instance, contain the date and time, and a string describing the work that the activity has just started or ended or the fact that it is dormant.

You can think of an activity as a finite state machine. It will always be in one of a limited number of processing states. Each activity could regularly update its status container with its current processing state.

## 12.2.4 Using A Utility Program

You can use two CICS-supplied utility programs to determine whether the process is stuck or merely dormant.

### 12.2.4.1 CICS-Supplied Utility Programs

These utilities are DFHATUP and DFHBARUP, and we cover them in more detail in Chapter 13, “CICS Business Transaction Services Utilities” on page 219.

- The audit trail utility, DFHATUP

You can use DFHATUP to print selected audit records from a log stream. If you use auditing to track the progress of your processes across the Parallel Sysplex or to investigate a stuck process, you could print its audit records.

- The repository utility, DFHBARUP

You can use DFHBARUP to print selected records from a repository. To investigate a stuck process, you could print its repository records.

### 12.2.4.2 User-Written Utility Programs

You could write a utility program that could check for and restart stuck processes, particularly if your activities use status containers. Your utility program could, for example:

1. Browse all processes of a specified process type.
2. Browse the descendant activities of each process returned in step 1.
3. Inquire on the status data-container of each activity and retrieve its contents.
4. Identify a stuck activity from the contents of its status container.
5. Issue an ACQUIRE command to acquire the stuck activity.
6. Issue a CANCEL command to cancel the stuck activity. The latter's completion event fires, causing its parent to be reactivated. The CHECK ACTIVITY command issued by the parent returns a completion status of FORCED. The parent should be coded to handle the abnormal completion of one of its children. The process is no longer stuck.

---

## 12.3 Activity Abends

If a program that implements an activity abends, the activity's parent receives control. If the failed activity was run asynchronously, the parent is reactivated. The CHECK ACTIVITY command issued by the parent returns a COMPSTATUS of ABEND.

Your application should be coded to deal with an activity abend. The parent of the failed activity might, for example, choose to do either of the following:

- Retry the failed activity
- Compensate the children of the failed activity



---

## Chapter 13. CICS Business Transaction Services Utilities

In this chapter, we describe CICS BTS utilities available for problem determination and operational aspects of CICS BTS implementation.

---

### 13.1 Audit Log

You may want to create an audit log to track the progress of complex business transactions or diagnose problems in programs that are being developed to form a new business application.

Though CICS BTS provides audit points much like CICS provides trace points, there are three main differences between them. An audit log of a process:

- Is written to a CICS journal, which resides on an MVS log stream.
- Can extend to days, weeks, or even months.
- May include records generated from CICS regions running in different MVS images in a Parallel Sysplex. Therefore, each audit record contains system, date, and time information. Because log streams can be shared by more than one region, it is possible to write audit records from different regions to the same log.

There are four incremental auditing levels:

1. None
2. Process-level
3. Activity-level
4. Full

Audit log records are written to an MVS log stream by the CICS Log Manager. You can read the records offline using the CICS audit log utility program, DFHATUP. DFHATUP allows you to filter records for specific process types, processes, and activities and interpret records into a readable format.

Audit records are buffered; they are written to the log stream only when the buffer is full or when a syncpoint occurs. This means that, when multiple CICS regions share the same log stream, audit records may not be in the exact date and time order.

#### 13.1.1 Specifying the Level of Audit Logging

You control the amount of audit logging that CICS performs for each process using the AUDITLOG and AUDITLEVEL attributes of the PROCESSTYPE definition.

The AUDITLEVEL option of the PROCESSTYPE definition allows you to specify one of four logging levels for processes of the defined type:

- ACTIVITY  
The log contains the process and activity primary audit points.
- FULL  
The log contains the process, activity primary, and secondary audit points.
- OFF

The log does not contain any audit log records for the specified process type.

- PROCESS

The log contains the process audit points only.

A process audit point contains the records whenever a process of this type:

- Is defined
- Is requested to run
- Is requested to link
- Is acquired
- Completes
- Is reset
- Is canceled
- Is suspended
- Is resumed

And:

- Each time data is placed in a process container belonging to a process of this type
- Each time a process container belonging to a process of this type is deleted
- Each time a root activity (DFHROOT) of this type of process is activated

An activity primary audit point contains the records every time a non-root activity belonging to a process of this type:

- Is requested to link
- Is activated
- Completes

An activity secondary audit point contains the records every time a non-root activity belonging to a process of this type:

- Is defined
- Is requested to run
- Is acquired
- Is reset
- Is canceled
- Is suspended
- Is resumed
- Is deleted

**Note:** If you specify any value for AUDITLEVEL other than OFF, you must also specify the AUDITLOG option of the PROCESSTYPE definition.

To reset the AUDITLEVEL attribute of an installed PROCESSTYPE definition, use the CEMT SET PROCESSTYPE command. Changes are preserved across a restart of CICS. Note that changes to an installed PROCESSTYPE definition have no effect on existing processes. It is because when a process is first defined, CICS BTS copies the information about the installed PROCESSTYPE definition into the process record.

**Note:** Full auditing has an adverse effect on performance. It is intended to provide the maximum amount of information to help track down problems when applications are being developed. It is not intended to be used on production systems.

## 13.1.2 Sample JCL to Print the Audit Log Contents

You should run DFHATUP as a batch job against a log stream that is not in use by any CICS region. If you run it against a log stream that is connected to CICS, DFHATUP will not find the records that CICS has in its buffers.

DFHATUP ignores the records that it does not recognize as BTS audit records.

Figure 102 shows the sample JCL we used to print the audit log.

```
//JATUP JOB (999,POK), 'CICSTS13',MSGCLASS=T,CLASS=A,
//          NOTIFY=&SYSUID
//*****-----*****
//***** DFHATUP - CICS BTS AUDIT LOG UTILITY PROGRAM *****
//*****-----*****
//ATUP EXEC PGM=DFHATUP,PARM='N(EN),P(60),T(M) '
//STEPLIB DD DISP=SHR,DSN=CICSTS13.CICS.SDFHLOAD
//SYSPRINT DD SYSOUT=*,DCB=RECFM=FBA          <-- OUTPUT
//AUDITLOG DD DSN=CICSTS13.CBTS.CBTSLOG,      <-- LS NAME
//          SUBSYS=(LOGR,DFHLGCVN),
//          DCB=BLKSIZE=32760
//SYSIN DD *
        AUDITLOG(SALESLOG)
        PTYPE(SALES)
        PROCESS(SALES1234567890)
/*
```

Figure 102. Sample JCL to Print the Audit Log Contents

Notice the subsystem exit program name, DFHLGCVN, which is different from the name known from earlier releases of CICS TS (DFHLG520, for example).

### 13.1.2.1 EXEC Parameters

You can use the PARM keyword on the EXEC statement to pass one or more of the following parameters to the DFHATUP utility.

- NATLANG({EN|CS|KA})

The language in which messages are to be issued.

The minimum abbreviation of this parameter is **N**. The possible values are:

<b>CS</b>	Traditional Chinese
<b>EN</b>	English (this is the default)
<b>KA</b>	Kanji

- PAGESIZE({60|nn})

The number of lines to be printed per page when the output from the utility is sent to a printer. Valid values are in the range 20 - 99. The default is 60.

The minimum abbreviation of this parameter is **P**.

- TRANSLATE({MIXEDCASE|UPPERCASE})

Whether the output from the utility is to be in mixed-case or uppercase. The default is mixed-case.

The minimum abbreviation of this parameter is **T**. The minimum abbreviations of MIXEDCASE and UPPERCASE are M and U, respectively.

### 13.1.2.2 SYSIN Control Statements

The SYSIN data set is used to pass information to DFHATUP.

The format of the SYSIN control statement is:

```
SYSIN DD *
  AUDITLOG(name)
  PTYPE(name) <PROCESS(name)>
  PROCESS(name)
  ACTIVITY(name)
```

An AUDITLOG statement cannot contain additional arguments. Other statements may consist of multiple arguments.

### 13.1.3 Example Output from the DFHATUP Utility

The example control statements in Figure 103 would format all the records written to the audit log for the SALES1234567890 process (which is of the SALES process type).

```
//SALESLOG DD DSN=CICSAA#.CICSDC1.JRNL001,
//          SUBSYS=(LOGR,DFHLGCNV),
//          DCB=BLKSIZE=32760
//SYSIN    DD *
AUDITLOG(SALESLOG)
PTYPE(SALES)
PROCESS(SALES1234567890)
/*
//*
```

Figure 103. Example Control Statements to Format All the Records for the SALES1234567890



### 13.1.3.1 Example Audit Log - Full Level Auditing

Figure 104 shows an excerpt of the audit log printout. You can see the beginning of our tests when process 1234567890 is invoked by the DEFINE PROCESS - API in program DFH0SAL1. The auditlog entries show the GMT time.

```

CBTS Audit Trail Utility - Parameter Validation                               Date : 29/01/1999 Time : 14:38:25 Page 000001
Exec Parm Options: Natlang (EN)
                      Translate (mixedcase)
                      Pagesize (60)

CBTS Audit Trail Utility - Audit Print                                   Date : 29/01/1999 Time : 14:38:25 Page 000002
Ptype(SALES ) Function(Define Process ) Process(SALES1234567890          ) System(FX2 ) Auditlog(BAMAUDIT)
Taskno(0000033) Activity(DFHROOT ) Transid(R ) Program(R ) Userid(CICSUSER)
                      ActivityId(BAMFILE1..GBIBMIYA.IYCWTC39....v..DFHROOT )
                      (CCDCDCF11CCCCDCEC4CECEECFFB2902A00CCDDDE44444444)
                      (21469351A172924981B98363339A709F5014689663000000000)

Current: Transid(P ) Program(P ) Userid(CICSUSER) Date(1999.029) Time(14:36:12.557162)
Ptype(SALES ) Function(Run Process ) Process(SALES1234567890          ) System(FX2 ) Auditlog(BAMAUDIT)
Taskno(0000033) Activity(DFHROOT ) Asynchronous
                      ActivityId(BAMFILE1..GBIBMIYA.IYCWTC39....v..DFHROOT )
                      (CCDCDCF11CCCCDCEC4CECEECFFB2902A00CCDDDE44444444)
                      (21469351A172924981B98363339A709F5014689663000000000)

Current: Transid(P ) Program(P ) Userid(CICSUSER) Date(1999.029) Time(14:36:13.921790)
Ptype(SALES ) Function(Activation ) Process(SALES1234567890          ) System(FX2 ) Auditlog(BAMAUDIT)
Taskno(0000034) Activity(DFHROOT ) Event(DFHINITIAL )
                      ActivityId(BAMFILE1..GBIBMIYA.IYCWTC39....v..DFHROOT )
                      (CCDCDCF11CCCCDCEC4CECEECFFB2902A00CCDDDE44444444)
                      (21469351A172924981B98363339A709F5014689663000000000)

Current: Transid(R ) Program(R ) Userid(CICSUSER) Date(1999.029) Time(14:36:14.142640)
Ptype(SALES ) Function(Define Activity ) Process(SALES1234567890        ) System(FX2 ) Auditlog(BAMAUDIT)
Taskno(0000034) Activity(B ) CompletionEvent(B ) Transid(B ) Program(B ) Userid(CICSUSER)
                      ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..j.....B )
                      (CCDCDCF11CCCCDCEC4CEDFECEFB29B0300C44444444444444)
                      (21469351A172924981B98229672A7111C012000000000000000)

Current: Transid(R ) Program(R ) Userid(CICSUSER) Date(1999.029) Time(14:36:14.295419)
Ptype(SALES ) Function(Run Activity ) Process(SALES1234567890          ) System(FX2 ) Auditlog(BAMAUDIT)
Taskno(0000034) Activity(B ) Synchronous
                      ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..j.....B )
                      (CCDCDCF11CCCCDCEC4CEDFECEFB29B0300C44444444444444)
                      (21469351A172924981B98229672A7111C012000000000000000)

Current: Transid(R ) Program(R ) Userid(CICSUSER) Date(1999.029) Time(14:36:14.295549)
Ptype(SALES ) Function(Activation ) Process(SALES1234567890          ) System(FX2 ) Auditlog(BAMAUDIT)
Taskno(0000035) Activity(B ) Event(DFHINITIAL )
                      ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..j.....B )
                      (CCDCDCF11CCCCDCEC4CEDFECEFB29B0300C44444444444444)
                      (21469351A172924981B98229672A7111C012000000000000000)

Current: Transid(B ) Program(B ) Userid(CICSUSER) Date(1999.029) Time(14:36:14.296323)
Ptype(SALES ) Function(Completion ) Process(SALES1234567890          ) System(FX2 ) Auditlog(BAMAUDIT)
Taskno(0000035) Activity(B ) Compstatus(Normal )
                      ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..j.....B )
                      (CCDCDCF11CCCCDCEC4CEDFECEFB29B0300C44444444444444)
                      (21469351A172924981B98229672A7111C012000000000000000)

Current: Transid(B ) Userid(CICSUSER) Date(1999.029) Time(14:36:14.408739)
Ptype(SALES ) Function(Define Activity ) Process(SALES1234567890        ) System(FX2 ) Auditlog(BAMAUDIT)
Taskno(0000034) Activity(C ) CompletionEvent(C ) Transid(C ) Program(C ) Userid(CICSUSER)
                      ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..j..6...C )
                      (CCDCDCF11CCCCDCEC4CEDFECEFB29DF500C44444444444444)
                      (21469351A172924981B98229672A71C69013000000000000000)

Current: Transid(R ) Program(R ) Userid(CICSUSER) Date(1999.029) Time(14:36:14.472960)
CBTS Audit Trail Utility - Audit Print                                   Date : 29/01/1999 Time : 14:38:25 Page 000003
Ptype(SALES ) Function(Run Activity ) Process(SALES1234567890          ) System(FX2 ) Auditlog(BAMAUDIT)
Taskno(0000034) Activity(C ) Asynchronous
                      ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..j..6...C )
                      (CCDCDCF11CCCCDCEC4CEDFECEFB29DF500C44444444444444)
                      (21469351A172924981B98229672A71C69013000000000000000)

Current: Transid(R ) Program(R ) Userid(CICSUSER) Date(1999.029) Time(14:36:14.473066)
Ptype(SALES ) Function(Define Activity ) Process(SALES1234567890        ) System(FX2 ) Auditlog(BAMAUDIT)
Taskno(0000034) Activity(D ) CompletionEvent(D ) Transid(D ) Program(D ) Userid(CICSUSER)
                      ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..j..9..D )
                      (CCDCDCF11CCCCDCEC4CEDFECEFB29DF00C44444444444444)
                      (21469351A172924981B98229672A71EE9014000000000000000)

```

Figure 104 (Part 1 of 4). Sample Audit Log Printout

```

Current: Transid(R ) Program(R ) Userid(CICSUSER) Date(1999.029) Time(14:36:14.482228) Activity(DFHROOT )
Ptype(SALES ) Function(Run Activity ) Process(SALES1234567890 ) System(FX2 ) Auditlog(BAMAUDIT)
Taskno(0000034) Activity(D ) Asynchronous
ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..j..9..D )
(CDCCDC11CCCCDCEC4CEDFECEFB29DBF00C44444444444444)
(21469351A172924981B98229672A71EE901400000000000000)

Current: Transid(R ) Program(R ) Userid(CICSUSER) Date(1999.029) Time(14:36:14.482346) Activity(DFHROOT )
Ptype(SALES ) Function(Activation ) Process(SALES1234567890 ) System(FX2 ) Auditlog(BAMAUDIT)
Taskno(0000036) Activity(C ) Event(DFHINITIAL )
ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..j..6...C )
(CDCCDC11CCCCDCEC4CEDFECEFB29DF500C44444444444444)
(21469351A172924981B98229672A71C69013000000000000000)

Current: Transid(C ) Program(C ) Userid(CICSUSER) Date(1999.029) Time(14:36:14.556761)
Ptype(SALES ) Function(Activation ) Process(SALES1234567890 ) System(FX2 ) Auditlog(BAMAUDIT)
Taskno(0000037) Activity(D ) Event(DFHINITIAL )
ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..j..9..D )
(CDCCDC11CCCCDCEC4CEDFECEFB29DBF00C44444444444444)
(21469351A172924981B98229672A71EE901400000000000000)

Current: Transid(D ) Program(D ) Userid(CICSUSER) Date(1999.029) Time(14:36:14.569775)
Ptype(SALES ) Function(Define Activity ) Process(SALES1234567890 ) System(FX2 ) Auditlog(BAMAUDIT)
Taskno(0000036) Activity(E ) CompletionEvent(E ) Transid(E ) Program(E ) Userid(CICSUSER)
ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..k..o..E )
(CDCCDC11CCCCDCEC4CEDFECEFB2906900C44444444444444)
(21469351A172924981B98229672A7294601500000000000000)

Current: Transid(C ) Program(C ) Userid(CICSUSER) Date(1999.029) Time(14:36:14.656929) Activity(C )
Ptype(SALES ) Function(Run Activity ) Process(SALES1234567890 ) System(FX2 ) Auditlog(BAMAUDIT)
Taskno(0000036) Activity(E ) Asynchronous
ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..k..o..E )
(CDCCDC11CCCCDCEC4CEDFECEFB2906900C44444444444444)
(21469351A172924981B98229672A7294601500000000000000)

Current: Transid(C ) Program(C ) Userid(CICSUSER) Date(1999.029) Time(14:36:14.657049) Activity(C )
Ptype(SALES ) Function(Define Activity ) Process(SALES1234567890 ) System(FX2 ) Auditlog(BAMAUDIT)
Taskno(0000036) Activity(F ) CompletionEvent(F ) Transid(F ) Program(F ) Userid(CICSUSER)
ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..k..I..F )
(CDCCDC11CCCCDCEC4CEDFECEFB290AC00C44444444444444)
(21469351A172924981B98229672A72BA9016000000000000000)

Current: Transid(C ) Program(C ) Userid(CICSUSER) Date(1999.029) Time(14:36:14.668485) Activity(C )
CBTS Audit Trail Utility - Audit Print Date : 29/01/1999 Time : 14:38:25 Page 000004
Ptype(SALES ) Function(Run Activity ) Process(SALES1234567890 ) System(FX2 ) Auditlog(BAMAUDIT)
Taskno(0000036) Activity(F ) Asynchronous
ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..k..I..F )
(CDCCDC11CCCCDCEC4CEDFECEFB290AC00C44444444444444)
(21469351A172924981B98229672A72BA9016000000000000000)

Current: Transid(C ) Program(C ) Userid(CICSUSER) Date(1999.029) Time(14:36:14.668584) Activity(C )
Ptype(SALES ) Function(Activation ) Process(SALES1234567890 ) System(FX2 ) Auditlog(BAMAUDIT)
Taskno(0000038) Activity(E ) Event(DFHINITIAL )
ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..k..o..E )
(CDCCDC11CCCCDCEC4CEDFECEFB2906900C44444444444444)
(21469351A172924981B98229672A7294601500000000000000)

Current: Transid(E ) Program(E ) Userid(CICSUSER) Date(1999.029) Time(14:36:14.757748)
Ptype(SALES ) Function(Activation ) Process(SALES1234567890 ) System(FX2 ) Auditlog(BAMAUDIT)
Taskno(0000039) Activity(F ) Event(DFHINITIAL )
ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..k..I..F )
(CDCCDC11CCCCDCEC4CEDFECEFB290AC00C44444444444444)
(21469351A172924981B98229672A72BA9016000000000000000)

Current: Transid(F ) Program(F ) Userid(CICSUSER) Date(1999.029) Time(14:36:14.790932)
Ptype(SALES ) Function(Define Activity ) Process(SALES1234567890 ) System(FX2 ) Auditlog(BAMAUDIT)
Taskno(0000037) Activity(G ) CompletionEvent(G ) Transid(G ) Program(G ) Userid(CICSUSER)
ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..k....G )
(CDCCDC11CCCCDCEC4CEDFECEFB2925000C44444444444444)
(21469351A172924981B98229672A72F75017000000000000000)

Current: Transid(D ) Program(D ) Userid(CICSUSER) Date(1999.029) Time(14:36:14.811252) Activity(D )
Ptype(SALES ) Function(Run Activity ) Process(SALES1234567890 ) System(FX2 ) Auditlog(BAMAUDIT)
Taskno(0000037) Activity(G ) Asynchronous
ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..k....G )
(CDCCDC11CCCCDCEC4CEDFECEFB2925000C44444444444444)
(21469351A172924981B98229672A72F75017000000000000000)

