UNIVERSITY OF LEIPZIG

Department of Computer Science

Generation of a Java front end for a standalone CICS application accessed through MQSeries
&
Securing CICS with RACF

Diplomarbeit – Master Thesis

Submitted by Tobias Busse
Born: 1976/07/14
Subject: Computer Science
Matr.Nr.: 7952583

1st Examiner: Prof. Dr.-Ing. W. G. Spruth
2nd Examiner: Peter Missen

Leipzig, August 2004
EXECUTIVE SUMMARY

This master thesis deals with the design, programming, implementation and presentation of on-line business applications for IBM's On-Line Transaction Processing (OLTP) system called Customer Information Control System (CICS). According to the book “Designing and Programming CICS Applications” published by John Horswill [HOR00] we explain two out of many feasible procedures to present the functionality of CICS resp. CICS business applications.

As the main result, we create for each of both procedures a business application representing a little clip of a bank customer account program. These applications access a database-like file stored on an Operating System/390 (OS/390) server to create, read, update, and delete customer accounts. Both use CICS not only as a transaction processing system but also as an application server that manages business applications 24 hours a day for 7 days a week. The difference between the two business applications are the data transfer and information display methods. The first procedure describes how to create a business application that uses the legacy 3270 interface to inquire a customer record stored on the OS/390-server and how to display the information on a 3270 terminal screen (also called as the green screen). In contrast, the second business application uses message queuing provided by IBM's Message Oriented Middleware (MOM) product MQSeries to transfer the request data to and the response data from the customer account file between CICS and the Java Virtual Machine (JVM). This JVM runs on a WINDOWS2000 client, whereas MQSeries has to run on both – the OS/390-server and the WINDOWS2000 client. For this procedure, the so-called MQSeries CICS Bridge – an interface between CICS and MQSeries on the OS/390-server – is installed and activated. It is also described how to use the Message Queue Interface (MQI) that connects MQSeries and JAVA on the client computer system.

Hence, the business logic component is used by both sample applications, but they differ in the presentation logic components. The programs of the business logic component, written in the programming language COBOL, are responsible for the data transfer to and from the customer account file to update or read it, for the business calculations, and for the error handling. The presentation logic component of the first business application – the CICS application NACT – is also written in COBOL and use the Basic Mapping Support (BMS) to set up the 3270 interface. In the second business application called the MQSeries CICS application MQNACT the programs of the presentation logic component are written in JAVA.

Additionally, this master thesis describes the security management of CICS using IBM's External Security Manager (ESM) Resource Access Control Facility (RACF). The main part of this procedure is to allow only authorised terminal users an access to an existing CICS address space, including some CICS default user IDs as for example the CICS Region User ID (CRU), and the Default CICS User ID (DCU). Furthermore, the authorised user should/may use only those CICS resources for which a permission exists. Securing CICS resources is explained on the examples of the CICS Transactions Security mechanism and the CICS Command Security mechanism.
Acknowledgements

My thanks goes to Prof. Dr. Wilhelm G. Spruth who gave me the best support and who never despaired on my English writing.

I wish to thank Peter Missen, head of the CICS User Group at IBM Hursley UK, for helping, for redacting this text, and, of course, for working with the CICS User Group.

My technical thanks goes to Dr. Paul Herrmann who always sustained the functionality of the JEDI OS/390-server.

Many thanks goes also to:

   Nils Michaelsen for his encouragement in my work,
   Ulrike Kirsch (IBM Leipzig Germany) for her support on MQSeries,
   Sandro Varges (IBM Mainz Germany) for his support on CICS (“Haeh, What is a CSD, Sandro?”),
   Thomas Renner (IBM Leipzig Germany) for his support on RACF and JCL,
   Ingo Karge who provided for me a desk at the IBM Leipzig location,
   the CICS User Group, especially to Paul …, James Taylor, and Kevin …, and
   to everyone at IBM Leipzig.