```

Figure 104 (Part 2 of 4). Sample Audit Log Printout

```

Current: Transid(D ) Program(D ) Userid(CICSUSER) Date(1999.029) Time(14:36:14.811377) Activity(D )
Ptype(SALES ) Function(Activation ) Process(SALES1234567890 ) System(FX2 ) Auditlog(BAMAUDIT)
Taskno(0000040) Activity(G ) Event(DFHINITIAL )
ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..k....G )
(CCDCCDCF11CCCDCEC4CEDFECEFB2925000C44444444444444)
(21469351A172924981B98229672A72F75017000000000000000)

Current: Transid(G ) Program(G ) Userid(CICSUSER) Date(1999.029) Time(14:36:14.844281)
Ptype(SALES ) Function(Completion ) Process(SALES1234567890 ) System(FX2 ) Auditlog(BAMAUDIT)
Taskno(0000039) Activity(F ) Compstatus(Normal )
ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..k..I..F )
(CCDCCDCF11CCCDCEC4CEDFECEFB290AC00C444444444444444)
(21469351A172924981B98229672A72BA90160000000000000000)

Current: Transid(F ) Userid(CICSUSER) Date(1999.029) Time(14:36:14.887329)
Ptype(SALES ) Function(Activation ) Process(SALES1234567890 ) System(FX2 ) Auditlog(BAMAUDIT)
Taskno(0000041) Activity(C ) Event(F )
ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..j..6...C )
(CCDCCDCF11CCCDCEC4CEDFECEFB29DF500C444444444444444)
(21469351A172924981B98229672A71C690130000000000000000)

Current: Transid(C ) Program(C ) Userid(CICSUSER) Date(1999.029) Time(14:36:14.979781)
CBTS Audit Trail Utility - Audit Print Date : 29/01/1999 Time : 14:38:25 Page 000005
Ptype(SALES ) Function(Completion ) Process(SALES1234567890 ) System(FX2 ) Auditlog(BAMAUDIT)
Taskno(0000038) Activity(E ) Compstatus(Normal )
ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..k..o..E )
(CCDCCDCF11CCCDCEC4CEDFECEFB2906900C444444444444444)
(21469351A172924981B98229672A729460150000000000000000)

Current: Transid(E ) Userid(CICSUSER) Date(1999.029) Time(14:36:15.070372)
Ptype(SALES ) Function(Activation ) Process(SALES1234567890 ) System(FX2 ) Auditlog(BAMAUDIT)
Taskno(0000042) Activity(C ) Event(E )
ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..j..6...C )
(CCDCCDCF11CCCDCEC4CEDFECEFB29DF500C444444444444444)
(21469351A172924981B98229672A71C690130000000000000000)

Current: Transid(C ) Program(C ) Userid(CICSUSER) Date(1999.029) Time(14:36:15.117121)
Ptype(SALES ) Function(Completion ) Process(SALES1234567890 ) System(FX2 ) Auditlog(BAMAUDIT)
Taskno(0000042) Activity(C ) Compstatus(Normal )
ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..j..6...C )
(CCDCCDCF11CCCDCEC4CEDFECEFB29DF500C444444444444444)
(21469351A172924981B98229672A71C690130000000000000000)

Current: Transid(C ) Userid(CICSUSER) Date(1999.029) Time(14:36:15.135971)
Ptype(SALES ) Function(Activation ) Process(SALES1234567890 ) System(FX2 ) Auditlog(BAMAUDIT)
Taskno(0000043) Activity(DFHROOT ) Event(C )
ActivityId(BAMFILE1..GBIBMIYA.IYCWC39.....v..DFHROOT )
(CCDCCDCF11CCCDCEC4CECECFB2902A00CCDDDE4444444444444)
(21469351A172924981B98363339A709F501468966300000000000)

Current: Transid(R ) Program(R ) Userid(CICSUSER) Date(1999.029) Time(14:36:15.169265)
Ptype(SALES ) Function(Acquire ActId ) Process(SALES1234567890 ) System(FX2 ) Auditlog(BAMAUDIT)
Taskno(0000045) Activity(G )
ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..k....G )
(CCDCCDCF11CCCDCEC4CEDFECEFB2925000C444444444444444)
(21469351A172924981B98229672A72F750170000000000000000)

Current: Transid(I ) Program(I ) Userid(CICSUSER) Date(1999.029) Time(14:36:21.922942)
Ptype(SALES ) Function(Cancel Activity ) Process(SALES1234567890 ) System(FX2 ) Auditlog(BAMAUDIT)
Taskno(0000045) Activity(G )
ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..k....G )
(CCDCCDCF11CCCDCEC4CEDFECEFB2925000C444444444444444)
(21469351A172924981B98229672A72F750170000000000000000)

Current: Transid(I ) Program(I ) Userid(CICSUSER) Date(1999.029) Time(14:36:21.923045)
Ptype(SALES ) Function(Completion ) Process(SALES1234567890 ) System(FX2 ) Auditlog(BAMAUDIT)
Taskno(0000045) Activity(G ) Compstatus(Forced )
ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..k....G )
(CCDCCDCF11CCCDCEC4CEDFECEFB2925000C444444444444444)
(21469351A172924981B98229672A72F750170000000000000000)

Current: Transid(I ) Program(I ) Userid(CICSUSER) Date(1999.029) Time(14:36:21.923093)
Ptype(SALES ) Function(Activation ) Process(SALES1234567890 ) System(FX2 ) Auditlog(BAMAUDIT)
Taskno(0000046) Activity(D ) Event(G )
ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..j..9..D )
(CCDCCDCF11CCCDCEC4CEDFECEFB29DBF00C444444444444444)
(21469351A172924981B98229672A71EE90140000000000000000)

Current: Transid(D ) Program(D ) Userid(CICSUSER) Date(1999.029) Time(14:36:21.948512)

```

Figure 104 (Part 3 of 4). Sample Audit Log Printout

```

CBTS Audit Trail Utility - Audit Print                                     Date : 29/01/1999 Time : 14:38:25 Page 000006
Ptype(SALES ) Function(Define Activity ) Process(SALES1234567890      ) System(FX2 ) Auditlog(BAMAUDIT)
Taskno(0000046) Activity(H      ) CompletionEvent(H      ) Transid(H      ) Program(H      ) Userid(CICSUSER)
ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..q.X...H      )
(CDCCDCDF11CCCCDCEC4CEDFECEFB29FED00C44444444444444)
(21469351A172924981B98229672A78F7F01800000000000000)

Current: Transid(D ) Program(D      ) Userid(CICSUSER) Date(1999.029) Time(14:36:21.990993) Activity(D      )
Ptype(SALES ) Function(Run Activity ) Process(SALES1234567890      ) System(FX2 ) Auditlog(BAMAUDIT)
Taskno(0000046) Activity(H      ) Asynchronous
ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..q.X...H      )
(CDCCDCDF11CCCCDCEC4CEDFECEFB29FED00C44444444444444)
(21469351A172924981B98229672A78F7F01800000000000000)

Current: Transid(D ) Program(D      ) Userid(CICSUSER) Date(1999.029) Time(14:36:21.991119) Activity(D      )
Ptype(SALES ) Function(Activation ) Process(SALES1234567890      ) System(FX2 ) Auditlog(BAMAUDIT)
Taskno(0000047) Activity(H      ) Event(DFHINITIAL      )
ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..q.X...H      )
(CDCCDCDF11CCCCDCEC4CEDFECEFB29FED00C44444444444444)
(21469351A172924981B98229672A78F7F01800000000000000)

Current: Transid(H ) Program(H      ) Userid(CICSUSER) Date(1999.029) Time(14:36:22.052659)
Ptype(SALES ) Function(Completion ) Process(SALES1234567890      ) System(FX2 ) Auditlog(BAMAUDIT)
Taskno(0000047) Activity(H      ) Compstatus(Normal      )
ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..q.X...H      )
(CDCCDCDF11CCCCDCEC4CEDFECEFB29FED00C44444444444444)
(21469351A172924981B98229672A78F7F01800000000000000)

Current: Transid(H ) Userid(CICSUSER) Date(1999.029) Time(14:36:22.123737)
Ptype(SALES ) Function(Activation ) Process(SALES1234567890      ) System(FX2 ) Auditlog(BAMAUDIT)
Taskno(0000048) Activity(D      ) Event(H      )
ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..j..9..D      )
(CDCCDCDF11CCCCDCEC4CEDFECEFB29FED00C44444444444444)
(21469351A172924981B98229672A71EE9014000000000000000)

Current: Transid(D ) Program(D      ) Userid(CICSUSER) Date(1999.029) Time(14:36:22.147332)
Ptype(SALES ) Function(Completion ) Process(SALES1234567890      ) System(FX2 ) Auditlog(BAMAUDIT)
Taskno(0000048) Activity(D      ) Compstatus(Normal      )
ActivityId(BAMFILE1..GBIBMIYA.IYK2ZFX2..j..9..D      )
(CDCCDCDF11CCCCDCEC4CEDFECEFB29FED00C44444444444444)
(21469351A172924981B98229672A71EE9014000000000000000)

Current: Transid(D ) Userid(CICSUSER) Date(1999.029) Time(14:36:22.162148)
Ptype(SALES ) Function(Activation ) Process(SALES1234567890      ) System(FX2 ) Auditlog(BAMAUDIT)
Taskno(0000049) Activity(DFHROOT      ) Event(D      )
ActivityId(BAMFILE1..GBIBMIYA.IYCWTC39....v..DFHROOT      )
(CDCCDCDF11CCCCDCEC4CECEECFFB2902A00CCDDDE4444444444)
(21469351A172924981B98363339A709F50146896630000000000)

Current: Transid(R ) Program(R      ) Userid(CICSUSER) Date(1999.029) Time(14:36:22.185932)
Ptype(SALES ) Function(Completion ) Process(SALES1234567890      ) System(FX2 ) Auditlog(BAMAUDIT)
Taskno(0000049) Activity(DFHROOT      ) Compstatus(Normal      )
ActivityId(BAMFILE1..GBIBMIYA.IYCWTC39....v..DFHROOT      )
(CDCCDCDF11CCCCDCEC4CECEECFFB2902A00CCDDDE4444444444)
(21469351A172924981B98363339A709F50146896630000000000)

Current: Transid(R ) Userid(CICSUSER) Date(1999.029) Time(14:36:22.482472)

CBTS Audit Trail Utility - Selection Results                             Date : 29/01/1999 Time : 14:38:25 Page 000007

Number of Audit records read :          40

Number of records selected   :          40

Processing Complete

```

Figure 104 (Part 4 of 4). Sample Audit Log Printout

---

## 13.2 Repository Data Set

There may be times when you need to examine records on a repository for diagnostic purposes.

You can use the repository utility program, DFHBARUP, to print selected records from a specified repository data set.

### 13.2.1 The Repository Utility Program, DFHBARUP

By default, DFHBARUP prints all the records currently on the specified repository. Thus, you could use it to take a snapshot of your CICS BTS system at the time the utility is run. The state of a repository may change from moment to moment. For example, records for new processes and activities may be added constantly; conversely, as processes complete, and events are deleted, their associated records will disappear from the repository.

DFHBARUP allows you to filter selected records, for example, print only the records associated with a specific process. Doing so would give you the current state of:

- The activities that have been defined to the process and have not yet been deleted.
- The containers associated with the activities.
- The events in the activities' event pools.

Alternatively, you could print only the records associated with a specific activity. Doing so would give you the current state of:

- The activity itself
- The containers associated with the activity
- The events in the activity's event pool

DFHBARUP formats the records it extracts to make them easier to interpret.

### 13.2.2 Sample JCL to Print the Repository

Run DFHBARUP as a batch job.

DFHBARUP reads the records in the order they are stored on the repository, that is, in keyed-sequence order. Figure 105 on page 228 shows the sample JCL to print the repository.

```

***
//JARUP JOB (999,P0K),'CICSTS13',MSGCLASS=T,CLASS=A ***
//          NOTIFY=&SYSUID ***
//*****
//* RUN DFHBARUP (REPOSITORY UTILITY PROGRAM)
//*****
//JARUP EXEC PGM=DFHBARUP,PARM='N(EN),P(60),T(M) '
//STEPLIB DD DSN=CICSTS13.CICS.SDFHLOAD,DISP=SHR
//SYSPRINT DD SYSOUT=A,DCB=RECFM=FBA <--- OUTPUT
//REPOS DD DISP=SHR,DSN=CICSTS13.CBTS.SALESREP <--- LS NAME
//SYSIN DD *
PTYPE(SALES) +
PROCESS(CUSTSALES1999.13872977829728.QA) +
ACTIVITY(ORDER)
/*
//*

```

Figure 105. Sample JCL to Print the Repository for the ORDER Activity of the CUSTSALES1999.13872977829728.QA Process

### 13.2.2.1 EXEC Parameters

You can use the PARM keyword on the EXEC statement to pass one or more of the following parameters to the DFHBARUP utility.

- NATLANG({EN|CS|KA})

The language in which messages are to be issued.

The minimum abbreviation of this parameter is **N**. The possible values are:

<b>CS</b>	Traditional Chinese
<b>EN</b>	English (this is the default)
<b>KA</b>	Kanji

- PAGESIZE({60|nn})

The number of lines to be printed per page when the output from the utility is sent to a printer. Valid values are in the range 20 - 99. The default is 60.

The minimum abbreviation of this parameter is **P**.

- TRANSLATE({MIXEDCASE|UPPERCASE})

Whether the output from the utility is to be in mixed-case or uppercase. The default is mixed-case.

The minimum abbreviation of this parameter is **T**. The minimum abbreviations of MIXEDCASE and UPPERCASE are M and U, respectively.

### 13.2.2.2 SYSIN Control Statements

The SYSIN data set is used to pass information to DFHBARUP.

The format of the SYSIN control statements is:

```

SYSIN DD *
PTYPE(name) <PROCESS(name)>
PROCESS(name)
ACTIVITY(name)

```

A REPOSITORY statement cannot contain additional arguments. Other statements may consist of multiple arguments.

### 13.2.3 Example Output from the DFHBARUP Utility

The example control statements in Figure 106 would format all the records currently on the SALEREP repository for SALES1234567890 process (which is of the SALES process type).

```
.  
.   
//SALESREP DD DISP=SHR,DSN=CICSTS13.CBTS.SALESREP  
//SYSIN DD *  
REPOSITORY(SALESREP)  
PTYPE(SALES) +  
PROCESS(SALES1234567890)  
/*
```

Figure 106. Example Control Statements to Format All Records on the SALEREP Repository

**Note:** A DFHBARUP report shows activity identifiers in the form they are stored on the repository. Unlike the activity identifiers returned by commands, such as ASSIGN and GETNEXT ACTIVITY, those shown by DFHBARUP are not prefixed with the CICS file name of the repository.

#### 13.2.3.1 Repository Printout Example

Figure 107 on page 230 shows an excerpt of the repository printout.

CICS Business Transaction Services - Parameter Validation  
 Exec Parm Options: Natlang (EN) (Default)  
 Translate (mixedcase)  
 Pagesize (60)

Date : 20/02/1999 Time : 00:40:53 Page 000001

REPOSITORY (REPOS)  
 \*PROCESS (ABCDEFGHIJKLMNORSTUVWXYZ0123456789)  
 \*PROCESS (MORT000000001)  
 \*PTYPE (BAMSHR)  
 \*PTYPE (BAMSHRA)  
 \*PTYPE (BAMSHRF)  
 \*PTYPE (BAMSHRP)  
 \*PTYPE (ORDER)  
 \*PTYPE (BAM1)  
 \*PROCESS (MORT000000003)  
 \*PTYPE (BAMSHRP)

CICS Business Transaction Services - Repository File Report

Date : 20/02/1999 Time : 00:40:53 Page 000002

Activity Name : DFHROOT Id : ..GBIBMIYA.IGDS257L....1.. DFHROOT T Generation : 000001  
 11CCCCDCEC4CCCEFFFD4C6BF004CCDDDE444444444  
 9072924981B974225736E3710104689663000000000

Definitional Attributes

Program : FINANCE  
 Transid : PERS  
 Userid : CTSR01D  
 Comp Event :

Current State

Mode : Dormant (Initial, Active, Dormant, Cancelling, Complete)  
 Suspended : No (Yes, No)  
 Generation : 000001  
 Child Count : 000005

Completion Status

Completion Response : Incomplete

```

000000 C1401910 C7C2C9C2 D4C9E8C1 4BC9C7C4 E2F2F5F7 D346CE63 B7F10001 40C4C6C8 *A ..GBIBMIYA.IGDS257L....1.. DFH*
000020 D9D6D6E3 40404040 40404040 40400000 000000F0 00004000 00001098 01500000 *ROOT .....0.. ....q.&...
000040 6EC4C6C8 C2C1C1C3 E3C9E5C9 00000000 40404040 01500001 00000000 D740C2C1 *>DFHBAACTIVI.... .&.....P BA*
000060 D4E2C8D9 4040D7C5 D9E2F0F0 F0F0F0F0 F0F0F140 40404040 40404040 40404040 *MSHR PERS000000001 *
000080 40404040 40404040 40400000 00000000 00000000 00000000 00000000 00000000 *
0000A0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 *.....
0000C0 1910C7C2 C9C2D4C9 E8C14BC9 C7C4E2F2 F5F7D346 CE63B7F1 00014003 00000000 *..GBIBMIYA.IGDS257L....1.. ....*
0000E0 00000000 00000000 00000000 00000000 00000005 00000000 00000000 000006A8 *.....y*
000100 00000001 00000001 00000684 00000000 00000000 00000000 00000000 00000000 *.....d.....*
000120 00000000 11C5211C 11C5211C 00000000 00000000 11C52A04 11C52A04 C6C9D5C1 *.....E...E.....E...E..FINA*
000140 D5C3C540 00000000 00000000 D7C5D9E2 C3E3E2D9 F0F1C440 40404040 40404040 *NCE .....PERSCTSR01D *
000160 40404040 40404040 01404040 40404040 40404040 4000C3C2 E3E24040 40400000 * ..... .CBTS ...*
000180 00000000 00004040 40400000 0001F0F0 F0F0F0F0 F0F0F1D4 D6D9E3C3 D6D5E3F0 *..... .00000001MORTCONT0*
0001A0 F0F0F0F0 F0F0F0F1 11C7303C 80000000 11C4E104 11C4E1E4 C3C4C6C8 C9D5C9E3 *00000001.G.....D...D.UCFHINIT*
0001C0 40404040 40404040 00000001 * ..... *
```

Related BTS Objects

Process Type : BAMSHR Name : PERS000000001  
 No Parent  
 Child Name : MAKEPAY Id : ..GBIBMIYA.IYCSCTSGM..=;..MAKEPAY Generation : 0000003  
 11CCCCDCEC4CECECECD177B500DCDCDCE444444444  
 A172924981B98323327419EEE0141257180000000000  
 Child Name : NOTENDED Id : ..GBIBMIYA.IYCSCTSGL.C.9..NOTENDE D Generation : 0000001  
 11CCCCDCEC4CECECECD4FC6F00DDECCDCE444444444  
 A172924981B983233273AF3490156355454000000000  
 Child Name : HOLIDAY Id : ..GBIBMIYA.IYCSCTSGL.7..HOLIDAY Generation : 0000001  
 11CCCCDCEC4CECECECD4F3B900CDDCCCE444444444  
 A172924981B983233273A76630186394180000000000  
 Child Name : NOTIFYEND Id : ..GBIBMIYA.IYCSCTSGL...NOTIFYE ND Generation : 0000001  
 11CCCCDCEC4CECECECD4B3550DDECCDCE444444444  
 A172924981B9832332739C2A8015639685400000000  
 Child Name : MORTREQ Id : ..GBIBMIYA.IYCSCTSGL.....MORTREQ Generation : 0000001

Figure 107 (Part 1 of 3). Sample Repository Printout



CICS Business Transaction Services - Repository File Report Date : 20/02/1999 Time : 00:40:53 Page 000003  
11CCCCDCEC4CECECECD48BD300DDDED CD 4444444444  
A172924981B9832332739ACF30146939580000000000

Eventpool  
Event : DFHINITIAL  
Type : System  
Fired : No  
Reattach : Yes  
Retrieved : No  
Subevent : No  
Event : PAYMENT-DUE  
Type : Input  
Fired : No  
Reattach : Yes  
Retrieved : No  
Subevent : No  
Event : ANNUAL-STATEMENT  
Type : Input  
Fired : No  
Reattach : Yes  
Retrieved : No  
Subevent : No  
Event : NOTIFY-END  
Type : Input  
Fired : No  
Reattach : Yes  
Retrieved : No  
Subevent : No  
Event : NOTENED  
Type : Input  
Fired : No  
Reattach : Yes  
Retrieved : No  
Subevent : No  
Event : INTEREST-CHANGED  
Type : Input  
Fired : No  
Reattach : Yes  
Retrieved : No  
Subevent : No  
Event : REPOSSESS  
Type : Input  
Fired : No  
Reattach : Yes  
Retrieved : No  
Subevent : No

Figure 107 (Part 2 of 3). Sample Repository Printout

Event : TIMERS  
Type : Composite  
Fired : No  
Reattach : Yes  
Retrieved : No  
Subevent : No           Number of Subevents : 0000003           Subevents Fired : 0000000

Event : EARLY-SETTLEMENT  
Type : Timer  
Fired : No  
Reattach : No  
Retrieved : No  
Subevent : Yes  
Timer : EARLY-SETTLEMENT

Event : CHECK-BONUS  
Type : Timer  
Fired : No  
Reattach : No  
Retrieved : No  
Subevent : Yes  
Timer : CHECK-BONUS

Event : HOLIDAY  
Type : Timer  
Fired : No  
Reattach : Yes  
Retrieved : No  
Subevent : No  
Timer : TIMER-HOLIDAY

Event : NOTIFY-END-DONE  
Type : Activity  
Fired : No  
Reattach : Yes  
Retrieved : No  
Subevent : No

Event : MAKE-PAYMENT  
Type : Timer  
Fired : No  
Reattach : No  
Retrieved : No  
Subevent : Yes  
Timer : MAKE-PAYMENT

Timer : EARLY-SETTLEMENT  
Status : Expired  
Date : 18/02/1999 Time : 14:24:20  
Event : EARLY-SETTLEMENT

Timer : CHECK-BONUS  
Status : Expired  
Date : 18/02/1999 Time : 14:23:20  
Event : CHECK-BONUS

Number of Repository records read :       16

Number of records selected                :     16

Processing Complete

Figure 107 (Part 3 of 3). Sample Repository Printout

## 13.3 Operator Commands

In this section, we describe CICS BTS operator commands, mainly new CEMT options, to be used to inquire about and control CICS BTS environment. These commands are:

- CEMT INQUIRE PROCESSTYPE
- CEMT INQUIRE TASK
- CEMT SET PROCESSTYPE

### 13.3.1 CEMT INQUIRE PROCESSTYPE

You use this command to retrieve information about a CICS BTS process type.

INQUIRE PROCESSTYPE returns information about the PROCESSTYPE definitions installed on this CICS region. In particular, it shows the current state of audit logging for each displayed process type.

Press the Clear key to clear the screen. There are two ways to start this transaction:

- Type CEMT INQUIRE PROCESSTYPE (CEMT I PROC). You get a screen that lists the current status.
- Type CEMT INQUIRE PROCESSTYPE (CEMT I PROC) followed by as many of the other attributes as are necessary to limit the range of information that you require. So, for example, if you enter CEMT I PROC PROCESS, the resulting display will show you those processes with the audit level as the process-level auditing.

To change various attributes, you can:

- Make your changes on the INQUIRE screen after tabbing to the appropriate field.
- Use the CEMT SET PROCESSTYPE command (see Figure 108).

```
I PROC
STATUS: RESULTS - OVERTYPE TO MODIFY
Proc(BAMAUD ) Fil(PTYPSALE) Aud(CBTS ) Off Ena
Proc(BAMAUDP ) Fil(PTYPSALE) Aud(CBTS ) Off Ena
Proc(BAMSHR ) Fil(PTYPSALE) Aud(CBTS ) Ful Ena
Proc(BAMSHRA ) Fil(PTYPSALE) Aud(CBTS ) Act Ena
Proc(BAMSHRF ) Fil(PTYPSALE) Aud(CBTS ) Off Ena
Proc(BAMSHRP ) Fil(PTYPSALE) Aud(CBTS ) Pro Ena
Proc(BAMSHR2 ) Fil(PTYPSALE) Aud(CBTS ) Off dis
Proc(BAM1 ) Fil(PTYPSALE) Aud(CBTS ) Ful Ena
```

Figure 108. CEMT INQUIRE PROCESSTYPE Screen

If you place the cursor alongside a specific entry in the list and press Enter, CICS expands the display as shown in Figure 109 on page 234.

```
I PROC
RESULT - OVERTYPE TO MODIFY
  Processtype(BAMSHR)
  File(PTYPSALE)
  Auditlog(CBTS)
  Auditlevel( Off )
  Status( Enabled )
```

Figure 109. The Expanded Display of an Individual Entry

The Auditlevel field shows the level of audit logging currently active for processes of this type. The values are:

- Activity  
Activity-level auditing. Audit records are written from:
  1. The process audit points
  2. The activity primary audit points
- Full  
Full auditing. Audit records are written from:
  1. The process audit points
  2. The activity primary *and* secondary audit points
- Off  
No audit log records are written.
- Process  
Process-level auditing. Audit records are written from the process audit points only.

The Auditlog field displays the 8-character name of the CICS journal used as the audit log for processes of this type.

The File field displays the 8-character name of the CICS repository file on which the process and activity records for processes of this type are stored.

The PROCesstype field indicates that this panel relates to a PROCESSTYPE inquiry and displays the 8-character name of a process-type.

The Status field displays whether new processes of this type can be created. The values are:

- Disabled  
The installed definition of the process type is disabled. New processes of this type cannot be defined.
- Enabled  
The installed definition of the process type is enabled. New processes of this type can be defined.

## 13.3.2 CEMT INQUIRE TASK

You use this command to retrieve information about a user task.

INQUIRE TASK returns information about user tasks. Only information about user tasks can be displayed or changed; information about CICS-generated system tasks or subtasks cannot be displayed or changed. System tasks are those tasks started (and used internally) by CICS and not as a result of a user transaction.

Press the Clear key to clear the screen. There are two ways to start this transaction:

- Type CEMT INQUIRE TASK (CEMT I TA). You get a display that lists the current status.
- Type CEMT INQUIRE TASK (CEMT I TA) followed by as many of the other attributes as are necessary to limit the range of information that you require. So, for example, if you enter `cent i ta`, the resulting display will show you the details of only those tasks for which the data is not shared with other tasks (isolated).

To change various attributes, you can:

- Make your changes on the INQUIRE screen after tabbing to the appropriate field.
- Use the CEMT SET TASK command (see Figure 110).

```
IN TASK
STATUS:      RESULTS - OVERTYPE TO MODIFY
Tas(0000033) Tra(CEMT) Fac(944D) Sus Ter Pri( 255 )
             Sta(T0) Use(CICSUSER) Uow(AB9001D5F56CC800) Hty(ZCIOWAIT) Hva(DFHZARQ)
Tas(0000037) Tra(CEMT) Fac(S21D) Run Ter Pri( 255 )
             Sta(T0) Use(CICSUSER) Uow(AB9002E745F93A00)
```

Figure 110. CEMT INQUIRE TASK Screen

If you place the cursor alongside a specific entry in the list and press Enter, CICS expands the display as shown in Figure 111 on page 236.

```

IN TASK
RESULT - OVERTYPE TO MODIFY
Task(0000033)
Tranid(CEMT)
Facility(944D)
Runstatus(Suspended)
Ftype(Term)
Priority( 255 )
Purgetype(          )
Startcode(T0)
Userid(CICSUSER)
Uow(AB9001D5F56CC800)
Htype()
Hvalue()
Htime()
Indoubt(Backout)
Indoubtwait(Wait)
Bridge()
Identifier()
Indoubtmins(000000)
Db2plan()
Activity()
Activityid()
Process()
Processtype()
Tcb(Qr)

```

Figure 111. The Expanded Display of an Individual Entry

Activityid(value) is an identifier of the CICS BTS activity that this task is executing on behalf of.

Activity(value) displays the 16-character, user-assigned, name of the CICS BTS activity that this task is executing on behalf of.

Process(value) displays the 36-character name of the CICS BTS process that this task is executing on behalf of.

Processtype(value) displays the 8-character process-type of the CICS BTS process that this task is executing on behalf of.

### 13.3.3 CEMT SET PROCESSTYPE

This command is to change the attributes of a CICS BTS process type.

SET PROCESSTYPE enables you to change the current state of audit logging and the enablement status of BTS PROCESSTYPE definitions installed on this CICS region.

**Note:** If you are using BTS in a single CICS region, you can freely use the SET PROCESSTYPE command to modify your process types. However, if you are using BTS in a Parallel Sysplex, it is strongly recommended that you use CICSplex SM to make such changes. This is because it is essential to keep resource definitions in step with each other across the Parallel Sysplex.

Activity, Full, Off or Process specifies the level of audit logging to be applied to processes of this type. The values are:

- Activity

Activity-level auditing. Audit records will be written from:

1. The process audit points
2. The activity primary audit points

- FULL

Full auditing. Audit records will be written from:

1. The process audit points
2. The activity primary and secondary audit points

- Off

No audit log records will be written.

- Process

Process-level auditing. Audit records will be written from the process audit points only.

ALL specifies that any changes you specify are made to all process-types that you are authorized to access. Enabled|Disabled specifies whether new processes of this type can be created. The values are:

- Disabled

The installed definition of the process-type is disabled. New processes of this type cannot be defined.

- Enabled

The installed definition of the process-type is enabled. New processes of this type can be defined.

PROCESstype(value) specifies the 8-character name of the process-type whose attributes are to be changed.





# Appendix A. Company Organization Application Development Log

In this appendix we provide our detailed development log. It is arranged in the same chronological order as the application is being developed based on the process model described in Chapter 6, "Designing Your Application" on page 129. You find JCL, resource definitions, CBAM screen outputs, error message codes, and short descriptions of how we approach the problem and solve it.

## A.1 Resource Definitions

Figure 112 shows the JCL to define a repository data set where all the CICS BTS objects are stored when the application is running.

```
//DEF#CBTS JOB (999,POK),'CICSTS13',MSGCLASS=T,CLASS=A,
//          NOTIFY=&SYSUID
/* *****
/*  DELETE/DEFINE THE CLUSTER FOR A CBTS REPOSITORY DATASET
/*  *****
//DELETE   EXEC PGM=IDCAMS,REGION=1M
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
DELETE CICSSYSF.CBTS.PTYPCORG
SET MAXCC=0
/*
//DEFINE   EXEC PGM=IDCAMS,REGION=1M
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
/*                               */
/* DEFINE A CBTS PROCESS REPOSITORY DATASET */
/* FOR PROCESS TYPE                */
/*                               */
DEFINE CLUSTER (NAME (CICSSYSF.CBTS.PTYPCORG) -
              LOG(UNDO) -
              CYL(2 1) -
              CISZ(4096) -
              SPANNED -
              VOLUMES(TOTCI4) -
              STORCLAS(CICSRSL) -
              KEYS(50 0) -
              INDEXED -
              RECORDSIZE(4096 16384) -
              FREESPACE( 5 5 ) -
              SHAREOPTIONS( 2 3 ) -
              ) -
DATA (NAME (CICSSYSF.CBTS.PTYPCORG.DATA) -
      ) -
INDEX (NAME (CICSSYSF.CBTS.PTYPCORG.INDEX) -
       )
/*
//
```

Figure 112. Sample JCL to Define Repository Dataset

After we have defined the repository, we define a process type for the new application.

Figure 113 on page 240 shows the CEDA PROCESSTYPE definition screen.

```

OVERTYPE TO MODIFY                                CICS RELEASE = 0530
CEDA DEFINE PROCESSTYPE( COMP-ORG )
  PROCESSTYPE      : COMP-ORG
  GROUP            : COMPORG
  DESCRIPTION      ==> PROCESS TYPE DEFINITION FOR COMPANY ORGANIZATION
  INITIAL STATUS
  STATUS           ==> ENABLED                ENABLED | DISABLED
  DATA SET PARAMETERS
  FILE            ==> PTYPCORG
  AUDIT TRAIL
  AUDITLOG        ==> AUDITLOG
  AUDITLEVEL      ==> OFF                    OFF | PROCESS | ACTIVITY | FULL

I New group COMPORG created.

                                           SYSID=PAA1 APPLID=SCSCPAA1
DEFINE SUCCESSFUL                          TIME: 15.29.38 DATE: 99.054
PF 1 HELP 2 COM 3 END                      6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL

```

Figure 113. Processtype Resource Definition

Next, we define Audit Log for debugging.

Figure 114 shows the JCL that we used to define a model log stream that is used to dynamically create a log stream referenced in the JOURNALMODEL.

```

//DEFALOG JOB (999,POK),'CICSTS3',CLASS=A,MSGCLASS=T,
//          NOTIFY=&SYSUID
//*
//** JOB TO ADD LOGSTREAM DEFS FOR CBTS AUDIT LOG
//**
//IXCMIAPU EXEC PGM=IXCMIAPU
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
  DATA TYPE(LOGR) REPORT(YES)
  DEFINE LOGSTREAM
    NAME(CICSTS13.CBTS.CORG)
    STRUCTNAME(LOG_GENERAL_001)
    STG_DUPLEX(NO)
    LS_SIZE(2000)
    HLQ(CICS)
    HIGHOFFLOAD(60)
    LOWOFFLOAD(10)
    MODEL(YES)
/*
//

```

Figure 114. Sample JCL to Define Audit Log

After we created the log stream, we made a JOURNALMODEL definition for the audit log to CICS through a CEDA panel.

Figure 115 on page 241 shows the JOURNALMODEL resource definition for the audit log.

```
OVERTYPE TO MODIFY                                CICS RELEASE = 0530
CEDA DEFINE JOURNALMODEL( CORGLOG )
JOURNALMODEL  : CORGLOG
GROUP         : COMPORG
DESCRIPTION   ==> AUDIT LOG FOR COMPANY ORGANIZATION PROCESSES
JOURNALNAME   ==> CORGLOG
TYPE          ==> MVS                                MVS | SMF | DUMMY
STREAMNAME    ==> CICSTS13.CBTS.CORG

                                           SYSID=PAA1 APPLID=SCSCPAA1
DEFINE SUCCESSFUL                               TIME: 20.42.34 DATE: 99.054
PF 1 HELP 2 COM 3 END                          6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
```

Figure 115. Journal Model Resource Definition for the Audit Log

We define the repository to the CICS environment through CEDA panel.

Figure 116 on page 243 shows the resource definition for the repository.

```

DEFINE FILE
OVERTYPE TO MODIFY
CEDA DEFINE FILE(          )
FILE          ==> PTYPCORG
GROUP         ==> COMPORG
DESCRIPTION   ==> COMPANY ORGANIZATION REPOSITORY FILE
VSAM PARAMETERS
DSNAME        ==> CICSSYSF.CBTS.PTYPCORG
PASSWORD      ==>
                PASSWORD NOT SPECIFIED
RLSACCESS     ==> YES
                YES | NO
LSRPOOLID     ==> 1
                1-8 | NONE
READINTEG     ==> CONSISTENT
                UNCOMMITTED | CONSISTENT | REPEATABLE
DSNSHARING    ==> ALLREQS
                ALLREQS | MODIFYREQS
STRINGS       ==> 001
                1-255
NSRGROUP      ==>
REMOTE ATTRIBUTES
REMOTESYSTEM  ==>
REMOTENAME    ==>
REMOTE AND CFDATATABLE PARAMETERS
RECORDSIZE    ==>
                1-32767
KEYLENGTH     ==>
                1-255 (1-16 FOR CF DATATABLE)
INITIAL STATUS
STATUS        ==> ENABLED
                ENABLED | DISABLED | UNENABLED
OPENTIME      ==> FIRSTREF
                FIRSTREF | STARTUP
DISPOSITION   ==> SHARE
                SHARE | OLD
BUFFERS
DATABUFFERS  ==> 00002
                2-32767
INDEXBUFFERS ==> 00001
                1-32767
DATATABLE PARAMETERS
TABLE         ==> NO
                NO | CICS | USER | CF
MAXNUMRECS   ==> NOLIMIT
                NOLIMIT | 1-99999999
CFDATATABLE PARAMETERS
CFDTPPOOL    ==>
TABLENAME     ==>
UPDATEMODEL  ==> LOCKING
                CONTENTION | LOCKING
LOAD         ==> NO
                NO | YES
DATA FORMAT
RECORDFORMAT ==> V
                V | F
OPERATIONS
ADD          ==> YES
                NO | YES
BROWSE       ==> YES
                NO | YES
DELETE       ==> YES
                NO | YES
READ         ==> YES
                YES | NO
UPDATE       ==> YES
                NO | YES
AUTO JOURNALLING
JOURNAL      ==> NO
                NO | 1-99
JNLREAD      ==> NONE
                NONE | UPDATEONLY | READONLY | ALL
JNLSYNCREAD ==> NO
                NO | YES
JNLUPDATE    ==> NO
                NO | YES
JNLADD       ==> NONE
                NONE | BEFORE | AFTER | ALL
JNLSYNCWRITE ==> YES
                YES | NO
RECOVERY PARAMETERS
RECOVERY     ==> BACKOUTONLY
                NONE | BACKOUTONLY | ALL
FWDRECOVLOG ==> NO
                NO | 1-99
BACKUPTYPE   ==> STATIC
                STATIC | DYNAMIC
SECURITY
RESSECNUM    : 00
                0-24 | PUBLIC

                SYSID=PAA1 APPLID=SCSCPAA1
DEFINE SUCCESSFUL
                TIME: 21.06.51 DATE: 99.054
PF 1 HELP 2 COM 3 END
                6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL

```

Figure 116. File Control Resource Definition for the CICS BTS Repository

We now make application-related resource definitions for transactions, programs and maps.

Figure 117 shows the JCL to define the resources for the organization application.

```
//DEF#RDO1 JOB (999,POK),NOTIFY=&SYSUID,  
//      CLASS=A,MSGCLASS=T,MSGLEVEL=(1,1),TIME=1440  
//*  
//*  RUN TO INSTALL THE COMPANY ORGANIZATION EMPLOYMENT APPLICATION  
//*  
//CSDUP   EXEC PGM=DFHCSDUP,REGION=4M  
//STEPLIB DD DISP=SHR,DSN=CICSLA13.CICS.SDFHLOAD  
//      DD DISP=SHR,DSN=CICSLA13.CPSM.SEYULOAD  
//DFHCSD  DD DISP=SHR,DSN=CICSSYSF.CICSTS13.DFHCSD  
//SYSPRINT DD SYSOUT=*  
//SYSIN   DD DISP=SHR,DSN=CICSSYSF.CBTS.CY.CNTL(RD0)  
/*
```

Figure 117. Sample JCL to Define Resource for the Organization Application

Figure 118 on page 245 shows the resources used by the organization application.

```

DEFINE MAPSET(COMPEMP) GROUP(COMPORG)
  RESIDENT(NO) USAGE(NORMAL) USELPACOPY(NO) STATUS(ENABLED)
  DESCRIPTION(COMPANY EMPLOYMENT MENU)
DEFINE PROGRAM(EMPLMAIN) GROUP(COMPORG)
  DESCRIPTION(NEW EMPLOYMENT MAIN PROGRAM)
  LANGUAGE(COBOL) RELOAD(NO) RESIDENT(NO) USAGE(NORMAL)
  USELPACOPY(NO) STATUS(ENABLED) CEDF(YES) DATALOCATION(BELOW)
  EXECKEY(USER) CONCURRENCY(QUASIRENT) DYNAMIC(NO)
  EXECUTIONSET(FULLAPI)
DEFINE PROGRAM(EMPLROOT) GROUP(COMPORG)
  DESCRIPTION(NEW EMPLOYMENT ROOT PROGRAM)
  LANGUAGE(COBOL) RELOAD(NO) RESIDENT(NO) USAGE(NORMAL)
  USELPACOPY(NO) STATUS(ENABLED) CEDF(YES) DATALOCATION(BELOW)
  EXECKEY(USER) CONCURRENCY(QUASIRENT) DYNAMIC(NO)
  EXECUTIONSET(FULLAPI)
DEFINE PROGRAM(EMPLACTM) GROUP(COMPORG)
  DESCRIPTION(NEW EMPLOYMENT ACTIVITY-ROOT PROGRAM)
  LANGUAGE(COBOL) RELOAD(NO) RESIDENT(NO) USAGE(NORMAL)
  USELPACOPY(NO) STATUS(ENABLED) CEDF(YES) DATALOCATION(BELOW)
  EXECKEY(USER) CONCURRENCY(QUASIRENT) DYNAMIC(NO)
  EXECUTIONSET(FULLAPI)
DEFINE PROGRAM(EMPLACTR) GROUP(COMPORG)
  DESCRIPTION(NEW EMPLOYMENT ACTIVITY-ROLE PROGRAM)
  LANGUAGE(COBOL) RELOAD(NO) RESIDENT(NO) USAGE(NORMAL)
  USELPACOPY(NO) STATUS(ENABLED) CEDF(YES) DATALOCATION(BELOW)
  EXECKEY(USER) CONCURRENCY(QUASIRENT) DYNAMIC(NO)
  EXECUTIONSET(FULLAPI)
DEFINE PROGRAM(EMPLACTT) GROUP(COMPORG)
  DESCRIPTION(NEW EMPLOYMENT ACTIVITY-TASK PROGRAM)
  LANGUAGE(COBOL) RELOAD(NO) RESIDENT(NO) USAGE(NORMAL)
  USELPACOPY(NO) STATUS(ENABLED) CEDF(YES) DATALOCATION(BELOW)
  EXECKEY(USER) CONCURRENCY(QUASIRENT) DYNAMIC(NO)
  EXECUTIONSET(FULLAPI)
DEFINE PROGRAM(EMPLACTS) GROUP(COMPORG)
  DESCRIPTION(NEW EMPLOYMENT ACTIVITY-SKILLS PROGRAM)
  LANGUAGE(COBOL) RELOAD(NO) RESIDENT(NO) USAGE(NORMAL)
  USELPACOPY(NO) STATUS(ENABLED) CEDF(YES) DATALOCATION(BELOW)
  EXECKEY(USER) CONCURRENCY(QUASIRENT) DYNAMIC(NO)
  EXECUTIONSET(FULLAPI)
DEFINE PROGRAM(JOBEACTM) GROUP(COMPORG)
  DESCRIPTION(NEW JOB ROOT PROGRAM)
  LANGUAGE(COBOL) RELOAD(NO) RESIDENT(NO) USAGE(NORMAL)
  USELPACOPY(NO) STATUS(ENABLED) CEDF(YES) DATALOCATION(BELOW)
  EXECKEY(USER) CONCURRENCY(QUASIRENT) DYNAMIC(NO)
  EXECUTIONSET(FULLAPI)
DEFINE PROGRAM(JOBEACTU) GROUP(COMPORG)
  DESCRIPTION(NEW JOB ACTIVITY-UNIT PROGRAM)
  LANGUAGE(COBOL) RELOAD(NO) RESIDENT(NO) USAGE(NORMAL)
  USELPACOPY(NO) STATUS(ENABLED) CEDF(YES) DATALOCATION(BELOW)
  EXECKEY(USER) CONCURRENCY(QUASIRENT) DYNAMIC(NO)
  EXECUTIONSET(FULLAPI)
DEFINE PROGRAM(JOBEACTT) GROUP(COMPORG)
  DESCRIPTION(NEW JOB ACTIVITY-TASK PROGRAM)
  LANGUAGE(COBOL) RELOAD(NO) RESIDENT(NO) USAGE(NORMAL)
  USELPACOPY(NO) STATUS(ENABLED) CEDF(YES) DATALOCATION(BELOW)
  EXECKEY(USER) CONCURRENCY(QUASIRENT) DYNAMIC(NO)
  EXECUTIONSET(FULLAPI)
DEFINE PROGRAM(JOBEACTR) GROUP(COMPORG)
  DESCRIPTION(NEW EMPLOYMENT ACTIVITY-ROLE PROGRAM)
  LANGUAGE(COBOL) RELOAD(NO) RESIDENT(NO) USAGE(NORMAL)
  USELPACOPY(NO) STATUS(ENABLED) CEDF(YES) DATALOCATION(BELOW)
  EXECKEY(USER) CONCURRENCY(QUASIRENT) DYNAMIC(NO)
  EXECUTIONSET(FULLAPI)
DEFINE PROGRAM(JOBEACTS) GROUP(COMPORG)
  DESCRIPTION(NEW EMPLOYMENT ACTIVITY-SKILLS PROGRAM)
  LANGUAGE(COBOL) RELOAD(NO) RESIDENT(NO) USAGE(NORMAL)
  USELPACOPY(NO) STATUS(ENABLED) CEDF(YES) DATALOCATION(BELOW)
  EXECKEY(USER) CONCURRENCY(QUASIRENT) DYNAMIC(NO)
  EXECUTIONSET(FULLAPI)