I would also like to thank my family for their constant support.

For my lovely butterfly Ivonne Lange, the sun should always shine. Thanks for your support, for your lovely cooking, for your understanding in my work and for your +encouragement. Together we will fight the future!
TABLE OF CONTENTS

List of Figures ................................................................. xi
List of Tables ................................................................. xv
List of Listings ................................................................. xvii
List of Abbreviations ....................................................... xix
Notices ........................................................................... xxi

1 Introduction to OS/390 .................................................... 23
1.1 Introduction .................................................................. 23
1.2 History of OS/390 ........................................................ 24
1.3 OS/390 in general ....................................................... 27
1.4 OS/390 V2R7 ............................................................. 30

2 Introduction to CICS ....................................................... 33
2.1 Introduction .................................................................. 33
2.2 History of CICS ........................................................ 34
2.3 CICS in general .......................................................... 36

3 The Initial Architecture of the CICS Business Applications 39

4 The CICS Business Application NACT ................................. 41
4.1 Introduction .................................................................. 41
4.2 Uploading the files ..................................................... 42
4.3 The NACT COBOL programs and their commands ............ 49
  4.3.1 Overview ........................................................... 49
  4.3.2 The presentation logic component ............................ 49
  4.3.3 The business logic component ................................. 54
  4.3.4 Other commands that control the programs ................. 56
  4.3.5 Excursion: Storing exchange data – When to use the COMMAREA? .......... 57
    4.3.5.1 Overview ..................................................... 57
    4.3.5.2 Temporary Storage – queuing and scratchpad facility ....................... 59
    4.3.5.3 Transient Data – queuing facility ............................... 60
    4.3.5.4 COMMAREA – a scratchpad facility ............................. 60
    4.3.5.5 Common Work Area, Transaction Work Area, and Terminal User Area – other
Generation of a Java front end for a standalone CICS application accessed through MQSeries &
Securing CICS with RACF

4.4 Storing the data – account file, locking file, and name file ........................................ 68
4.4.1 File description ....................................................................................................... 68
4.4.2 Installing the storing data ......................................................................................... 70
4.5 The CICS resource definitions ....................................................................................... 74
4.5.1 Overview ............................................................................................................... 74
4.5.2 Setting up the CICS resources .................................................................................. 74

5 The MQSeries CICS Business Application MQNACT 81

5.1 Introduction ............................................................................................................. 81
5.2 Messaging and Queuing ................................................................................................ 83
5.2.1 Overview ............................................................................................................... 83
5.2.2 Excursion: MQSeries is the UK Post Office ............................................................. 84
5.2.3 Messages ................................................................................................................. 85
5.2.4 Queue manager ....................................................................................................... 87
5.2.5 Queue manager objects ........................................................................................... 88
5.2.5.1 Local and remote queues .................................................................................. 88
5.2.5.2 Message Channels ............................................................................................ 89
5.2.5.3 MQI Channels .................................................................................................... 90
5.2.6 Message Queuing Interface ..................................................................................... 91

5.3 The architecture of the MQSeries CICS application ...................................................... 92
5.4 Setting up MQSeries on the OS/390-server .................................................................. 96
5.4.1 The OS/390-server queue manager MQA1 ............................................................. 96
5.4.2 The MQSeries CICS Bridge ..................................................................................... 100
5.4.2.1 Overview ............................................................................................................ 100
5.4.2.2 Configuring CICS to use the MQSeries CICS Bridge ........................................ 101
5.4.2.3 Configuring MQSeries to use the MQSeries CICS Bridge ................................ 105
5.4.2.4 Running the MQSeries CICS Bridge .................................................................. 106
5.4.2.5 An automatic start job for the MQSeries CICS Bridge ....................................... 109
5.4.3 The required queue manager objects ...................................................................... 112
5.4.3.1 The transmission queue TBUSSE.NACT .......................................................... 112
5.4.3.2 The remote queue definition TBUSSE.NACT.REPLYQ ..................................... 116
5.4.3.3 The dead-letter queue MQA1.DEAD.QUEUE ................................................... 117
5.4.3.4 The channel sender TBUSSE.NACT.OS.WIN .................................................. 118
5.4.3.5 The channel receiver TBUSSE.NACT.WIN.OS ................................................. 120