```

Figure 118 (Part 1 of 3). Resource Definition for the Organization Application

```

DEFINE TRANSACTION(EMPM) GROUP(COMPORG)
DESCRIPTION(MAIN TRANSACTION TO START THE EMPLOYMENT PROCESS)
PROGRAM(EMPLMAIN) TWASIZE(0) PROFILE(DFHICICST) STATUS(ENABLED)
TASKDATALOC(ANY) TASKDATAKEY(USER) STORAGEECLEAR(NO)
RUNAWAY(SYSTEM) SHUTDOWN(DISABLED) ISOLATE(YES) DYNAMIC(NO)
ROUTABLE(NO) PRIORITY(1) TRANCLASS(DFHTCL00) DTIMOUT(NO)
RESTART(NO) SPURGE(YES) TPURGE(YES) DUMP(YES) TRACE(YES)
CONFDATA(NO) ACTION(BACKOUT) WAIT(YES) WAITTIME(0,0,10)
RESSEC(NO) CMDSEC(NO)

DEFINE TRANSACTION(EMPR) GROUP(COMPORG)
DESCRIPTION(ROOT TRANSACTION OF THE EMPLOYMENT PROCESS)
PROGRAM(EMPLROOT) TWASIZE(0) PROFILE(DFHICICST) STATUS(ENABLED)
TASKDATALOC(ANY) TASKDATAKEY(USER) STORAGEECLEAR(NO)
RUNAWAY(SYSTEM) SHUTDOWN(DISABLED) ISOLATE(YES) DYNAMIC(NO)
ROUTABLE(NO) PRIORITY(1) TRANCLASS(DFHTCL00) DTIMOUT(NO)
RESTART(NO) SPURGE(YES) TPURGE(YES) DUMP(YES) TRACE(YES)
CONFDATA(NO) ACTION(BACKOUT) WAIT(YES) WAITTIME(0,0,10)
RESSEC(NO) CMDSEC(NO)

DEFINE TRANSACTION(EMPN) GROUP(COMPORG)
DESCRIPTION(ROOT TRANSACTION OF THE NEW EMPLOYEE)
PROGRAM(EMPLACTM) TWASIZE(0) PROFILE(DFHICICST) STATUS(ENABLED)
TASKDATALOC(ANY) TASKDATAKEY(USER) STORAGEECLEAR(NO)
RUNAWAY(SYSTEM) SHUTDOWN(DISABLED) ISOLATE(YES) DYNAMIC(NO)
ROUTABLE(NO) PRIORITY(1) TRANCLASS(DFHTCL00) DTIMOUT(NO)
RESTART(NO) SPURGE(YES) TPURGE(YES) DUMP(YES) TRACE(YES)
CONFDATA(NO) ACTION(BACKOUT) WAIT(YES) WAITTIME(0,0,10)
RESSEC(NO) CMDSEC(NO)

DEFINE TRANSACTION(EMPO) GROUP(COMPORG)
DESCRIPTION(ACTIVITY TRANSACTION OF THE EMPLOYEE-ROLE CONNECTION)
PROGRAM(EMPLACTR) TWASIZE(0) PROFILE(DFHICICST) STATUS(ENABLED)
TASKDATALOC(ANY) TASKDATAKEY(USER) STORAGEECLEAR(NO)
RUNAWAY(SYSTEM) SHUTDOWN(DISABLED) ISOLATE(YES) DYNAMIC(NO)
ROUTABLE(NO) PRIORITY(1) TRANCLASS(DFHTCL00) DTIMOUT(NO)
RESTART(NO) SPURGE(YES) TPURGE(YES) DUMP(YES) TRACE(YES)
CONFDATA(NO) ACTION(BACKOUT) WAIT(YES) WAITTIME(0,0,10)
RESSEC(NO) CMDSEC(NO)

DEFINE TRANSACTION(EMPT) GROUP(COMPORG)
DESCRIPTION(ACTIVITY TRANSACTION OF THE EMPLOYEE-TASK CONNECTION)
PROGRAM(EMPLACTT) TWASIZE(0) PROFILE(DFHICICST) STATUS(ENABLED)
TASKDATALOC(ANY) TASKDATAKEY(USER) STORAGEECLEAR(NO)
RUNAWAY(SYSTEM) SHUTDOWN(DISABLED) ISOLATE(YES) DYNAMIC(NO)
ROUTABLE(NO) PRIORITY(1) TRANCLASS(DFHTCL00) DTIMOUT(NO)
RESTART(NO) SPURGE(YES) TPURGE(YES) DUMP(YES) TRACE(YES)
CONFDATA(NO) ACTION(BACKOUT) WAIT(YES) WAITTIME(0,0,10)
RESSEC(NO) CMDSEC(NO)

DEFINE TRANSACTION(EMPS) GROUP(COMPORG)
DESCRIPTION(ACTIVITY TRANSACTION OF THE EMPLOYEE-SKILLS CONN)
PROGRAM(EMPLACTS) TWASIZE(0) PROFILE(DFHICICST) STATUS(ENABLED)
TASKDATALOC(ANY) TASKDATAKEY(USER) STORAGEECLEAR(NO)
RUNAWAY(SYSTEM) SHUTDOWN(DISABLED) ISOLATE(YES) DYNAMIC(NO)
ROUTABLE(NO) PRIORITY(1) TRANCLASS(DFHTCL00) DTIMOUT(NO)
RESTART(NO) SPURGE(YES) TPURGE(YES) DUMP(YES) TRACE(YES)
CONFDATA(NO) ACTION(BACKOUT) WAIT(YES) WAITTIME(0,0,10)
RESSEC(NO) CMDSEC(NO)

```

Figure 118 (Part 2 of 3). Resource Definition for the Organization Application



```

DEFINE TRANSACTION(JOBN) GROUP(COMPORG)
DESCRIPTION(ROOT TRANSACTION OF THE NEW JOB)
PROGRAM(JOBEACTM) TWASIZE(0) PROFILE(DFHICICST) STATUS(ENABLED)
TASKDATALOC(ANY) TASKDATAKEY(USER) STORAGEECLEAR(NO)
RUNAWAY(SYSTEM) SHUTDOWN(DISABLED) ISOLATE(YES) DYNAMIC(NO)
ROUTABLE(NO) PRIORITY(1) TRANCLASS(DFHTCL00) DTIMOUT(NO)
RESTART(NO) SPURGE(YES) TPURGE(YES) DUMP(YES) TRACE(YES)
CONFDATA(NO) ACTION(BACKOUT) WAIT(YES) WAITTIME(0,0,10)
RESSEC(NO) CMDSEC(NO)

DEFINE TRANSACTION(JOBU) GROUP(COMPORG)
DESCRIPTION(ACTIVITY TRANSACTION OF THE JOB-UNIT CONNECTION)
PROGRAM(JOBEACTU) TWASIZE(0) PROFILE(DFHICICST) STATUS(ENABLED)
TASKDATALOC(ANY) TASKDATAKEY(USER) STORAGEECLEAR(NO)
RUNAWAY(SYSTEM) SHUTDOWN(DISABLED) ISOLATE(YES) DYNAMIC(NO)
ROUTABLE(NO) PRIORITY(1) TRANCLASS(DFHTCL00) DTIMOUT(NO)
RESTART(NO) SPURGE(YES) TPURGE(YES) DUMP(YES) TRACE(YES)
CONFDATA(NO) ACTION(BACKOUT) WAIT(YES) WAITTIME(0,0,10)
RESSEC(NO) CMDSEC(NO)

DEFINE TRANSACTION(JOBT) GROUP(COMPORG)
DESCRIPTION(ACTIVITY TRANSACTION OF THE JOB-TASK CONNECTION)
PROGRAM(JOBEACTT) TWASIZE(0) PROFILE(DFHICICST) STATUS(ENABLED)
TASKDATALOC(ANY) TASKDATAKEY(USER) STORAGEECLEAR(NO)
RUNAWAY(SYSTEM) SHUTDOWN(DISABLED) ISOLATE(YES) DYNAMIC(NO)
ROUTABLE(NO) PRIORITY(1) TRANCLASS(DFHTCL00) DTIMOUT(NO)
RESTART(NO) SPURGE(YES) TPURGE(YES) DUMP(YES) TRACE(YES)
CONFDATA(NO) ACTION(BACKOUT) WAIT(YES) WAITTIME(0,0,10)
RESSEC(NO) CMDSEC(NO)

DEFINE TRANSACTION(JOBO) GROUP(COMPORG)
DESCRIPTION(ACTIVITY TRANSACTION OF THE JOB-ROLE CONNECTION)
PROGRAM(JOBEACTR) TWASIZE(0) PROFILE(DFHICICST) STATUS(ENABLED)
TASKDATALOC(ANY) TASKDATAKEY(USER) STORAGEECLEAR(NO)
RUNAWAY(SYSTEM) SHUTDOWN(DISABLED) ISOLATE(YES) DYNAMIC(NO)
ROUTABLE(NO) PRIORITY(1) TRANCLASS(DFHTCL00) DTIMOUT(NO)
RESTART(NO) SPURGE(YES) TPURGE(YES) DUMP(YES) TRACE(YES)
CONFDATA(NO) ACTION(BACKOUT) WAIT(YES) WAITTIME(0,0,10)
RESSEC(NO) CMDSEC(NO)

DEFINE TRANSACTION(JOBS) GROUP(COMPORG)
DESCRIPTION(ACTIVITY TRANSACTION OF THE JOB-SKILLS CONNECTION)
PROGRAM(JOBEACTS) TWASIZE(0) PROFILE(DFHICICST) STATUS(ENABLED)
TASKDATALOC(ANY) TASKDATAKEY(USER) STORAGEECLEAR(NO)
RUNAWAY(SYSTEM) SHUTDOWN(DISABLED) ISOLATE(YES) DYNAMIC(NO)
ROUTABLE(NO) PRIORITY(1) TRANCLASS(DFHTCL00) DTIMOUT(NO)
RESTART(NO) SPURGE(YES) TPURGE(YES) DUMP(YES) TRACE(YES)
CONFDATA(NO) ACTION(BACKOUT) WAIT(YES) WAITTIME(0,0,10)
RESSEC(NO) CMDSEC(NO)

```

Figure 118 (Part 3 of 3). Resource Definition for the Organization Application

Next, we define the map using BMS macros.

Figure 119 shows the BMS macros to define Company Organization Employment entry map.

```

          TITLE 'CICS BTS - MAP FOR company organization'
*****
*
* MODULE NAME = compemp
*
* DESCRIPTIVE NAME = COMPANY Entry Map for CICS BTS Application
*
* %COPYRIGHT          IBM CONFIDENTIAL
*
* STATUS = %SP00
*
*-----*
*
* CHANGE ACTIVITY :
* $SEG(COMPEMP),COMP(SAMPLES),PROD(%PRODUCT) :
*
*   PN= REASON REL YYMMDD TBALL : REMARKS
*   $P0= .      %B0 990224      : Created.
*
*****
COMPEMS  DFHMSD TYPE=&SYSPARM,MODE=INOUT,CTRL=(FREEKB,FRSET),
          TIOAPFX=YES,LANG=COBOL,EXTATT=MAPONLY
COMPANY  DFHMDI SIZE=(24,80)
          DFHMDF POS=(01,10),LENGTH=40,ATTRB=(BRT,ASKIP),
          INITIAL='CICS BTS COMPANY EMPLOYMENT ENTRY FORM',
          COLOR=YELLOW,HIGHLIGHT=UNDERLINE
          DFHMDF POS=(01,51),LENGTH=01,ATTRB=(PROT,ASKIP)
          DFHMDF POS=(01,60),LENGTH=4,ATTRB=(BRT,ASKIP),
          INITIAL='DATE',COLOR=YELLOW
DATE     DFHMDF POS=(01,65),LENGTH=10,ATTRB=(BRT,ASKIP)
          DFHMDF POS=(01,76),LENGTH=01,ATTRB=(PROT,ASKIP)
          DFHMDF POS=(02,60),LENGTH=4,ATTRB=(BRT,ASKIP),
          INITIAL='TIME',COLOR=YELLOW
TIME     DFHMDF POS=(02,65),LENGTH=8,ATTRB=(BRT,ASKIP)
          DFHMDF POS=(02,74),LENGTH=01,ATTRB=(PROT,ASKIP)
          DFHMDF POS=(05,10),LENGTH=15,ATTRB=(PROT,ASKIP),
          COLOR=BLUE,INITIAL='EMPLOYEE NAME'
EMPNAM   DFHMDF POS=(05,30),LENGTH=30,ATTRB=(IC,ALPHA,UNPROT),
          COLOR=NEUTRAL,INITIAL='first, last name'
          DFHMDF POS=(05,61),LENGTH=01,ATTRB=(PROT,ASKIP)
          DFHMDF POS=(07,10),LENGTH=19,ATTRB=(PROT,ASKIP),
          COLOR=RED,INITIAL='ORGANIZATION UNIT'
ORGUNIT  DFHMDF POS=(07,30),LENGTH=04,ATTRB=(ALPHA,UNPROT),
          COLOR=NEUTRAL,INITIAL='ou00'
          DFHMDF POS=(07,35),LENGTH=01,ATTRB=(PROT,ASKIP)
          DFHMDF POS=(09,10),LENGTH=15,ATTRB=(PROT,ASKIP),
          COLOR=RED,INITIAL='EMPLOYEE BAND'
EMPBAND  DFHMDF POS=(09,30),LENGTH=04,ATTRB=(ALPHA,UNPROT),
          COLOR=NEUTRAL,INITIAL='aa00'
          DFHMDF POS=(09,35),LENGTH=01,ATTRB=(PROT,ASKIP)
          DFHMDF POS=(11,10),LENGTH=15,ATTRB=(PROT,ASKIP),
          COLOR=BLUE,INITIAL='EMPLOYEE ROLE 1'
EMPROL1  DFHMDF POS=(11,30),LENGTH=20,ATTRB=(ALPHA,UNPROT),
          COLOR=NEUTRAL,INITIAL='role 1'
          DFHMDF POS=(11,51),LENGTH=01,ATTRB=(PROT,ASKIP)
          DFHMDF POS=(12,10),LENGTH=15,ATTRB=(PROT,ASKIP),
          COLOR=BLUE,INITIAL='EMPLOYEE ROLE 2'
EMPROL2  DFHMDF POS=(12,30),LENGTH=20,ATTRB=(ALPHA,UNPROT),
          COLOR=NEUTRAL,INITIAL='role 2'
          DFHMDF POS=(12,51),LENGTH=01,ATTRB=(PROT,ASKIP)
          DFHMDF POS=(14,10),LENGTH=15,ATTRB=(PROT,ASKIP),
          COLOR=BLUE,INITIAL='EMPLOYEE TASK 1'

```

Figure 119 (Part 1 of 2). Company Organization Employment Entry Map Definition

```

EMPTAS1 DFHMDf POS=(14,30),LENGTH=20,ATTRB=(ALPHA,UNPROT),      *
        COLOR=NEUTRAL,INITIAL='task 1'
        DFHMDf POS=(14,51),LENGTH=01,ATTRB=(PROT,ASKIP)
        DFHMDf POS=(15,10),LENGTH=15,ATTRB=(PROT,ASKIP),      *
        COLOR=BLUE,INITIAL='EMPLOYEE TASK 2'
EMPTAS2 DFHMDf POS=(15,30),LENGTH=20,ATTRB=(ALPHA,UNPROT),      *
        COLOR=NEUTRAL,INITIAL='task 2'
        DFHMDf POS=(15,51),LENGTH=01,ATTRB=(PROT,ASKIP)
        DFHMDf POS=(16,10),LENGTH=15,ATTRB=(PROT,ASKIP),      *
        COLOR=BLUE,INITIAL='EMPLOYEE TASK 3'
EMPTAS3 DFHMDf POS=(16,30),LENGTH=20,ATTRB=(ALPHA,UNPROT),      *
        COLOR=NEUTRAL,INITIAL='task 3'
        DFHMDf POS=(16,51),LENGTH=01,ATTRB=(PROT,ASKIP)
        DFHMDf POS=(18,10),LENGTH=15,ATTRB=(PROT,ASKIP),      *
        COLOR=GREEN,INITIAL='JOB DESCRIPTION'
JOBDESC DFHMDf POS=(18,30),LENGTH=30,ATTRB=(ALPHA,UNPROT),      *
        COLOR=NEUTRAL,INITIAL='Job Description'
        DFHMDf POS=(18,61),LENGTH=01,ATTRB=(PROT,ASKIP)
        DFHMDf POS=(19,10),LENGTH=15,ATTRB=(PROT,ASKIP),      *
        COLOR=GREEN,INITIAL='(OPTIONAL) '
        DFHMDf POS=(23,1),LENGTH=4,ATTRB=ASKIP,COLOR=TURQUOISE, *
        INITIAL='MSG:'
MSG      DFHMDf POS=(23,6),LENGTH=73,COLOR=TURQUOISE
        DFHMDf POS=(23,80),LENGTH=01,ATTRB=(PROT,ASKIP)
        DFHMSD TYPE=FINAL
        END

```

Figure 119 (Part 2 of 2). Company Organization Employment Entry Map Definition

Figure 120 shows the screen layout of the Company Organization Employment entry map.

```

                CICS BTS COMPANY EMPLOYMENT ENTRY FORM                DATE
                                                                TIME

EMPLOYEE NAME      first, last name

ORGANIZATION UNIT  ou00

EMPLOYEE BAND      aa00

EMPLOYEE ROLE 1   role 1
EMPLOYEE ROLE 2   role 2
EMPLOYEE TASK 1   task 1
EMPLOYEE TASK 2   task 2
EMPLOYEE TASK 3   task 3

JOB DESCRIPTION    Job Description
(OPTIONAL)

MSG:

```

Figure 120. Company Organization Employment Entry Map Diagram

Then, we compile the map.

Figure 121 shows the JCL to compile map.

```
//CMPMAP JOB (POK,9999),'ASS CICS',CLASS=A,MSGCLASS=S,REGION=4M,
//      NOTIFY=&SYSUID
//*
//MAPS  PROC INDX='CICSTS13.CICS',          FOR SDFHMAC
//      TARGETX='CICSSYSF.CBTS',          TARGET HLQ FOR MAP
//      SOURCEX='CICSSYSF.CBTS.CY',       SOURCE HLQ
//      DSCTLIB='CICSSYSF.CBTS.DSECT',    TARGET FOR DSECT
//      MAPNAME=,                          NAME OF MAPSET - REQUIRED
//      A=,                                  A=A FOR ALIGNED MAP
//      RMODE=24,                            24/ANY
//      ASMBLR=ASMA90,                       ASSEMBLER PROGRAM NAME
//      REG=2048K,                           REGION FOR ASSEMBLY
//      OUTC=*,                              PRINT SYSOUT CLASS
//      WORK=SYSDA                           WORK FILE UNIT
//COPY  EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=&OUTC
//SYSUT2  DD DSN=&&TEMPM,UNIT=&WORK,DISP=(,PASS),
//      DCB=(RECFM=FB,LRECL=80,BLKSIZE=400),
//      SPACE=(400,(50,50))
//SYSIN  DD DUMMY
//SYSUT1  DD DISP=SHR,DSN=&SOURCEX..CNTL(&MAPNAME) THE MAP SOURCE
//ASMMAP  EXEC PGM=&ASMBLR,REGION=&REG,
//      PARM='SYSPARM(&A.MAP),DECK,NOOBJECT'
//* PARM='SYSPARM(&A.MAP),DECK,NOLOAD'
//SYSPRINT DD SYSOUT=&OUTC
//SYSLIB  DD DSN=&INDEX..SDFHMAC,DISP=SHR
//      DD DSN=SYS1.MACLIB,DISP=SHR
//SYSUT1  DD UNIT=&WORK,SPACE=(CYL,(5,5))
//SYSUT2  DD UNIT=&WORK,SPACE=(CYL,(5,5))
//SYSUT3  DD UNIT=&WORK,SPACE=(CYL,(5,5))
//SYSPUNCH DD DSN=&&MAP,DISP=(,PASS),UNIT=&WORK,
//      DCB=(RECFM=FB,LRECL=80,BLKSIZE=400),
//      SPACE=(400,(50,50))
//SYSIN  DD DSN=&&TEMPM,DISP=(OLD,PASS)
//LINKMAP EXEC PGM=IEWL,PARM='LIST,LET,XREF,RMODE(&RMODE)'
//SYSPRINT DD SYSOUT=&OUTC
//SYSLMOD DD DSN=&TARGETX..LOADLIB(&MAPNAME),DISP=SHR
//SYSUT1  DD UNIT=&WORK,SPACE=(1024,(20,20))
//SYSLIN  DD DSN=&&MAP,DISP=(OLD,DELETE)
//* NOLOAD CHANGED TO NOOBJECT @BA91512
//ASMDSECT EXEC PGM=&ASMBLR,REGION=&REG,
//      PARM='SYSPARM(&A.DSECT),DECK,NOOBJECT'
//SYSPRINT DD SYSOUT=&OUTC
//SYSLIB  DD DSN=&INDEX..SDFHMAC,DISP=SHR
//      DD DSN=SYS1.MACLIB,DISP=SHR
//SYSUT1  DD UNIT=&WORK,SPACE=(CYL,(5,5))
//SYSUT2  DD UNIT=&WORK,SPACE=(CYL,(5,5))
//SYSUT3  DD UNIT=&WORK,SPACE=(CYL,(5,5))
//SYSPUNCH DD DSN=&DSCTLIB(&MAPNAME),DISP=OLD
//SYSIN  DD DSN=&&TEMPM,DISP=(OLD,DELETE)
//      PEND
//DOMAPS  EXEC MAPS,MAPNAME=COMPEMP
//COPY.SYSUT1 DD DISP=SHR,DSN=CICSSYSF.CBTS.CY.CNTL(&MAPNAME)
//LINKMAP.SYSLMOD DD DSN=&TARGETX..LOADLIB(&MAPNAME),DISP=SHR
```

Figure 121. Sample JCL to Compile the Employment Entry Map

Table 17 on page 251 shows the inventory of the company organization employment business process (activity, program, transaction, and event).

<i>Table 17. Activity, Transaction, Program, and Event Names in the Application</i>			
<b>Activity</b>	<b>Transaction</b>	<b>Program</b>	<b>Event</b>
-	EMPM	EMPMAIN	-
DFHROOT	EMPR	EMPROOT	-
NEWEMPL	EMPN	EMPLACTM	EMPL-EVENT
EMPLROLE	EMPO	EMPLACTR	EMPL-ROLE-EVENT
EMPLTASK	EMPT	EMPLACTT	EMPL-TASK-EVENT
EMPLCOSC	EMPS	EMPLACTS	EMPL-COSC-EVENT
NEWEMJD	EMJD	EMPLJOBE	-
NEWJD	JOBN	JOBEACTM	JOB-EVENT
JOBEORGU	JOBU	JOBEACTU	JOBE-ORGU-EVENT
JOBETASK	JOBT	JOBEACTT	JOBE-TASK-EVENT
JOBEROLE	JOBO	JOBEACTR	JOBE-ROLE-EVENT
JOBECOSC	JOBS	JOBEACTS	JOBE-COSC-EVENT
-	EMPC	EMPLCONF	-

## A.2 Program Development Problem Log

The following is a log of what happened when we developed the application.

- Feb 25, 1999
  1. Map and Mapset were wrongly coded in the program and were not in sync with the map generation.
  2. The process name contained spaces, thus causing an invalid request.
  3. RUN ACQPROCESS failed with RESP(108) RESP2(27) without the root program coded.

Figure 122 shows the CBAM screen dump of the process types in the repository.

CBAM			
Processtype	File	Status	Auditlevel
COMP-ORG	PTYPCORG	Enabled	Off
ORDER	PTYPSALE	Enabled	Full

Figure 122. CBAM Screen Dump

With the cursor under process type COMP-ORG, we pressed the Enter key.

Figure 123 shows the CBAM screen dump of the process type COMP-ORG in the repository.

CBAM		PROCESSTYPE COMP-ORG		
PROCESS	MODE	COMP	SUSP	CONTS
EMPL000028A	COMPLETE	ABEND	NO	-

Figure 123. CBAM Processtype Screen Dump

With the cursor under process EMPL000028A, we pressed the Enter key.

Figure 124 shows the CBAM screen dump of the activity of the process EMPL000028A in the repository.

CBAM		PROCESSTYPE COMP-ORG		
PROCESS	ACTIVITY	MODE	COMP	SUSP
EMPL000028A	DFHROOT	COMPLETE	ABEND	NO

Figure 124. CBAM Process Screen Dump

With the cursor under activity DFHROOT, we pressed the Enter key.

Figure 125 on page 253 shows the CBAM screen dump of the ROOT activity of the process EMPL000028A in the repository.

```

CBAM          PROCESS EMPL000028A          PROCESSTYPE COMP-ORG

ACTIVITY DFHROOT

PROGRAM      EMPLROOT
TRANSID     EMPR
USERID      CICSUSER

CONTAINERS
EVENTS
TIMERS
  
```

Figure 125. CBAM Activity Screen Dump

4. RUN ACQPROCESS failed with Resp(108) resp2(27) without the first level activity program coded.

Figure 126 shows the CBAM screen dump of the activities of the process EMPL000042I in the repository.

```

CBAM          PROCESS EMPL000042I          PROCESSTYPE COMP-ORG

ACTIVITY                                     MODE      COMP      SUSP
DFHROOT                                     COMPLETE  ABEND     NO
NEWEMPL                                     COMPLETE  ABEND     NO
  
```

Figure 126. CBAM Process Screen Dump

With the cursor under activity DFHROOT, we pressed the Enter key.

Figure 127 shows the CBAM screen dump of the ROOT activity of the process EMPL000042I in the repository.

```

CBAM          PROCESS EMPL000042I          PROCESSTYPE COMP-ORG

ACTIVITY DFHROOT

PROGRAM      EMPLROOT
TRANSID     EMPR
USERID      CICSUSER

CONTAINERS
EVENTS
TIMERS
  
```

Figure 127. CBAM Activity Screen Dump

With the cursor under EVENTS, we pressed the Enter key.

Figure 128 on page 254 shows the CBAM screen dump of the ROOT activity's events of the process EMPL000042I in the repository.

```

CBAM          PROCESS EMPL000042I          PROCESSTYPE COMP-ORG
ACTIVITY DFHROOT
EVENT          TYPE          FIRED  COMPOSITE    TIMER
DFHINITIAL     SYSTEM       NO
EMPL-EVENT     ACTIVITY    NO      JD-EMPL-COMPLETE
JD-EMPL-COMPLETE COMPOSITE NO      AND
NEW-EMPL-IEVENT INPUT       NO

```

Figure 128. CBAM Event Screen Dump

With the cursor under activity NEWEMPL of process EMPL000042I, we pressed the Enter key.

Figure 129 shows the CBAM screen dump of the NEWEMPL activity of the process EMPL000042I in the repository.

```

CBAM          PROCESS EMPL000042I          PROCESSTYPE COMP-ORG
ACTIVITY NEWEMPL
PROGRAM        EMPLACTM
TRANSID       EMPN
USERID       CY-KJ-UC
CONTAINERS
EVENTS
TIMERS

```

Figure 129. CBAM Activity Screen Dump

- Feb 26, 1999
  1. The process abended with AEZJ abend code.

Figure 130 gives the detailed description of the abend code AEZJ from the manual.

```

AEZJ
EXPLANATION: CONTAINERERR condition not handled.
This is one of a number of abends issued by the EXEC
interface program. Because of their similar characteristics
these abends are described as a group.
See the description of abend AEIA for further details.
MODULE: DFHEIP

```

Figure 130. AEZJ Explanation

The following shows the CBAM screen displays of the process in the repository.



Figure 131 on page 255 shows the CBAM screen dump of the activities of the process EMPL000503C in the repository.

CBAM	Process EMPL000503C	Processtype COMP-ORG		
Activity		Mode	Comp	Susp
DFHROOT		Complete	Abend	No
NEWEMPL		Complete	Abend	No
EMPLTASK		Complete	Normal	No
EMPLROLE		Complete	Normal	No

Figure 131. CBAM Process Screen Dump

With the cursor under activity DFHROOT, we pressed the Enter key.

Figure 132 shows the CBAM screen dump of the ROOT activity of the process EMPL000503C in the repository.

CBAM	Process EMPL000503C	Processtype COMP-ORG		
Activity	DFHROOT			
Program	EMPLROOT			
Transid	EMPR			
Userid	CICSUSER			
Containers				
Events				
Timers				

Figure 132. CBAM Activity Screen Dump

With the cursor under Events, we pressed the Enter key.

Figure 133 shows the CBAM screen dump of the ROOT activity's events of the process EMPL000503C in the repository.

CBAM	Process EMPL000503C	Processtype COMP-ORG		
Activity	DFHROOT			
Event	Type	Fired	Composite	Timer
DFHINITIAL	System	No		
EMPL-EVENT	Activity	No	JD-EMPL-COMPLETE	
JD-EMPL-COMPLETE	Composite	No	AND	
NEW-EMPL-IEVENT	Input	No		

Figure 133. CBAM Event Screen Dump

With the cursor under activity NEWEMPL, we pressed the Enter key (see Figure 134 on page 256).

```

CBAM          Process EMPL000503C          Processtype COMP-ORG

Activity NEWEMPL

Program      EEMPLACTM
Transid     EMPN
Userid      CY-KJ-UC

Containers
Events
Timers

```

Figure 134. CBAM Activity Screen Dump

With the cursor under activity Events, we pressed the Enter key (see Figure 135).

```

CBAM          Process EMPL000503C          Processtype COMP-ORG

Activity NEWEMPL

Event        Type      Fired Composite      Timer

DFHINITIAL   System   No
EMPL-ROLE-EVENT Activity Yes   EEMPLREL-COMplete
EMPL-TASK-EVENT Activity No   EEMPLREL-COMplete
EEMPLREL-COMplete Composite No   AND

```

Figure 135. CBAM Event Screen Dump

With the cursor under activity EEMPLTASK of process EMPL000503C, we pressed the Enter key (see Figure 136).

```

CBAM          Process EMPL000503C          Processtype COMP-ORG

Activity EEMPLTASK

Program      EEMPLACTT
Transid     EEMPT
Userid      CICSUSER

Containers
Events
Timers

```

Figure 136. CBAM Activity Screen Dump

With the cursor under Events, we pressed the Enter key (see Figure 137 on page 257).

```

CBAM          Process EMPL000503C          Processtype COMP-ORG
Activity EMPLTASK
Event          Type      Fired Composite      Timer
DFHINITIAL    System    No

```

Figure 137. CBAM Event Screen Dump

With the cursor under Activity EMPLROLE of process EMPL000503C, we pressed the Enter key (see Figure 138).

```

CBAM          Process EMPL000503C          Processtype COMP-ORG
Activity EMPLROLE
Program      EMPLACTR
Transid      EMPO
Userid       CICSUSER

Containers
Events
Timers

```

Figure 138. CBAM Activity Screen Dump

With the cursor under Events, we pressed the Enter key (see Figure 139).

```

CBAM          Process EMPL000503C          Processtype COMP-ORG
Activity EMPLROLE
Event          Type      Fired Composite      Timer
DFHINITIAL    System    No

```

Figure 139. CBAM Event Screen Dump

We turned on the CEDX on transaction EMPN to trap the first activity program. We cleared the screen and typed CEDX EMPN.

This response appeared on the screen: TRANSACTION EMPN: EDF ON

With CEDX EMPN and CEDF on, it locates the Get Container(empl-level1-con) and returns with container error RESP(110) RESP2(10)

We fixed these problems by coding the right PUT containers (root and empl-level1-con) statements in the right place. This illustrates that the container management in coding is extremely crucial.

2. Unfortunately, the business transaction still abended.

We look at the following CBAM screen display (see Figure 140 on page 258).

CBAM	Process EMPL0006094	Processtype COMP-ORG		
Activity		Mode	Comp	Susp
DFHROOT		Complete	Abend	No
NEWEMPL		Complete	Normal	No

Figure 140. CBAM Process Screen Dump

With the cursor under activity DFHROOT of process EMPL0006094, we pressed the Enter key (see Figure 141).

CBAM	Process EMPL0006094	Processtype COMP-ORG		
Activity	DFHROOT			
Program	EMPLROOT			
Transid	EMPR			
Userid	CICSUSER			
Containers				
Events				
Timers				

Figure 141. CBAM Activity Screen Dump

With the cursor under Containers, we pressed the Enter key (see Figure 142).

CBAM	Process EMPL0006094	Processtype COMP-ORG		
Activity	DFHROOT			
Container	Datalength			
ROOT-CONTAINER	4			

Figure 142. CBAM Container Screen Dump

With the cursor under Events of the DFHROOT activity, we pressed the Enter key (see Figure 143).

CBAM	Process EMPL0006094	Processtype COMP-ORG		
Activity	DFHROOT			
Event	Type	Fired	Composite	Timer
DFHINITIAL	System	No		
JD-EMPL-COMPLETE	Composite	Yes	AND	
NEW-EMPL-IEVENT	Input	No		

Figure 143. CBAM Event Screen Dump

Using CEDX EMPR, we trapped the source of the problem: The program keeps retrieving the subevent (see Figure 144 on page 259, Figure 145 on page 259, and Figure 146 on page 259).

```

TRANSACTION: EMPR PROGRAM: EMPLROOT TASK: 0007138 APPLID: SCSCPA1 DISPLAY: 00
STATUS:  COMMAND EXECUTION COMPLETE
EXEC CICS RETRIEVE SUBEVENT
SUBEVENT ('.....')
EVENT ('JD-EMPL-COMPLETE')
EVENTTYPE (0)
NOHANDLE

OFFSET:X'000A3C'   LINE:00201           EIBFN=X'3612'
RESPONSE: END                                           EIBRESP=83

EIBRESP           = 83  END
EIBRESP2          = 10
  
```

Figure 144. SUBEVENT Abend

CBAM	Process EMPL0007305	Processtype COMP-ORG		
Activity		Mode	Comp	Susp
DFHROOT		Dormant	Incomplete	No
NEWEMPL		Complete	Normal	No

Figure 145. CBAM Process Screen Dump

With the cursor under activity DFHROOT, we pressed the Enter key.

CBAM	Process EMPL0007305	Processtype COMP-ORG		
Activity	DFHROOT			
Program	EMPLROOT			
Transid	EMPR			
Userid	CICSUSER			
Containers				
Events				
Timers				

Figure 146. CBAM Activity Screen Dump

With the cursor under Containers, we pressed the Enter key (see Figure 147 on page 260).

CBAM	Process EMPL0007305	Processtype COMP-ORG
Activity DFHROOT		
Container	Datalength	
ROOT-CONTAINER	4	

Figure 147. CBAM Container Screen Dump

With the cursor under Events, we pressed the Enter key (see Figure 148).

CBAM	Process EMPL0007305	Processtype COMP-ORG
Activity DFHROOT		
Event	Type	Fired Composite Timer
DFHINITIAL	System	No
NEW-EMPL-IEVENT	Input	No

Figure 148. CBAM Event Screen Dump

We fixed the logic error (Job Description is an optional activity to be run). The OK switch was not set to value Y. After fixing the problem, the following shows on the CBAM screen (Figure 149).

CBAM	Process EMPL0007305	Processtype COMP-ORG
Activity	Mode	Comp Susp
DFHROOT	Complete	Incomplete No
NEWEMPL	Complete	Normal No

Figure 149. CBAM Process Screen Dump

The DFHROOT activity is incomplete due to the input-event created and not fired.

- March 1, 1999
  1. When we added the activity EMPLCOSC in the flow, the business process abended with the following CBAM screen displays (see Figure 150 on page 261).

CBAM	Process EMPL0008219	Processtype COMP-ORG		
Activity		Mode	Comp	Susp
DFHROOT		Complete	Abend	No
NEWEMPL		Complete	Unexpected	No
EMPLCOSC		Complete	Forced	No
EMPLTASK		Complete	Normal	No
EMPLROLE		Complete	Normal	No

Figure 150. CBAM Process Screen Dump

With the cursor under activity DFHROOT, we pressed the ENTER key (see Figure 151).

CBAM	Process EMPL0008219	Processtype COMP-ORG		
Activity	DFHROOT			
Program	EMPLROOT			
Transid	EMPR			
Userid	CICSUSER			
Containers				
Events				
Timers				

Figure 151. CBAM Activity Screen Dump

With the cursor under Events, we pressed the ENTER key (see Figure 152).

CBAM	Process EMPL0008219	Processtype COMP-ORG		
Activity	DFHROOT			
Event	Type	Fired	Composite	Timer
DFHINITIAL	System	No		
JD-EMPL-COMPLETE	Composite	Yes	AND	
NEW-EMPL-IEVENT	Input	No		

Figure 152. CBAM Event Screen Dump

With the cursor under activity NEWEMPL, we pressed the ENTER key (see Figure 153 on page 262).

```

CBAM          Process EMPL0008219          Processtype COMP-ORG

Activity NEWEMPL

Program      EMLPACTM
Transid     EMPN
Userid      CY-KJ-UC

Containers
Events
Timers

```

Figure 153. CBAM Activity Screen Dump

With the cursor under Events, we pressed the ENTER key (see Figure 154).

```

CBAM          Process EMPL0008219          Processtype COMP-ORG

Activity NEWEMPL

Event        Type      Fired Composite      Timer

DFHINITIAL   System     No
EMPLCOSC     Activity  Yes

```

Figure 154. CBAM Event Screen Dump

Using CEDX EMPN, we trapped the source of the problem (see Figure 155).

```

TRANSACTION: EMPN PROGRAM: EMLPACTM TASK: 0008590 APPLID: SCSCPA1 DISPLAY: 00
STATUS: COMMAND EXECUTION COMPLETE
EXEC CICS DEFINE ACTIVITY
ACTIVITY ('EMPLCOSC ')
TRANSID ('EMPS')
PROGRAM ('EMLPACTS')
NOHANDLE

OFFSET:X'0010F6' LINE:00318 EIBFN=X'3402'
RESPONSE: ACTIVITYERR EIBRESP=109

EIBRESP = 109 ACTIVITYERR
EIBRESP2 = 3

```

Figure 155. Define ACTIVITY Abend

The cause of the problem is an attempt to define activity EMPLCOSC which has already been defined. The reason is the PERFORM EMPL-CS subroutine which we put in the block of handling event WHEN EMPL-REL-COMplete and PERFORM EMPL-REL-CHECK routine. Program EMLPACTM did not use the loop to handle reattachment event. It retrieved only one reattachment event per activation. Next, we moved the PERFORM EMPL-CS subroutine outside of the EVALUATE statement into



the TASK-OKAY and ROLE-OKAY block and ran this business transaction again.

- As expected, it abended again with the same CBAM screen display. The reason for this is as follows:

Before the business transaction can complete normally, an activity must have deleted all the activity completion events in its event pool. In our case, the activity EMPLCOSC's COMPSTATUS is FORCED and its parent, NEWEMPL, has the UNEXPECTED value, which leads EMPLROOT to abend the transaction.

We put the CHECK ACTIVITY statement and the EVENT handling statement in the EVALUATE block for the employee competence scheme activity. Now the business transaction works correctly.

- March 2, 1999

- We typed the job description, which is an optional field, on the screen, and the business transaction abended unexpectedly (see Figure 156).

CBAM	Process EMPL0009099	Processtype COMP-ORG		
Activity	Mode	Comp	Susp	
DFHROOT	Complete	Abend	No	
NEWJD	Complete	Abend	No	
JOBECOSC	Complete	Normal	No	
JOBEROLE	Complete	Normal	No	
JOBETASK	Complete	Normal	No	
JOBEORGU	Complete	Normal	No	
NEWEMPL	Complete	Normal	No	

Figure 156. CBAM Process Screen Dump

With the cursor under Events of the activity NEWJD which abends, we pressed the ENTER key (see Figure 157).

CBAM	Process EMPL0009099	Processtype COMP-ORG			
Activity NEWJD	Event	Type	Fired	Composite	Timer
	DFHINITIAL	System	No		
	JOBECOSC	Activity	No		
	JOBEROLE	Activity	No		

Figure 157. CBAM Event Screen Dump

We looked at the CEMT display; it had 'INVALID Retrieve subEVENT'. By looking at the program JOBEACTM, we found the event handling but no event parameter in the DEFINE ACTIVITY statements. If EVENT is not specified, the completion event is given the same name as the activity itself.

After we corrected the program, the business transaction ran correctly (see Figure 158 on page 264).

Activity	Mode	Comp	Susp
DFHROOT	Dormant	Incomplete	No
NEWJD	Complete	Normal	No
NEWEMPL	Complete	Normal	No

Figure 158. CBAM Process Screen Dump

# Appendix B. CICS BTS Commands

## B.1 Application Programming Interface (API)

Figure 159 and Figure 160 on page 266 show a summary of CICS BTS commands and their parameters.

		(more parms	ABCODE	ABPROGRAM	ABSTIME	ACQACTIVITY	ACQPROCESS	ACTIVITY	ACTIVITYID	BROWSETOKEN	COMPOSITE	COMPSTATUS	CONTAINER	DATALLENGTH	EVENT	EVENTTYPE	FACILITYTKN	FIRESTATUS	FLENGTH	FROM	INPUTEVENT	INTO	LEVEL	MODE	PREDICATE	PROCESS	PROCESSTYPE	PROGRAM	SET	STATUS	SUBEVENT	SUSPSTATUS	TIMER	TRANSID	USERID		
Output only parameters = X, in/out parameter = I		X	X	X	X	I	I	I	I	I	X	X	I	X	I	I	I	X	X	I	I	X	X	X	X	I	I	I	X	I	X	I	I	I			
ACQUIRE									O																O	O											
ADD SUBEVENT															R															R							
ASSIGN								O	O																O	O											
CANCEL						O	O	O																													
CHECK	ACQ PROCESS		O	O								R												O								O					
	ACTIVITY		O	O		O		O				R												O								O					
	TIMER																												R			R					
DEFINE	ACTIVITY							R	O						O												O						R	O			
	COMPOSITE EVENT AND														R																O						
	COMPOSITE EVENT OR														R																O						
	INPUT EVENT														R																						
	PROCESS																								R	R	O							R	O		
	TIMER	yes													O																						
DELETE	ACTIVITY							R							O																		R				
	CONTAINER					O	O	O				R														O											
	EVENT														R																						
	TIMER																																		R		
ENDBROWSE	ACTIVITY									R																											
	CONTAINER									R																											
	EVENT									R																											
	PROCESS									R																											
FORCE TIMER						O	O	O																											R		
GET CONTAINER						O	O	O																													
GETNEXT	ACTIVITY							R	O	R									O						O			O									
	CONTAINER									R			R																								
	EVENT									R	O				R	O		O								O									O		

Figure 159. CICS BTS Application Programming Interface Commands, Part 1

**CICS BTS API Commands and their parameters**

	(more parms)	ABCODE	ABPROGRAM	ABSTIME	ACACTIVITY	ACQPROCESS	ACTIVITY	ACTIVITYID	BROWSETOKEN	COMPOSITE	COMPSTATUS	CONTAINER	DATALLENGTH	EVENT	EVENTTYPE	FACILITYTKN	FIRESTATUS	FLENGTH	FROM	INPUTEVENT	INTO	LEVEL	MODE	PREDICATE	PROCESS	PROCESSTYPE	PROGRAM	SET	STATUS	SUBEVENT	SUSPSTATUS	TIMER	TRANSID	USERID			
Output only parameters = X, in/out parameter = I	X	X	X	I	I	I	I	I	X	X	I	X	I	I	I	X	I	I	I	X	X	X	X	I	I	I	I	X	I	X	I	I	I	I			
INQUIRE																																					
LINK																																					
PUT CONTAINER																																					
REMOVE SUBEVENT																																					
RESET																																					
RESUME																																					
RETRIEVE																																					
RETURN																																					
RUN SYNC																																					
RUN ASYNC																																					
START BROWSE																																					
SUSPEND																																					
TEST EVENT																																					

R = required parameter    O = optional parameter

Figure 160. CICS BTS Application Programming Interface Commands, Part 2

## B.2 System Programming Interface (SPI)

Figure 161 shows a summary of CICS BTS SPI commands and their parameters.