5.5 Setting up MQSeries on the Windows2000 client ......................................................... 122
5.5.1 The WINDOWS2000-client queue manager TBUSSE.NACT .................................. 122
5.5.2 The required queue manager objects ...................................................................... 125
5.5.2.1 A definition script for the queue manager objects .............................................. 125
5.5.2.2 The transmission queue TBUSSE.NACT.XMITQ ............................................. 126
5.5.2.3 The reply-to queue TBUSSE.NACT.REPLYQ ................................................... 127
5.5.2.4 The remote queue definition TBUSSE.NACT.REMOTEQ ................................ 128
5.5.2.5 The dead letter queue TBUSSE.NACT.DEAD.LETTER.QUEUE ...................... 128
5.5.2.6 The server connection TBUSSE.NACT.CLIENT ............................................. 128
## Table of Contents

5.5.2.7 The channel sender TBUSSE.NACT.WIN.OS .................................................. 129  
5.5.2.8 The channel receiver TBUSSE.NACT.OS.WIN ............................................... 129  
5.6 Building the JAVA application ............................................................................. 132  
5.6.1 Coding the MQSeries communication logic ...................................................... 132  
5.6.1.1 Creating a connection to the WINDOWS2000-client queue manager .......... 132  
5.6.1.2 Opening the MQSeries queues for message transport .................................... 133  
5.6.1.3 Creating the request message and send it ...................................................... 134  
5.6.1.4 Receiving the response message ................................................................. 136  
5.6.1.5 Finalising the connection to the WINDOWS2000-client queue manager .......... 136  
5.6.2 Coding the presentation logic ........................................................................... 137  
5.7 Connecting both queue managers ....................................................................... 142  
5.7.1 Checking the status of the queue managers and activate services ................. 142  
5.7.2 Connecting the WINDOWS2000-client queue manager with the OS/390-server queue manager .......................................................... 144  
5.7.3 Connecting the OS/390-server queue manager with the WINDOWS2000-client queue manager .......................................................... 146  
5.8 Starting the JAVA application ............................................................................. 149  
5.9 Terminating the connection between the MQSeries servers .............................. 151  
5.10 Common MQSeries problems indicated due to this thesis ............................... 155  
5.10.1 Ghost channel connections on the OS/390-server queue manager ............... 155  
5.10.2 Resetting channel in-doubt status – The message sequence error ............... 156  