	(more parms)	ACTIVTD	ACTIVITY	ATTRIBUTES	ATTRLEN	AUDITLEVEL	AUDITLOG	DISABLED	ENABLED	FILE	FULL	OFF	PROCESS	PROCESSTYPE	STATUS	TASK
Output only parameters = X, in/out parameters = I		X	X	I	I	X	X	I	I	X	I	I	X	I	X	I
CREATE PROCESSTYPE	yes			R	R									R		
DISCARD PROCESSTYPE														R		
INQUIRE PROCESSTYPE						O	O			O				R	O	
INQUIRE TASK		O	O										O	O		R
SET PROCESSTYPE			O			O	O	O	O	O	O	O	O	R	O	

R = Required Parameter    O = Optional Parameter

Figure 161. CICS BTS System Programming Interface Commands

# Appendix C. UDP Process Control Catalogue

## Printout of PCC Containing Three Processes

Figure 162 through Figure 181 on page 280 show the UDP process control catalogue.

<i>Universal Process Driver - Process Control Table</i>							
<b>ProcessNumber</b> 1 = EMPL&							
<b>InActivit</b> 1 = DFHROOT							
<b>ReattachmentEvent</b> 1 = DFHINITIAL							
<i>Step</i>	<i>Call</i>	<i>CallType</i>	<i>Activity</i>	<i>Event</i>	<i>SubEvent</i>	<i>Trx</i>	<i>ProcessName</i>
10	DEFINE			NEWEMP			
	Condition:		GoTo:	/	Container	Program	UserID
19	DEFINE		NEWJD	NEWJD		JOBN	
	Condition:		GoTo:	/	Container	Program	UserID
20	RUN	ASYN	NEWJD				
	Condition:		GoTo:	/	Container	Program	UserID
29	DEFINE		NEWEMPL	NEWEMPL		EMPN	
	Condition:		GoTo:	/	Container	Program	UserID
30	RUN	ASYN	NEWEMPL				
	Condition:		GoTo:	/	Container	Program	UserID
40	DEFINE			JD-EMPL-COMPLETE			
	Condition:		GoTo:	/	Container	Program	UserID

Figure 162. UDP Process Control Catalogue, Part 1 of 20

<i>Universal Process Driver - Process Control Table</i>							
<b>ReattachmentEvent</b> 10 = JD-EMPL-COMPLETE							
<i>Step</i>	<i>Call</i>	<i>CallType</i>	<i>Activity</i>	<i>Event</i>	<i>SubEvent</i>	<i>Trx</i>	<i>ProcessName</i>
10	CHECK		NEWJD				
	Condition:		GoTo:	/	Container	Program	UserID
20	GET				JDIS		
	Condition:		GoTo:	/	Container	Program	UserID
30	CHECK		NEWEMPL				
	Condition:		GoTo:	/	Container	Program	UserID
40	GET				EMPL		
	Condition:		GoTo:	/	Container	Program	UserID
50	DELETE			JD-EMPL-COMPLETE			
	Condition:		GoTo:	/	Container	Program	UserID
60	RUN	SYNC	NEWEMJD	CONNECTEMP-JD			
	Condition:		GoTo:	/	Container	Program	UserID

Figure 163. UDP Process Control Catalogue, Part 2 of 20

## Universal Process Driver - Process Control Table

InActivit 2 = NEWJD

ReattachmentEvent l = DFHINITIAL

Step	Call	CallType	Activity	Event	SubEvent	Trx	ProcessName
10	GET						
	Condition:		GoTo:	/	Container	PROCESS	Program UserID
20	RUN		ASYN JOBEORGU	CONNECTOU			
	Condition:		GoTo:	/	Container		Program UserID
30	RUN		ASYN JOBETASK	CONNECTTASK			
	Condition:		GoTo:	/	Container		Program UserID
40	DEFINE			JD-REL-COMPLETE			
	Condition:		GoTo:	/	Container		Program UserID

Figure 164. UDP Process Control Catalogue, Part 3 of 20

## Universal Process Driver - Process Control Table

ReattachmentEvent 8 = JD-REL-COMPLETE

Step	Call	CallType	Activity	Event	SubEvent	Trx	ProcessName
20	CHECK		EMPLROLE				
	Condition:		GoTo:	/	Container		Program UserID
30	GET						
	Condition:		GoTo:	/	Container	CORG	Program UserID
40	CHECK		EMPLTASK				
	Condition:		GoTo:	/	Container		Program UserID
50	GET						
	Condition:		GoTo:	/	Container	CTSK	Program UserID
60	RUN		ASYN JOBEROLE	CONNECTROLE			
	Condition:		GoTo:	/	Container		Program UserID
70	RUN		ASYN JOBECOSC	CONNECTCS			
	Condition:		GoTo:	/	Container		Program UserID
80	DEFINE			ROLE-CS-CONNECT			
	Condition:		GoTo:	/	Container		Program UserID
90	DELETE			JD-REL-COMPLETE			
	Condition:		GoTo:	/	Container		Program UserID
100	PUT						
	Condition:		GoTo:	/	Container	JDIS	Program UserID

Figure 165. UDP Process Control Catalogue, Part 4 of 20

## Universal Process Driver - Process Control Table

**InActivit** 5 = NEWEMPL

**ReattachmentEvent** 1 = DFHINITIAL

Step	Call	CallType	Activity	Event	Sub Event	Trx	ProcessName
10	GET						
	Condition:		GoTo:	/	Container	PROCESS	Program UserID
20	RUN	ASYN	EMPLROLE	CONNECTRde			
	Condition:		GoTo:	/	Container		Program UserID
30	RUN	ASYN	EMPLTASK	CONNECTTASK			
	Condition:		GoTo:	/	Container		Program UserID
40	DEFINE			EMPLREL-COMPLETE			
	Condition:		GoTo:	/	Container		Program UserID

Figure 166. UDP Process Control Catalogue, Part 5 of 20

## Universal Process Driver - Process Control Table

**ReattachmentEvent** 7 = EMPLREL-COMPLETE

Step	Call	CallType	Activity	Event	Sub Event	Trx	ProcessName
10	CHECK		EMPLROLE				
	Condition:		GoTo:	/	Container		Program UserID
20	GET						
	Condition:		GoTo:	/	Container	CRLE	Program UserID
30	CHECK		EMPLTASK				
	Condition:		GoTo:	/	Container		Program UserID
40	GET						
	Condition:		GoTo:	/	Container	CTSK	Program UserID
50	RUN	ASYN	EMPLCOSC	CONNECTEMP-CS			
	Condition:		GoTo:	/	Container		Program UserID
60	DELETE			EMPLREL-COMPLETE			
	Condition:		GoTo:	/	Container		Program UserID
70	LINK						
	Condition:		GoTo:	/	Container		Program EMPL0002 UserID
80	PUT						
	Condition:		GoTo:	/	Container	EMPL	Program UserID

**End of Process: EMPL&**

Figure 167. UDP Process Control Catalogue, Part 6 of 20

### Universal Process Driver - Process Control Table

**ProcessNumber** 2 = NEWJD&

**InActivit** 1 = DFHROOT

**ReattachmentEvent** 1 = DFHINITIAL

Step	Call	CallType	Activity	Event	Sub Event	Trx	ProcessName
10	DEFINE		NEWJD	NEWJD		JOBN	
	Condition:		GoTo:	/	Container	Program	UserID
20	RUN	ASYN	NEWJD				
	Condition:		GoTo:	/	Container	Program	UserID
30	DEFINE		NewTask	NewTaskComplete		TASK	
	Condition:		GoTo:	/	Container	Program	UserID
40	RUN		NewTask				
	Condition:		GoTo:	/	Container	Program	UserID

Figure 168. UDP Process Control Catalogue, Part 7 of 20

### Universal Process Driver - Process Control Table

**ReattachmentEvent** 11 = NEWJD

Step	Call	CallType	Activity	Event	Sub Event	Trx	ProcessName
10	CHECK		NewTask				
	Condition:		GoTo:	/	Container	Program	UserID
20	IF NOT						
	Condition:	NORMAL	GoTo:	/ 50	Container	Program	UserID
30	LINK						
	Condition:		GoTo:	/	Container	Program	RESULT01 UserID
40	PUT				PROCESS		
	Condition:		GoTo:	/	Container	Program	UserID
50	RETURN		END				
	Condition:		GoTo:	/	Container	Program	UserID

**End of Process:** NEWJD&

Figure 169. UDP Process Control Catalogue, Part 8 of 20



**Universal Process Driver - Process Control Table**

<b>ProcessNumber</b> 3 = HOLI&							
<b>InActivit</b> 13 = Holiday							
<b>ReattachmentEvent</b> <i>l</i> = DFHINITIAL							
<i>Step</i>	<i>Call</i>	<i>CallType</i>	<i>Activity</i>	<i>Event</i>	<i>SubEvent</i>	<i>Trx</i>	<i>ProcessName</i>
10	GET						
	<i>Condition:</i>		<i>GoTo:</i>	/	<i>Container</i>	GENERAL	<i>Program</i>
							<i>UserID</i>
20	DEFINE		Flight-Book	Flight-Book		HFLY	
	<i>Condition:</i>		<i>GoTo:</i>	/	<i>Container</i>		<i>Program</i>
							<i>UserID</i>
30	RUN		ASYN Flight-Book				
	<i>Condition:</i>		<i>GoTo:</i>	/	<i>Container</i>		<i>Program</i>
							<i>UserID</i>
40	DEFINE		Hotelbook	Hotelbook		HHOT	
	<i>Condition:</i>		<i>GoTo:</i>	/	<i>Container</i>		<i>Program</i>
							<i>UserID</i>
50	RUN		ASYN Hotelbook				
	<i>Condition:</i>		<i>GoTo:</i>	/	<i>Container</i>		<i>Program</i>
							<i>UserID</i>
60	DEFINE		Carbook	Carbook		HCAR	
	<i>Condition:</i>		<i>GoTo:</i>	/	<i>Container</i>		<i>Program</i>
							<i>UserID</i>
70	RUN		ASYN Carbook				
	<i>Condition:</i>		<i>GoTo:</i>	/	<i>Container</i>		<i>Program</i>
							<i>UserID</i>
80	DEFINE		Ex&-Book	Ex&-Book		HEXC	
	<i>Condition:</i>		<i>GoTo:</i>	/	<i>Container</i>		<i>Program</i>
							<i>UserID</i>
90	RUN		ASYN Ex&-Book				
	<i>Condition:</i>		<i>GoTo:</i>	/	<i>Container</i>		<i>Program</i>
							<i>UserID</i>

Figure 170. UDP Process Control Catalogue, Part 9 of 20

**Universal Process Driver - Process Control Table**

100	LINK						
	<i>Condition:</i>		<i>GoTo:</i>	/	<i>Container</i>		<i>Program</i>
						MOREEXC	<i>UserID</i>
110	IF						
	<i>Condition:</i>	MORE	<i>GoTo:</i>	/	80 <i>Container</i>		<i>Program</i>
							<i>UserID</i>
120	DEFINE			Book-Holiday			
	<i>Condition:</i>		<i>GoTo:</i>	/	<i>Container</i>		<i>Program</i>
							<i>UserID</i>
130	DEFINE			NcResponse			
	<i>Condition:</i>		<i>GoTo:</i>	/	<i>Container</i>		<i>Program</i>
							<i>UserID</i>
140	PUT						
	<i>Condition:</i>		<i>GoTo:</i>	/	<i>Container</i>	HOLIDAY	<i>Program</i>
							<i>UserID</i>

Figure 171. UDP Process Control Catalogue, Part 10 of 20

## Universal Process Driver - Process Control Table

*ReattachmentEvent* 25 = NoResponse

<i>Step</i>	<i>Call</i>	<i>CallType</i>	<i>Activity</i>	<i>Event</i>	<i>SubEvent</i>	<i>Trx</i>	<i>ProcessName</i>
10	GET						
	<i>Condition:</i>		<i>GoTo:</i>	/	<i>Container</i>	HOLIDAY	<i>Program</i>
							<i>UserID</i>
20	TEST			Flight-Book			
	<i>Condition:</i>		<i>GoTo:</i>	/	<i>Container</i>		<i>Program</i>
							<i>UserID</i>
30	IF NOT						
	<i>Condition:</i>	NORMAL	<i>GoTo:</i>	26 / 500	<i>Container</i>		<i>Program</i>
							<i>UserID</i>
40	CHECK		Flight-Book				
	<i>Condition:</i>		<i>GoTo:</i>	/	<i>Container</i>		<i>Program</i>
							<i>UserID</i>
50	IF NOT						
	<i>Condition:</i>	COMPLETE	<i>GoTo:</i>	26 / 500	<i>Container</i>		<i>Program</i>
							<i>UserID</i>
60	GET						
	<i>Condition:</i>		<i>GoTo:</i>	/	<i>Container</i>	BOOKFL	<i>Program</i>
							<i>UserID</i>
70	LINK						
	<i>Condition:</i>		<i>GoTo:</i>	/	<i>Container</i>		<i>Program</i>
						TESTCONT	<i>UserID</i>
80	IF NOT						
	<i>Condition:</i>	OK	<i>GoTo:</i>	26 / 500	<i>Container</i>		<i>Program</i>
							<i>UserID</i>
90	TEST			Hotelbook			
	<i>Condition:</i>		<i>GoTo:</i>	/	<i>Container</i>		<i>Program</i>
							<i>UserID</i>
100	IF NOT						
	<i>Condition:</i>	NORMAL	<i>GoTo:</i>	26 / 500	<i>Container</i>		<i>Program</i>
							<i>UserID</i>

Figure 172. UDP Process Control Catalogue, Part 11 of 20

### Universal Process Driver - Process Control Table

110	CHECK	Hotelbook	Condition:	GoTo:	/	Container	Program	UserID
120	IF NOT		Condition: COMPLETE	GoTo:	26 / 500	Container	Program	UserID
130	GET		Condition:	GoTo:	/	Container	BOOKHO	Program UserID
140	LINK		Condition:	GoTo:	/	Container	Program	TESTCONT UserID
150	IF NOT		Condition: OK	GoTo:	26 / 500	Container	Program	UserID
160	CHECK	Carbook	Condition:	GoTo:	/	Container	Program	UserID
170	GET		Condition:	GoTo:	/	Container	BOOKCA	Program UserID
180	CHECK	Ex&-Book	Condition:	GoTo:	/	Container	Program	UserID
190	GET		Condition:	GoTo:	/	Container	BOOEEX&	Program UserID
200	IF		Condition: MORE	GoTo:	/ 180	Container	Program	UserID
210	DELETE	Book-Holiday	Condition:	GoTo:	/	Container	Program	UserID

Figure 173. UDP Process Control Catalogue, Part 12 of 20

### Universal Process Driver - Process Control Table

220	PUT		Condition:	GoTo:	/	Container	HOLIDAY	Program UserID
230	RETURN		Condition:	GoTo:	/	Container	Program	UserID

Figure 174. UDP Process Control Catalogue, Part 13 of 20

## Universal Process Driver - Process Control Table

ReattachmentEvent 26 = Book-Holiday

Step	Call	CallType	Activity	Event	SubEvent	Trx	ProcessName
10	DELETE			NoResponse			
	Condition:		GoTo:	/	Container	Program	UserID
20	GET						
	Condition:		GoTo:	/ 0	Container	RETRY	Program UserID
30	IF						
	Condition:	OK	GoTo:	/ 300	Container	Program	UserID
40	CHECK		Flight-Book				
	Condition:		GoTo:	/	Container	Program	UserID
50	GET						
	Condition:		GoTo:	/	Container	BOOKFL	Program UserID
60	CALL						
	Condition:		GoTo:	/	Container	Program	AGREETST UserID
70	IF NOT						
	Condition:	OK	GoTo:	/ 500	Container	Program	UserID
80	CHECK		Hotelbook				
	Condition:		GoTo:	/	Container	Program	UserID
90	IF NOT						
	Condition:	COMPLETE	GoTo:	/ 500	Container	Program	UserID
100	GET						
	Condition:		GoTo:	/	Container	BOOKHO	Program UserID

Figure 175. UDP Process Control Catalogue, Part 14 of 20

### Universal Process Driver - Process Control Table

110	CALL							
	Condition:	GoTo:	/	Container	Program	AGREETST	UserID	
120	IF NOT							
	Condition:	OK	GoTo:	/	500	Container	Program	UserID
130	CHECK	Carbook						
	Condition:	GoTo:	/	Container	Program		UserID	
140	IF							
	Condition:	COMPLETE	GoTo:	/	160	Container	Program	UserID
150	PUT							
	Condition:	GoTo:	/	Container	RETRY	Program	UserID	
151	GOTO							
	Condition:	GoTo:	/	170	Container	Program	UserID	
160	GET							
	Condition:	GoTo:	/	Container	BOOKCA	Program	UserID	
170	CHECK	Ex&-Book						
	Condition:	GoTo:	/	Container	Program		UserID	
180	GET							
	Condition:	GoTo:	/	Container	BOOKEX&	Program	UserID	
190	IF							
	Condition:	MORE	GoTo:	/	170	Container	Program	UserID
200	DELETE	Book-Holiday						
	Condition:	GoTo:	/	Container	Program		UserID	

Figure 176. UDP Process Control Catalogue, Part 15 of 20

### Universal Process Driver - Process Control Table

210	RETURN					
	Condition:	GoTo:	/	Container	Program	UserID
300	CHECK	Carbook				
	Condition:	GoTo:	/	Container	Program	UserID
310	GET					
	Condition:	GoTo:	/	Container	BOOKCA	Program UserID
320	GET					
	Condition:	GoTo:	/	Container	HOLIDAY	Program UserID
330	CHECK	Ex&-Cancel				
	Condition:	GoTo:	/	Container	Program	UserID
340	GET					
	Condition:	GoTo:	/	Container	BOOKEX&	Program UserID
350	IF					
	Condition:	MORE	GoTo:	/ 330	Container	Program UserID
360	DELETE			Book-Holiday		
	Condition:	GoTo:	/	Container	Program	UserID
370	RETURN					
	Condition:	GoTo:	/	Container	Program	UserID
500	DEFINE			Cancel-Holiday		
	Condition:	GoTo:	/	Container	Program	UserID
510	GET					
	Condition:	GoTo:	/	Container	BOOKFL	Program UserID

Figure 177. UDP Process Control Catalogue, Part 16 of 20

### Universal Process Driver - Process Control Table

511	CALL								
	Condition:	GoTo:	/	Container	Program	BOOKED	UserID		
512	IF NOT								
	Condition:	OK	GoTo:	/	550	Container	Program	UserID	
520	DEFINE	Flight-Cancel							
	Condition:	GoTo:	/	Container	Program	UserID			
530	RUN	ASYN Flight-Cancel							
	Condition:	GoTo:	/	Container	Program	UserID			
540	ADD SUB	Flight-Cancel							
	Condition:	GoTo:	/	Container	Program	UserID			
550	GET								
	Condition:	GoTo:	/	Container	BOOKHO	Program	UserID		
551	CALL								
	Condition:	GoTo:	/	Container	Program	BOOKED	UserID		
552	IF NOT								
	Condition:	OK	GoTo:	/	590	Container	Program	UserID	
560	DEFINE	Hotel-Cancel							
	Condition:	GoTo:	/	Container	Program	UserID			
570	RUN	ASYN Hotel-Cancel							
	Condition:	GoTo:	/	Container	Program	UserID			
580	ADD SUB	Hotel-Cancel							
	Condition:	GoTo:	/	Container	Program	UserID			

Figure 178. UDP Process Control Catalogue, Part 17 of 20

### Universal Process Driver - Process Control Table

590	GET							
	Condition:	GoTo:	/	Container	BOOKCA	Program		UserID
591	CALL							
	Condition:	GoTo:	/	Container		Program	BOOKED	UserID
592	IF NOT							
	Condition:	OK	GoTo:	/	630	Container		Program
600	DEFINE	Car-Cancel						
	Condition:	GoTo:	/	Container		Program		UserID
610	RUN	ASYN Car-Cancel						
	Condition:	GoTo:	/	Container		Program		UserID
620	ADD SUB	Car-Cancel						
	Condition:	GoTo:	/	Container		Program		UserID
630	GET							
	Condition:	GoTo:	/	Container	BOOEX&	Program		UserID
631	CALL							
	Condition:	GoTo:	/	Container		Program	BOOKED	UserID
632	IF NOT							
	Condition:	OK	GoTo:	/	670	Container		Program
640	DEFINE	Ex&-Cancel						
	Condition:	GoTo:	/	Container		Program		UserID
650	RUN	ASYN Ex&-Cancel						
	Condition:	GoTo:	/	Container		Program		UserID

Figure 179. UDP Process Control Catalogue, Part 18 of 20



### *Universal Process Driver - Process Control Table*

660	ADD SUB			Ex&-Cancel		
	<i>Condition:</i>	<i>GoTo:</i>	/	<i>Container</i>	<i>Program</i>	<i>UserID</i>
670	IF					
	<i>Condition:</i>	MORE	<i>GoTo:</i>	/ 630	<i>Container</i>	<i>Program</i>
680	DELETE			Book-Holiday		
	<i>Condition:</i>		<i>GoTo:</i>	/	<i>Container</i>	<i>Program</i>
690	PUT					
	<i>Condition:</i>		<i>GoTo:</i>	/	<i>Container</i>	HOLIDAY
					<i>Program</i>	<i>UserID</i>

Figure 180. UDP Process Control Catalogue, Part 19 of 20

## Universal Process Driver - Process Control Table

*ReattachmentEvent* 27 = Cancel-Holiday

<i>Step</i>	<i>Call</i>	<i>CallType</i>	<i>Activity</i>	<i>Event</i>	<i>Sub Event</i>	<i>Trx</i>	<i>ProcessName</i>
10	GET						
	<i>Condition:</i>		<i>GoTo:</i>	/	<i>Container</i>	HOLIDAY	<i>Program</i>
							<i>UserID</i>
20	CHECK		Flight-Cancel				
	<i>Condition:</i>		<i>GoTo:</i>	/	<i>Container</i>		<i>Program</i>
							<i>UserID</i>
30	CHECK		Hotel-Cancel				
	<i>Condition:</i>		<i>GoTo:</i>	/	<i>Container</i>		<i>Program</i>
							<i>UserID</i>
40	CHECK		Car-Cancel				
	<i>Condition:</i>		<i>GoTo:</i>	/	<i>Container</i>		<i>Program</i>
							<i>UserID</i>
50	CHECK		Ex&-Cancel				
	<i>Condition:</i>		<i>GoTo:</i>	/	<i>Container</i>		<i>Program</i>
							<i>UserID</i>
60	IF						
	<i>Condition:</i>	MORE	<i>GoTo:</i>	/	50 <i>Container</i>		<i>Program</i>
							<i>UserID</i>
70	DELETE			Cancel-Holiday			
	<i>Condition:</i>		<i>GoTo:</i>	/	<i>Container</i>		<i>Program</i>
							<i>UserID</i>
80	PUT						
	<i>Condition:</i>		<i>GoTo:</i>	/	<i>Container</i>	HOLIDAY	<i>Program</i>
							<i>UserID</i>
90	RETURN						
	<i>Condition:</i>		<i>GoTo:</i>	/	<i>Container</i>		<i>Program</i>
							<i>UserID</i>

**End of Process:** HOLI&

Figure 181. UDP Process Control Catalogue, Part 20 of 20

---

## Appendix D. Glossary

### A

**ACID properties.** The term, coined by Haerder and Reuter [1983], and used by Jim Gray and Andreas Reuter to denote the properties of a transaction: <sup>1</sup>

**Atomicity** A transaction's changes to the state (of resources) are atomic: either all happen or none happen.

**Consistency** A transaction is a correct transformation of the state. The actions taken as a group do not violate any of the integrity constraints associated with the state.