6 Securing CICS with RACF 159  
6.1 Introduction ........................................................................................................... 159  
6.2 RACF Topics ......................................................................................................... 161  
6.2.1 RACF mechanisms ......................................................................................... 161  
6.2.2 RACF commands .......................................................................................... 163  
6.2.3 Data set and general resource profiles ......................................................... 164  
6.3 Implementing RACF protection for the CICS region A06C001 ......................... 169  
6.3.1 The CICS region’s SIT .................................................................................. 169  
6.3.2 User management of the CICS region .......................................................... 171  
6.3.3 The CRU ....................................................................................................... 171  
6.3.3.1 What is it? ............................................................................................... 171  
6.3.3.2 Defining the CRU to RACF ..................................................................... 172  
6.3.3.3 Authorising the CRU to invoke CICS as a Started Job ......................... 173  
6.3.3.4 Authorities required for the CRU ......................................................... 174  
6.3.4 The DCU ....................................................................................................... 175  
6.3.4.1 What is it? ............................................................................................. 175  
6.3.4.2 Defining the DCU to RACF ................................................................. 176  
6.3.5 The PLTPIU .................................................................................................... 177  
6.3.6 The CICS region data set protection .............................................................. 178  
6.3.6.1 Protecting the CICS region data sets .................................................... 178  
6.3.6.2 Authorising access to the CICS data sets ............................................. 180
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.3.7 Informing CICS terminal users about the forthcoming security change</td>
<td>182</td>
</tr>
<tr>
<td>6.3.8 Other parameters necessary for CICS security</td>
<td>184</td>
</tr>
<tr>
<td>6.4 Securing the resources for the CICS region A06C001</td>
<td>185</td>
</tr>
<tr>
<td>6.4.1 Decision about useful and necessary security mechanism</td>
<td>185</td>
</tr>
<tr>
<td>6.4.2 The CICS Terminal User Security</td>
<td>187</td>
</tr>
<tr>
<td>6.4.3 The Surrogate User Security</td>
<td>187</td>
</tr>
<tr>
<td>6.4.4 The CICS Transaction Security</td>
<td>189</td>
</tr>
<tr>
<td>6.4.4.1 The CICS Transaction Security Mechanism</td>
<td>189</td>
</tr>
<tr>
<td>6.4.4.2 Using security profiles to protect CICS transactions</td>
<td>190</td>
</tr>
<tr>
<td>6.4.4.3 Securing the IBM-supplied CAT1-transactions</td>
<td>191</td>
</tr>
<tr>
<td>6.4.4.4 Securing the IBM-supplied CAT2-transactions</td>
<td>193</td>
</tr>
<tr>
<td>6.4.4.5 Securing the IBM-supplied CAT3-transactions</td>
<td>194</td>
</tr>
<tr>
<td>6.4.4.6 Securing the MQSeries CICS transactions used for the NACT application</td>
<td>195</td>
</tr>
<tr>
<td>6.4.4.7 Securing the transactions NACT and CSKL</td>
<td>197</td>
</tr>
<tr>
<td>6.4.5 The CICS Command Security</td>
<td>197</td>
</tr>
<tr>
<td>6.4.5.1 The CICS Command Security Mechanism</td>
<td>197</td>
</tr>
<tr>
<td>6.4.5.2 Securing predefined CICS resources subject to CICS Command Security</td>
<td>198</td>
</tr>
<tr>
<td>6.5 Authorising access to the CICS region</td>
<td>206</td>
</tr>
<tr>
<td>6.6 Adjust the LOGIN terminal to pass capital letters to RACF</td>
<td>208</td>
</tr>
</tbody>
</table>

### 7 Summary and Further Work

- 7.1 Summary ............................................................................................................... 213
- 7.2 Further Work ......................................................................................................... 216

**Bibliography** .............................................................................................................. 217

**Appendices** ............................................................................................................... 221

**Appendix A** ............................................................................................................... 223
- A.1 Defining an SDS template using ISPF/PDF .......................................................... 223
- A.1 Listings referenced to in chapter 4 ...................................................................... 227

**Appendix B** ............................................................................................................... 229
- B.1 Reaching the log MSGUSR of the CICS region .................................................... 229
- B.2 Listings referenced to in chapter 5 ...................................................................... 233

**Appendix C** ............................................................................................................... 237
- C.1 Restarting the CICS region ................................................................................... 237
- C.2 Correction of a CICS system log failure after an OS/390-server IPL and a CICS restart | 238
- C.3 Listings referenced to in chapter 6 ...................................................................... 243
LIST OF FIGURES