**Isolation** Even though transactions execute concurrently, they appear to be serialized. In other words, it appears to each transaction that any other transaction executed either before it or after it.

**Durability** Once a transaction completes successfully (commits), its changes to the state survive failures.

**Note:** In CICS TS, the ACID properties apply to a unit of work (UOW). See also *unit of work*.

**acquired activity.** An *activity* that a program executing outside the *process* that contains the activity has gained access to, by issuing an ACQUIRE command. The activity remains acquired until the next *syncpoint* occurs.

Acquiring an activity enables the program to:

- Read and write to the activity's *data-containers*.
- Read the process data-containers of the process that contains the activity
- Issue various commands, including RUN and LINK, against the activity.

See also *acquired process*.

**acquired process.** The *process* whose *root activity* a program currently has access to.

A program acquires a process in one of two ways: either by defining it; or, if the process already exists, by issuing an ACQUIRE PROCESS command. The process remains acquired until the next *syncpoint* occurs.

Acquiring a process enables the program to:

- Read and write to the process's *data-containers*
- Read and write to the root activity's *data-containers*
- Issue various commands, including RUN and LINK, against the process.

A program can acquire only one process (root activity) *or* one descendant activity within the same *unit of work*.

**activation.** The attachment of an *activity* to perform one of a series of processing steps. In order to perform all its processing, an activity may need to be activated several times. See *pseudoconversational*.

---

<sup>1</sup> Transaction Processing: Concepts and Techniques (1993)

**activity.** In CICS BTS, one part of a *process* managed by CICS business transaction services. Typically, an activity is part of a *business transaction*.

A program that implements an activity differs from a traditional CICS application program only in its being designed to respond to CICS BTS *events*.

**activity completion event.** An *atomic event* that fires when an *activity* completes.

**activity identifier.** A means of uniquely referring to an instance of a CICS BTS *activity*. Activity identifiers are assigned by CICS.

**activity tree.** A hierarchy of *activities*. An activity tree may be several levels deep.

**asynchronous.** Descriptive of an occurrence that happens at a time that is unrelated to the time of another occurrence. The two occurrences are mutually asynchronous. The relationship between the times at which they happen is unpredictable. Compare with *synchronous*. In CICS BTS, applies to an *activity* or program that executes independently of its initiator. The initiator does not wait for the requested activity to complete, but may be informed of its outcome later.

**atomic event.** A single low-level non-composite *event*. The types of atomic event are:

- *Activity completion*
- *Input*
- *Timer*
- *System*

See also *composite event*.

**audit trail utility.** A CICS-supplied utility program, DFHATUP, that enables you to print selected CICS BTS audit records from a specified logstream.

## B

**browse token.** Identifier of a particular browse of CICS BTS objects within a CICS region. The same token returned on a STARTBROWSE command must be supplied on the corresponding GETNEXT and ENDBROWSE commands. CICS discards it after the ENDBROWSE.

**business activity.** A set of tasks that are organized and broken down into a set of procedures to accomplish a specific goal. The distinction between a subfunction and an activity is as much a matter of interpretation as of scope (from *A Professional's Guide to Systems Analysis, ISBN 0-07-042948-0*).

**business function.** A series of related business activities involving one or more entities performed for the direct or indirect purpose of fulfilling one or more missions or objectives of the firm generating revenue for the firm, servicing the customer of the firm, producing the products and services of the firm, or managing, administering, monitoring, recording, or reporting on the activities, states, or conditions of the entities of the firm (from *A Professional's Guide to Systems Analysis, ISBN 0-07-042948-0*).

**business process.** A sequence of related activities or a sequence of related tasks that make up a business activity. These activities or tasks are usually

interdependent and there is a well-defined flow from one activity to another or from one task to another (from *A Professional's Guide to Systems Analysis*, ISBN 0-07-042948-0).

**business task.** The lowest discrete unit of discrete work that can be identified. A business activity may be composed of many business tasks. Business tasks are highly repetitive, highly formalized, and rigidly defined (from *A Professional's Guide to Systems Analysis*, ISBN 0-07-042948-0).

**business transaction.** A self-contained business function, for example, the booking of an airline ticket.

Traditionally, in CICS a business transaction might be implemented as multiple *user transactions*; the booking of the airline ticket might be undertaken by transactions that inquire about availability, reserve the seat, deal with payment, and print the ticket, for example. Using CICS BTS, a business transaction might be implemented as multiple *activities*.

Contrast with *transaction*.

## C

**child activity.** An *activity* that has been defined by another activity, its *parent*.

**CICS BTS.** CICS business transaction services.

**CICS BTS-set.** The set of CICS regions across which related *CICS BTS processes* and *activities* may execute.

**CICS business transaction services.** CICS domains that support an application programming interface (API) and services that simplify the development of *business transactions*.

**CICSplex.** (1) A CICS complex. A CICSplex consists of two or more regions that are linked using CICS intercommunication facilities. The links can be either intersystem communication (ISC) or multiregion operation (MRO) links, but within a CICSplex, are more usually MRO. Typically, a CICSplex has at least one terminal-owning region (TOR), more than one application-owning region (AOR), and may have one or more regions that own the resources that are accessed by the AORs. A CICSplex may be implemented within a *sysplex*.

(2) The largest set of CICS regions or systems to be manipulated by a single CICSplex SM entity.

**CICSplex System Manager (CICSplex SM).** An IBM CICS system-management product, provided as part of CICS Transaction Server for OS/390 Release 3, that provides a single-system image and a single point of control for one or more CICSplexes.

**compensation.** The act of modifying the effects of a *child activity*. Typically, compensation undoes the actions taken by an activity. For example, compensation for an order activity might be to cancel the order.

**compensation program.** A program that implements the *compensation* actions for an activity. It may or may not be the same program used for the activity's normal execution.

**composite event.** A *high-level event*, typically formed from the combination of two or more *atomic events*. However, composite events can be *empty*, that is, they may contain no *sub-events*.

**context-switch.** The *activation of a process or activity*:

- In a separate unit of work from the requestor.
- With the transaction attributes specified on the DEFINE PROCESS or DEFINE ACTIVITY command, rather than with those of the requesting transaction.

The relationship of the process or activity to the requestor is as between separate transactions, except that data can be passed between the two units of work.

A context-switch occurs when a process or activity is activated by a RUN command but not when it is activated by a LINK command.

## D

**data-container.** A named area of storage, maintained by CICS BTS, and used to pass data between *activities* or between different invocations of the same activity.

Each data-container is associated with an activity; it is identified by its name and by the activity for which it is a container. An activity can have any number of containers as long as they all have different names.

See also *process container*.

**defined userID.** A user identifier (userid) named on a DEFINE PROCESS or DEFINE ACTIVITY command. It specifies the userid under whose authority the *process or activity* will be run, if it is activated by a RUN command.

**Note:** If the process or activity is activated by a LINK command, it runs under the authority of the userid of the transaction that issues the LINK.

**distributed routing model.** A *peer-to-peer* dynamic routing system in which each of the participating CICS regions can be both a *routing region* and a *target region*. The distributed routing model is implemented by the *distributed routing program*.

**distributed routing program.** A CICS-supplied user-replaceable program that can be used to dynamically route:

- CICS BTS *processes* and *activities*
- Non-terminal-related EXEC CICS START requests

**dynamic routing model.** The traditional, hierarchical CICS dynamic routing system in which a single terminal-owning region (the *routing region*) routes transactions between several application-owning regions (the *target regions*). The dynamic routing model is implemented by the *dynamic routing program*.

**dynamic routing program.** A CICS-supplied user-replaceable program that can be used to dynamically route:

- Transactions started from terminals
- Transactions started by terminal-related EXEC CICS START commands
- CICS-to-CICS distributed program link (DPL) requests
- Program-link requests received from outside CICS

## E

**entity.** Any real person, place, or thing; or logical person, place, or thing that can be definitively described, and that is of immediate and/or ongoing interest to the firm as a whole or to some aspect of the firm. An entity may also be an idea, concept or convenience (from *A Professional's Guide to Systems Analysis*, ISBN 0-07-042948-0).

**entity set.** All known or suspected variants of the singular entities that make up the global set. In the entity-relationship model, the entity set is treated as if it were synonymous with the individual entities that it comprises; that is, the set is treated as if each of its component entities were defined and behaved in a similar manner (from *A Professional's Guide to Systems Analysis*, ISBN 0-07-042948-0).

**event.** A means by which CICS business transaction services inform an *activity* that an action is required or an action has completed. An activity can define events (by naming them) about which it wants to be informed. See also *atomic event*, *composite event*.

**event pool.** The set of events recognized by an *activity* (system events and user events that have been defined to it). Each activity has an event pool associated with it. An activity's event pool is initialized when the activity is created and deleted when the activity is deleted. Event-related commands, such as DEFINE INPUT EVENT and DEFINE COMPOSITE EVENT operate on the event pool associated with the *current* activity.

## F

**fire status.** A Boolean flag indicating whether or not an *event* has occurred (fired). The fire status of an event can be either FIRED (true) or NOTFIRED (false).

**flat browse.** A browse of the descendant activities of a specified process, on which each descendant activity can be returned exactly once.

**flat transaction.** A transaction that has no *nested transactions*.

## I

**input event.** An *atomic event* that can be sent to an *activity* by its parent, or from outside the *process*. It tells the activity why it has been activated.

See also *system event*.

## L

**local request queue.** A recoverable VSAM data set used to store pending CICS BTS requests, for example, *timers* and *unserviceable requests*. It is used to ensure that, if CICS fails, no pending requests are lost.

Unlike *repository* data sets, the local request queue:

- Is a mandatory CICS data set; you must define one, and only one, to each CICS region even if you don't use CICS BTS.

- Is never shared. It relates solely to requests that are issued on the local region.

## M

**mode.** The processing state of an *activity*. An activity can be in an initial, active, dormant (that is, waiting for an *event*), cancelling, or complete mode.

**MRO.** Multiregion operation.

**multiregion operation (MRO).** Communication between CICS systems without the use of SNA networking facilities. The systems must be in the same operating system; or, if the XCF access method is used, in the same MVS *sysplex*. See also *CICSplex*.

**MVS image.** A single occurrence of the MVS/ESA operating system that has the ability to process a workload. One MVS image can occupy the whole of a central processing complex (CPC), or one physical partition of a CPC, or one logical partition of a CPC that is operating in PR/SM™ mode.

**MVS sysplex.** See *sysplex*.

## N

**nested transaction.** A transaction that is started as a dependent transaction by another transaction. Together they form a tree of transactions.

**nested UOW.** An UOW (child) that is started as a dependent UOW by another UOW (parent). CICS uses nested UOW instead of nested transactions.

## P

**parallel activity.** An *activity* that is being executed at the same time as another within the same process instance. During the time that the two activities are both running, they are said to be executing in parallel.

**Parallel Sysplex.** An MVS *sysplex* where all the MVS images are linked through a coupling facility.

**parent activity.** An *activity* that starts another activity, its *child*.

**PCC.** Process Control Catalogue

**predicate.** A logical expression, typically involving *sub-events*, used to define a *composite event*. When the predicate becomes true, the composite event fires.

**process.** In CICS BTS, a collection of one or more *activities*. A process is the largest unit that CICS business transaction services can work with and has a unique name by which it can be referenced and invoked.

Typically, a process is an instance of a *business transaction*.



**process container.** A *data-container* associated with a *process*. Process containers can be read by all the *activities* that make up the process. Note that they are not the same as the root activity's containers.

**Process Control Catalogue (PCC).** A set of tables that contain all data to control the flow of processes specified with these tables. The catalogue is read by the *Universal Process Driver (UPD)* which runs the processes.

**process-type.** The category to which a process belongs. All the activities in a process inherit the same process-type attribute.

Categorizing processes makes it easier to find a particular process or activity; the CICS BTS browsing commands allow filtering by process type.

**pseudo-conversational.** A type of CICS application design that appears to a terminal user as a continuous conversation but consists internally of multiple transactions. In CICS BTS, the way in which an *activity* can be reattached (reactivated) when a predefined *event* occurs in order to take the next in a set of processing steps. See *activation*.

## R

**reattachment event.** An *event* whose firing has caused an *activity* to be activated.

**reattachment queue.** A list of the *reattachment events* that have caused a particular *activity* to be activated. Each activity has a reattachment queue associated with it. The queue may be empty.

Events remain on the reattachment queue until they are retrieved by the activity or until a syncpoint occurs.

**repository.** A VSAM data set on which the states of CICS BTS *processes* are stored.

When a process is not executing under the control of CICS BTS, its state (and the states of its constituent *activities*) are preserved by being written to a repository data set. The states of all processes of a particular *process-type* (and of their activity instances) are stored on the same repository data set. Records for multiple process-types can be written to the same repository.

**repository utility.** A CICS-supplied utility program, DFHBARUP, that enables you to print selected records from a specified CICS BTS *repository* data set.

**requesting region.** In CICS BTS, the CICS region on which the request to run a *process* or *activity* is issued. To be eligible for dynamic routing, the process or activity must be started by an EXEC CICS RUN ASYNCHRONOUS command.

Compare with *routing region* and *target region*.

**root activity.** The *activity* at the top of an *activity tree*. It has no *parent activity*. A *process* always contains a root activity.

**routing region.** In the dynamic routing of CICS BTS *processes* and *activities*, the CICS region on which the *distributed routing program* runs. In CICS BTS routing, the routing region is the same as the *requesting region*. See also *target region*.

## S

**stuck process.** A *process* that cannot proceed because it is waiting for an event that cannot, or does not, occur.

**sub-event.** An *atomic event* that has been added to a *composite event*.

**sub-event queue.** A list of the *sub-events* of a particular *composite event* that have fired. Each composite event has a sub-event queue associated with it. The queue may be empty.

Sub-events remain on the sub-event queue until they are retrieved, or until a syncpoint occurs.

**syncpoint.** Synchronization point. An intermediate point in an application program at which updates or modifications are logically complete.

**synchronous.** Descriptive of an occurrence that happens or exists at precisely the same time as another occurrence. Descriptive of an operation that occurs regularly or predictably with regard to the occurrence of a specified operation in another process; for example, the calling of an input/output routine that receives control at a precoded location in a program. Contrast with *asynchronous*.

**sysplex.** A systems complex consisting of multiple MVS images coupled together by hardware elements and software services. When multiple MVS images are coupled using the OS/390 cross-system coupling facility (XCF), which provides the services to form a sysplex, they can be viewed as a single entity. Compare with *CICSplex*.

**system event.** A type of *input event* that is triggered by CICS BTS's internal processing. For example, issuing a RUN command against an *activity* for the first time in a process instance triggers a DFHINITIAL system event.

## T

**target region.** In CICS BTS, the CICS region on which a routed *process* or *activity* executes. Compare with *requesting region* and *routing region*.

**task.** (1) A unit of work for the processor; therefore, the basic multiprogramming unit under the control program. (CICS runs as a task under MVS/ESA.) (2) Under CICS, the execution of a transaction for a particular user. Contrast with *transaction*.

**timer.** A CICS BTS object that expires when the system time becomes greater than a specified time or after a specified period has elapsed.

When you define a timer, a *timer event* is automatically associated with it. When the timer expires, its associated event fires.

**timer event.** An *atomic event* that fires when its associated *timer* expires.

**transaction.** A transaction can be regarded as a unit of processing (consisting of one or more application programs) initiated by a single request. A transaction may require the initiation of one or more *tasks* for its execution. Contrast with *business transaction*.

## U

**unit of work.** A sequence of processing actions (database changes, for example) that must be completed before any of the individual actions performed by a transaction can be regarded as committed. Once changes are committed (by successful completion of the UOW and recording of the syncpoint on the system log), they become durable and are not backed out in the event of a subsequent failure of the task or system.

The beginning and end of the sequence may be marked by:

- Start and end of transaction, when there are no intervening syncpoints
- Start of task and a syncpoint
- A syncpoint and end of task
- Two syncpoints.

Thus a UOW is completed when a transaction takes a syncpoint, which occurs either when a transaction issues an explicit syncpoint request, or when CICS takes an implicit syncpoint at the end of the transaction. In the absence of user syncpoints explicitly taken within the transaction, the entire transaction is one UOW.

**unserviceable request.** A request to run an *activation* of an *activity* that currently cannot be satisfied either because the activity is not available or because the region on which the request must run is inaccessible.

**Universal Process Driver.** An application that is able to run processes defined in a *Process Control Catalogue (PCC)*. It can be understood as a generic parent activity that gets the information how to run its work from the catalogue. It is driven only by data.

**UPD.** Universal Process Driver

**UOW.** Unit of work.

**user-defined event.** An *event* defined by the BTS application programmer. The CICS BTS user-defined events are:

- *Activity completion events*
- *Input events*
- *Timer events-see timer*
- *Composite events*

Compare with *system event*.

**user-related activity.** An *activity* that requires human involvement. Such an activity cannot be started automatically by CICS BTS because it is dependent on a user being ready to process the work.

**user transaction.** A user-written transaction. See *transaction*.



---

## Appendix E. Special Notices

This publication is intended to help customers implementing CICS Transaction Server for OS/390 Version 1 Release 3 and CICS Business Transaction Services provided by this release. The information in this publication is not intended as the specification of any programming interfaces that are provided by CICS Transaction Server for OS/390 Version 1 Release 3. See the PUBLICATIONS section of the IBM Programming Announcement for CICS Transaction Server for OS/390 Version 1 Release 3 for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

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

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

Any performance data contained in this document was determined in a controlled environment, and therefore, the results that may be obtained in other operating environments may vary significantly. Users of this document should verify the applicable data for their specific environment.

Reference to PTF numbers that have not been released through the normal distribution process does not imply general availability. The purpose of including these reference numbers is to alert IBM customers to specific information relative to the implementation of the PTF when it becomes available to each customer according to the normal IBM PTF distribution process.

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

ACF/VTAM	AIX
APPN	AS/400
C Set ++	C++/MVS
C/MVS	C/370
CICS	CICS OS/2
CICS/ESA	CICS/MVS
CICS/VSE	CICS/400
CICS/6000	COBOL/370
DATABASE 2	DB2
DFSMS	DFSMS/MVS
DFSMSdfp	DFSMSdss
DFSMShsm	DFSMSrmm
ES/3090	ES/4381
ES/9000	ESA/390
ESCON	IBM
IMS	IMS/ESA
MQSeries	MVS/DFP
MVS/ESA	NetView
OpenEdition	Operating System/2
OS/2	OS/390
RACF	RAMAC
S/390	S/390 Parallel Enterprise Server
SystemView	VTAM

The following terms are trademarks of other companies:

C-bus is a trademark of Corollary, Inc.

Java and HotJava are trademarks of Sun Microsystems, Incorporated.

Microsoft, Windows, Windows NT, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

Pentium, MMX, ProShare, LANDesk, and ActionMedia are trademarks or registered trademarks of Intel Corporation in the U.S. and other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

SET and the SET logo are trademarks owned by SET Secure Electronic Transaction LLC.

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





---

## Appendix F. Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

---

### F.1 International Technical Support Organization Publications

For information on ordering these ITSO publications see "How to Get ITSO Redbooks" on page 297.

- *CICS Transaction Server for OS/390: Version 1 Release 3 Implementation Guide*, SG24-5274
- *CICSplex SM Business Application Services: A New Solution to CICS Resource Management*, SG24-5267
- *Java Application Development for CICS: Base Services and CORBA Client Support*, SG24-5275
- *Revealed! CICS Transaction Gateway with More CICS Clients Unmasked*, SG24-5277
- *Developing Distributed Transaction Applications with Encina*, SG24-5241
- *Object Technology in Application Development*, SG24-4290

---

### F.2 Redbooks on CD-ROMs

Redbooks are also available on the following CD-ROMs. Click the CD-ROMs button at <http://www.redbooks.ibm.com/> for information about all the CD-ROMs offered, updates and formats.

CD-ROM Title	Collection Kit Number
System/390 Redbooks Collection	SK2T-2177
Networking and Systems Management Redbooks Collection	SK2T-6022
Transaction Processing and Data Management Redbooks Collection	SK2T-8038
Lotus Redbooks Collection	SK2T-8039
Tivoli Redbooks Collection	SK2T-8044
AS/400 Redbooks Collection	SK2T-2849
Netfinity Hardware and Software Redbooks Collection	SK2T-8046
RS/6000 Redbooks Collection (BkMgr Format)	SK2T-8040
RS/6000 Redbooks Collection (PDF Format)	SK2T-8043
Application Development Redbooks Collection	SK2T-8037

---

### F.3 Other Publications

These publications are also relevant as further information sources:

- *CICS Release Guide*, GC34-5352
- *CICS Business Transaction Services*, SC34-5268
- *CICS Application Programming Reference*, SC33-1688
- *CICS Application Programming Guide*, SC33-1687
- *CICS Shared Data Tables Guide*, SC33-1702
- *CICS System Definition Guide*, SC33-1682

- *CICS Resource Definition Guide*, SC33-1684
- *CICS Intercommunication Guide*, SC33-1695
- *CICS Messages and Codes*, GC33-1694
- *CICS Operations and Utilities Guide*, SC33-1685
- *CICS Supplied Transactions*, SC33-1686
- *CICS Recovery and Restart Guide*, SC33-1698
- *CICS Internet Guide*, SC34-5445
- *Setting Up a Sysplex*, GC28-1779
- *CICSplex System Manager for MVS/ESA Concepts and Planning*, GC33-0786
- *CICSplex System Manager for MVS/ESA Administration*, SC34-5401
- *CICSplex System Manager for MVS/ESA Managing Business Applications*, SC33-1809
- *CICSplex System Manager for MVS/ESA User Interface Guide*, SC33-0788
- *CICSplex System Manager for MVS/ESA Operations Reference*, SC33-0789
- *CICSplex System Manager for MVS/ESA Application Programming Guide*, SC34-5457
- *CICSplex System Manager for MVS/ESA Application Programming Reference*, SC34-5458
- *CICSplex System Manager for MVS/ESA Resource Table Reference*, SC33-1220
- *CICSplex System Manager for MVS/ESA Messages and Codes*, GC33-0790
- *A Professional's Guide to Systems Analysis, Second Edition* , Martin E. Modell, McGraw-Hill, 1996, ISBN 0-07-042948-0
- *Transaction Processing: Concepts and Techniques* , Jim Gray and Andreas Reuter, Morgan Kaufmann Publishers, 1993 ISBN 1-55860-190-2
- *Principles of Transaction Processing for the Systems Professional* , Philip A. Bernstein and Eric Newcomer, Morgan Kaufmann Publishers, 1997 ISBN 1-55860-415-4
- *Object-Oriented SSADM* , Keith Robinson and Graham Berisford, Prentice Hall International, 1994 ISBN 0-13-309444-8
- *An Introduction to Parallel Programming* , K. Mani Chandy and Stephen Taylor, Jones and Bartlett Publishers, 1992 ISBN 0-86720-208-4
- *Design Patterns for Object-Oriented Software Development* , Wolfgang Pree, Addison-Wesley, 1995, ISBN 0-201-42294-8

---

## How to Get ITSO Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, redpieces, and CD-ROMs. A form for ordering books and CD-ROMs by fax or e-mail is also provided.

- **Redbooks Web Site** <http://www.redbooks.ibm.com/>

Search for, view, download, or order hardcopy/CD-ROMs redbooks from the redbooks Web site. Also read redpieces and download additional materials (code samples or diskette/CD-ROM images) from this redbooks site.

Redpieces are redbooks in progress; not all redbooks become redpieces and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

- **E-mail Orders**

Send orders by e-mail including information from the redbook fax order form to:

In United States:  
Outside North America:

e-mail address: [usib6fpl@ibmmail.com](mailto:usib6fpl@ibmmail.com)  
Contact information is in the "How to Order" section at this site:  
<http://www.elink.ibmmlink.ibm.com/pbl/pbl/>

- **Telephone Orders**

United States (toll free)  
Canada (toll free)  
Outside North America

1-800-879-2755  
1-800-IBM-4YOU  
Country coordinator phone number is in the "How to Order" section at this site:  
<http://www.elink.ibmmlink.ibm.com/pbl/pbl/>

- **Fax Orders**

United States (toll free)  
Canada  
Outside North America

1-800-445-9269  
1-403-267-4455  
Fax phone number is in the "How to Order" section at this site:  
<http://www.elink.ibmmlink.ibm.com/pbl/pbl/>

This information was current at the time of publication, but is continually subject to change. The latest information may be found at the redbooks Web site.

### IBM Intranet for Employees

IBM employees may register for information on workshops, residencies, and redbooks by accessing the IBM Intranet Web site at <http://w3.itso.ibm.com/> and clicking the ITSO Mailing List button. Look in the Materials repository for workshops, presentations, papers, and Web pages developed and written by the ITSO technical professionals; click the Additional Materials button. Employees may access MyNews at <http://w3.ibm.com/> for redbook, residency, and workshop announcements.

---

# IBM Redbook Fax Order Form

Please send me the following:

Title	Order Number	Quantity
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____

---

First name \_\_\_\_\_ Last name \_\_\_\_\_

Company \_\_\_\_\_

Address \_\_\_\_\_

City \_\_\_\_\_ Postal code \_\_\_\_\_ Country \_\_\_\_\_

Telephone number \_\_\_\_\_ Telefax number \_\_\_\_\_ VAT number \_\_\_\_\_

Invoice to customer number \_\_\_\_\_

Credit card number \_\_\_\_\_

---

Credit card expiration date \_\_\_\_\_ Card issued to \_\_\_\_\_ Signature \_\_\_\_\_

**We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.**

---

## List of Abbreviations

<b>ACP</b>	autoinstall control program	<b>DOR</b>	data owning region
<b>AOR</b>	application owning region	<b>DPL</b>	distributed program link
<b>API</b>	application programming interface	<b>DSA</b>	dynamic storage area
<b>APPC</b>	Advanced Program-to-Program Communication	<b>ECB</b>	event control block
<b>ARM</b>	automatic restart manager	<b>ECI</b>	external call interface
<b>AXM</b>	authorized cross-memory	<b>EIP</b>	EXEC interface program
<b>BAS</b>	business application services	<b>EOV</b>	end of volume
<b>BMS</b>	basic mapping support	<b>EPI</b>	external presentation interface
<b>BTS</b>	business transaction services	<b>ESDS</b>	entry sequenced data set
<b>BWO</b>	backup while open	<b>ESSS</b>	environment services system services
<b>CAS</b>	coordinating address space	<b>EXCI</b>	external CICS interface
<b>CICS</b>	Customer Information Control System	<b>FOR</b>	file owning region
<b>CICS TS</b>	Customer Information Control System Transaction Server	<b>GEM</b>	global enterprise manager
<b>CICSVR</b>	CICS VSAM Recovery	<b>GLUE</b>	global user exit
<b>CF</b>	coupling facility	<b>GRS</b>	generalized serialization services
<b>CFDT</b>	coupling facility data table	<b>GTF</b>	generalized trace facility
<b>CFRM</b>	coupling facility resource management	<b>HFS</b>	hierarchical file system
<b>CMAS</b>	CICSplex SM address space	<b>HLL</b>	high level language
<b>CMOS</b>	complementary metal oxide semiconductor	<b>HTTP</b>	hypertext transfer protocol
<b>CMT</b>	CICS-managed table	<b>IBM</b>	International Business Machines Corporation
<b>CORBA</b>	Common Object Request Broker Architecture	<b>ICF</b>	Integrated Catalog Facility (VSAM catalog)
<b>CSA</b>	common system area	<b>IIOF</b>	Internet Inter-ORB Protocol
<b>CSD</b>	CICS system definition	<b>IMS</b>	Information Management System
<b>CWA</b>	common work area	<b>IPL</b>	initial program load
<b>CWI</b>	CICS Web Interface	<b>IRC</b>	interregion communication
<b>DASD</b>	direct access storage device	<b>ISC</b>	intersystem communication
<b>DB2</b>	DATABASE 2	<b>ISPF</b>	Interactive System Productivity Facility
<b>DBCTL</b>	database control	<b>IT</b>	information technology
<b>DBRC</b>	database recovery control	<b>JDK</b>	Java Development Kit
<b>DCE</b>	Distributed Computing Environment	<b>JCT</b>	journal control table
<b>DCT</b>	destination control table	<b>JVM</b>	Java virtual machine
<b>DDL</b>	data definition language	<b>KSDS</b>	key sequenced data set
<b>DDM</b>	distributed data management	<b>LE</b>	Language Environment
<b>DFP</b>	data facility product	<b>LMAS</b>	local managed address space
<b>DML</b>	data manipulation language	<b>LPA</b>	link pack area
<b>DNS</b>	domain name server	<b>LSR</b>	local shared resources
		<b>LUW</b>	logical unit of work

<b>MAS</b>	managed address space	<b>RRMS</b>	recoverable resource management services
<b>MRO</b>	multiregion operation	<b>RTA</b>	real time analysis
<b>MVS</b>	multiple virtual storage	<b>SHCDS</b>	sharing control data set
<b>NSR</b>	nonshared resources	<b>SMS</b>	Storage Management Subsystem
<b>OLTP</b>	online transaction processing	<b>SIT</b>	system initialization table
<b>ONC RPC</b>	Open Network Computing Remote Procedure Call	<b>SMF</b>	system management facility
<b>OO</b>	Object Oriented	<b>SOS</b>	short on storage
<b>ORB</b>	Object Request Broker	<b>SPI</b>	system programming interface
<b>PDS</b>	partitioned data set	<b>TCB</b>	task control block
<b>PDSE</b>	partitioned data set extended	<b>TD</b>	transient data
<b>PLT</b>	program list table	<b>TGMC</b>	Tivoli General Manager for CICS
<b>PLTPI</b>	program list table post initialization	<b>TIOA</b>	terminal input output area
<b>PLTSD</b>	program list table shutdown	<b>TOR</b>	terminal owning region
<b>POR</b>	printer-owning region	<b>TPNS</b>	teleprocessing network simulator
<b>PTF</b>	program temporary fix	<b>TRUE</b>	task related user exit
<b>QOR</b>	queue owning region	<b>TS</b>	temporary storage
<b>RACF</b>	Resource Access Control Facility	<b>TSOR</b>	temporary storage owning region
<b>RBA</b>	relative byte address	<b>TST</b>	temporary storage table
<b>RCT</b>	resource control table	<b>TWA</b>	transaction work area
<b>RCDS</b>	recovery control data set	<b>UMT</b>	user-managed table
<b>RDO</b>	resource definition online	<b>UOW</b>	unit of work
<b>RLS</b>	record level sharing	<b>UPD</b>	universal process driver
<b>RMAS</b>	remote managed address space	<b>UR</b>	unit of recovery
<b>RMF</b>	resource management facility	<b>URM</b>	user replaceable module
<b>RODM</b>	resource object data manager	<b>VSAM</b>	Virtual Storage Access Method
<b>ROR</b>	resource-owning region	<b>WLM</b>	workload management
<b>RPC</b>	remote procedure call	<b>WWW</b>	World Wide Web
<b>RRDS</b>	relative record data set	<b>XCF</b>	cross-system coupling facility
		<b>XPI</b>	exit programming interface

---

# Index

## A

activity  
  access rules 22  
  definition 8  
  execution 17  
activity mode  
  active 23  
  cancelling 23  
  complete 23  
  dormant 23  
  initial 23  
AEZX abend code 210  
API commands  
  ACQUIRE ACTIVITYID 22  
  ACQUIRE PROCESS 22, 38, 53  
  activity handling 10  
  ADD SUBEVENT 38, 46  
  ASSIGN PROCESS 38, 45, 116  
  CANCEL ACQPROCESS 169  
  CHECK ACQACTIVITY 184  
  CHECK ACQPROCESS 125  
  CHECK ACTIVITY 18, 38, 46, 184, 216  
  CHECK TIMER 28  
  COMPOSITE INPUT EVENT 26  
  DEFINE ACTIVITY 17, 38, 45  
  DEFINE COMPOSITE EVENT 28, 38, 46, 73  
  DEFINE INPUT EVENT 26, 38, 49  
  DEFINE PROCESS 16, 22, 38, 40  
  DEFINE TIMER 26, 38, 52, 74  
  DELETE ACTIVITY 23, 90  
  DELETE CONTAINER 23, 38, 54  
  DELETE EVENT 23, 28, 38, 49, 54  
  DELETE TIMER 23, 28, 38, 52  
  ENDBROWSE ACTIVITY 90  
  ENDBROWSE EVENT 90  
  event handling 10  
  GET CONTAINER 38, 40, 80  
  GETNEXT ACTIVITY 89  
  GETNEXT EVENT 90, 125  
  GETNEXT PROCESS 124  
  INQUIRE PROCESSTYPE 124  
  INQUIRE TIMER 125  
  LINK ACQPROCESS 16  
  LINK ACTIVITY 17, 81  
  process and activity handling 10  
  process handling 10  
  PUT CONTAINER 38, 40, 81  
  RESET ACTIVITY 23, 38, 50, 185  
  RESUME ACQACTIVITY 104  
  RESUME ACQPROCESS 104, 126  
  RETRIEVE REATTACH EVENT 26, 38, 45, 53

## API commands (*continued*)

  RETRIEVE SUBEVENT 28, 30, 39, 48  
  RETURN ENDACTIVITY 39, 56, 88, 108, 142, 143, 145, 168  
  RUN ACQPROCESS 16, 39, 40, 53, 68, 108  
  RUN ACTIVITY 17, 19, 39, 45, 81  
  STARTBROWSE ACTIVITY 89  
  STARTBROWSE EVENT 89, 125  
  STARTBROWSE PROCESS 124  
  summary 10, 265  
  SUSPEND ACQACTIVITY 104  
  SUSPEND ACQPROCESS 104, 125  
  TEST EVENT 86  
  timer handling 10  
application design 129  
asynchronous execution 19  
atomized function 131  
audit log 13, 34, 211, 219, 240  
AUDITLEVEL attribute 219  
AUDITLOG attribute 219

## B

business activity 130  
business function 130  
business process 129, 130, 133  
business process analysis 129, 132  
business process model 136  
business task 131  
business transaction 14  
business transaction model 180

## C

CBAM 32, 100, 252, 253, 255, 257, 259, 260, 263  
CEMT options  
  INQUIRE PROCESSTYPE 233  
  INQUIRE TASK 32, 169, 233  
  SET PROCESSTYPE 220, 233  
CFDT 181  
child activity 8  
Company Organization sample application  
  activities  
    EMPLCOSC 141  
    EMPLROLE 139, 141  
    EMPLTASK 139, 141  
    JOBECOSC 140  
    JOBORGU 138  
    JOBEROLE 140  
    JOBETASK 138  
    NEWEMJD 144  
    NEWEMPL 139, 140, 142, 156  
    NEWJD 138, 139, 141, 161

## Company Organization sample application (*continued*)

- containers
  - EMPL 143
  - PROCESS 138
- events
  - CONNECT-EMP-CS 142
  - DFHINITIAL 137, 138
  - EMPLREL-COMPLETE 140
  - JD-EMPL-COMPLETE 143
  - JD-REL-COMPLETE 139
  - NEWEMP 144
  - ROLE-CS-CONNECT 141
- compensation 59
- container
  - activity 24
  - process 24
- context-switch 18

## D

- data-container 8
- DFHATUP 213, 217, 219, 221
- DFHBARUP 217, 227
- DFHINITIAL system event 20, 22, 26, 34, 40, 70, 91, 94, 101, 109, 137, 139, 172, 187, 198
- DFHLG520 221
- DFHLGCVN 221
- DFHROOT activity 8, 20, 38, 61, 100, 106, 137, 143, 144, 168, 220, 252, 253, 255, 258, 259, 260

## E

- ENDACTIVITY option 11
- entity relationship model 133
- event
  - atomic 27
  - completion 27
  - composite 28
  - deletion 28
  - DFHINITIAL 41
  - FIRE status 26, 28, 46
  - handling 29
  - input 27
  - NOTFIRED status 26, 28, 45, 48, 50, 101
  - system 27
  - timer 27

## H

### HOLIDAY sample application

- activities
  - BOOKCAR 69
  - BOOEXCUR1 69
  - BOOEXCUR2 69
  - BOOEXCUR3 69
  - BOOKFLIGHT 69
  - BOOKHOTEL 69

## HOLIDAY sample application (*continued*)

- activities (*continued*)
  - CANCELCAR 69
  - CANCELEXCUR1 69
  - CANCELEXCUR2 69
  - CANCELEXCUR3 69
  - CANCELFLIGHT 69
  - CANCELHOTEL 69
- CBAM displays 100
- components 61
- container layout 61
- containers
  - BOOKCA 69
  - BOOKCAR 91
  - BOOEXCUR1 69, 91
  - BOOEXCUR2 69, 91
  - BOOEXCUR3 69, 91
  - BOOKFL 69
  - BOOKFLIGHT 90
  - BOOKHO 69
  - BOOKHOTEL 90
  - GENERAL 65, 69, 90, 91, 93
  - HOLIDAY 69
  - RETRY 69
- control file 63
- events
  - BOOK-HOLIDAY 69, 74, 81
  - CANCEL-HOLIDAY 69, 86
  - CAR-BOOK 69
  - CAR-CANCEL 69
  - DFHINITIAL 65, 69, 91, 94
  - EX1-BOOK 69
  - EX1-CANCEL 69
  - EX1-EVENT 69
  - EX2-BOOK 69
  - EX2-CANCEL 69
  - EX2-EVENT 69
  - EX3-BOOK 69
  - EX3-CANCEL 69
  - EX3-EVENT 69
  - FLIGHT-BOOK 69
  - FLIGHT-CANCEL 69
  - HOTEL-BOOK 69
  - HOTEL-CANCEL 69
  - NO-RESPONSE 69, 82
- overview 59
- process initiation 65

## I

- initial request 7

## L

- local request queue 13, 208



log stream 219

## M

message mapping 173

MORTGAGE and FINANCE sample application

activities

- ANNSTAT 109
- ANNUAL-STATEMENT 109
- EARLY-SETTLEMENT 109
- HOLIDAY 109
- INTCHANGE 109
- INTEREST-CHANGED 109
- MAKEPAY 109
- MORTREQ 109
- NEWMORT 109
- NOTENDED 109
- NOTIFY-END 109
- PAYDUE 109
- PAYMENT-OVERDUE 109
- PAYREC 109
- REMDT 109
- REPOSSESS 109
- SETTEARLY 109

components 106

container layout 106

containers

- AUDIT 109
- EARLYSETT 108, 109
- GENERAL 108, 109
- MORTREQ 108
- NEWMORT 109
- PAYDUE 108, 109
- PAYREC 108, 109
- RETRY 108, 109

events

- ANNUAL-DONE 109
- ANNUAL-STATEMENT 109
- DFHINITIAL 109
- EARLY-SETT-DONE 109
- EARLY-SETTLEMENT 109
- HOLIDAY 109
- HOLIDAY-DONE 109
- INTCHANGE-DONE 109
- INTEREST-CHANGED 109
- INTEREST-CHANGED 109
- INTEREST-DONE 109
- MAKE-PAYMENT 109
- MAKEPAY-DONE 109
- MORTREQ-DONE 109
- NOTENDED 109
- NOTENDED-DONE 109
- NOTIFY-END 109
- NOTIFY-END-DONE 109
- PAYMENT-DUE 109
- PAYMENT-DUE-DONE 109
- PAYMENT-OVERDUE 109

MORTGAGE and FINANCE sample application

(continued)

events (continued)

- PAYMENT-REC-DONE 109
- PAYMENT-RECEIVED 109
- REMDT-DONE 109
- REMOVE-DET-DONE 109
- REPOSSESS 109
- RESETPAY-DONE 109
- TIMERS 109

process communication 103

## N

name scope summary 31

## O

organizational pyramid 129

## P

parent activity 8

process

- access rules 22

- definition 7

process control catalogue

- ActivityTable 195, 200

- EventTable 196, 200

- example of a process description 199

- ProcessControl 196

- ProcessControlTable 202

- ProcessTable 195, 199

- SubEventTable 196, 201

process type 8

## R

RDO resource objects

- JOURNALMODEL 13, 211, 212, 240

- PROCESSTYPE 8, 12, 40, 207, 239

repository 12, 227, 239

reuse

- 3270 bridge 171

- activity 172

- CICS BTS elements 172

- existing applications 171

- process 172

- programs 171

- reuse management 172

- transactions 171

root activity 8, 168, 186

routing of processes and activities 211

## S

### SALES sample application

#### activities

- CREDIT 41
- DELIVERY 41
- INVOICE 41
- REMINDER 41
- root 41
- STOCK 41

API commands used 38

BTS resources used 39

components 38

#### containers

- CREDIT 41
- PROCESS 41
- STOCK 41
- TEMP 41

#### events

- CBAM-INSPECT 41
- DEL-INV-COMPLETE 41, 47
- DELETE-TIMER 41, 52
- DFHINITIAL 41
- REM-COMPLETE 41, 51
- SEND-PAY-REMINDER 41, 49

overview 37

process flow 22

process initiation 39

### SPI commands

- CREATE PROCESSTYPE 12, 266
- DISCARD PROCESSTYPE 12, 266
- INQUIRE PROCESSTYPE 12, 266
- INQUIRE TASK 12, 266
- SET PROCESSTYPE 12, 266

stuck process 215, 216

synchronous execution 18

syncpoint 42

## T

timer 9

## U

### universal process driver

global description 197

model of 195

process definition 197

---

# ITSO Redbook Evaluation

Business Process Model Implementation with CICS Business Transaction Services  
SG24-5464-00

Your feedback is very important to help us maintain the quality of ITSO redbooks. **Please complete this questionnaire and return it using one of the following methods:**

- Use the online evaluation form found at <http://www.redbooks.ibm.com/>
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to [redbook@us.ibm.com](mailto:redbook@us.ibm.com)

Which of the following best describes you?

**Customer**     **Business Partner**     **Solution Developer**     **IBM employee**  
 **None of the above**

**Please rate your overall satisfaction** with this book using the scale:  
**(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)**

**Overall Satisfaction** \_\_\_\_\_

Please answer the following questions:

Was this redbook published in time for your needs?                      Yes\_\_\_\_ No\_\_\_\_

If no, please explain:

---

---

---

---

What other redbooks would you like to see published?

---

---

---

**Comments/Suggestions:**        **(THANK YOU FOR YOUR FEEDBACK!)**

---

---

---

---

---

**SG24-5464-00**  
**Printed in the U.S.A.**

**Business Process Model Implementation with CICS Business Transaction Services**

**SG24-5464-00**