Figure 1: Pedigree of the operating systems belonging to OS/390 ......................................................... 24
Figure 2: OS/390 – Basic architecture – Virtual address spaces (regions) .................................................. 37
Figure 3: CICS – Basic architecture ........................................................................................................ 37
Figure 4: The initial architecture of the CICS business applications NACT and MQNACT .................. 40
Figure 5: FTP session – Uploading a file into a SDS ............................................................................. 44
Figure 6: DATA SET LIST UTILITY screen in ISPF/PDF ..................................................................... 44
Figure 7: DSSLIST-screen in ISPF/PDF .................................................................................................. 45
Figure 8: Receiving the files from the data set “TBUSSE.CICSADP.NEWSEQ” ................................. 45
Figure 9: Receiving the files into the new data set “TBUSSE.CICSADP.LOADLIB” .......................... 46
Figure 10: List all received data sets (in blue colour) ............................................................................ 46
Figure 11: Deleting the SDS template “TBUSSE.CICSADP.NEWSEQ” .............................................. 47
Figure 12: Confirming the deletion of the SDS template ...................................................................... 47
Figure 13: Confirmation that the SDS template was deleted ................................................................ 48
Figure 14: Message also appeared in the DSSLIST utility ................................................................. 48
Figure 15: Summary of the components of the bank customer account application NACT ............... 50
Figure 16: ACCOUNTS MENU screen .............................................................................................. 51
Figure 17: ACCOUNTS DETAILS screen (example) ............................................................................ 51
Figure 18: ERROR REPORT screen (example) ................................................................................. 52
Figure 19: Scratchpad facility vers. Queuing facility ........................................................................... 59
Figure 20: The Scratchpad facilities CWA, TWA, and COMMAREA in comparison ......................... 62
Figure 21: CEDA DISP LI(DEMOLIST) ............................................................................................... 79
Figure 22: The two-way communication model for the MQNACT application (image taken from [AMQ95], ch. 2.0) .................................................. 86
Figure 23: Bidirectional communication ([ICM00], ch. 1.1.1.1.3) ....................................................... 90
Figure 24: The architecture of the MQSeries CICS application MQNACT ......................................... 94
Figure 25: The architecture of the MQSeries CICS application MQNACT with named MQSeries objects ................................................................................................................................. 95
Figure 26: MQSeries for OS/390 on CICS – Main Menu panel ........................................................... 98
Figure 27: MQSeries for OS/390 on CICS – Start a System Function panel ......................................... 99
Figure 28: MQSeries for OS/390 on CICS – Stop a System Function panel ...................................... 99
Figure 29: MQSeries for OS/390 on CICS – Display Connection panel ............................................ 103
Figure 30: MQSeries for OS/390 on CICS – Display Connection panel ............................................ 104
Figure 31: After starting the CKBR transaction the monitor is locked, system is in wait status .............. 107
Figure 32: Shutting down the MQSeries CICS Bridge manually – 01 ................................................. 108
Figure 33: Shutting down the MQSeries CICS Bridge manually – 02 ................................................. 108
Figure 34: Defining the transmission queue TBUSSE.NACT – 01 ...................................................... 114
Figure 35: Defining the transmission queue TBUSSE.NACT – 02 ...................................................... 114
Figure 36: Defining the transmission queue TBUSSE.NACT – 03 ...................................................... 115
Figure 37: Defining the transmission queue TBUSSE.NACT – 04 ...................................................... 115
Figure 38: Defining the remote queue definition TBUSSE.NACT.REPLYQ – 01 ............................... 116
Figure 39: Defining the remote queue definition TBUSSE.NACT.REPLYQ – 02 ............................... 117
Figure 40: Defining the channel sender for the transmission queue – 01 ........................................... 119
Figure 41: Defining the channel sender for the transmission queue – 02 ........................................... 119
Figure 42: Defining the channel receiver for the CICS Bridge queue – 01 ......................................... 120
Figure 43: Defining the channel receiver for the CICS Bridge queue – 02 ......................................... 121
Figure 44: Defining the queue manager TBUSSE.NACT – 01 ............................................................ 123
Figure 45: Defining the queue manager TBUSSE.NACT – 02 ............................................................ 123
Figure 46: The MQSeries Services, the green arrow indicates that the queue manager is up .......... 124
Figure 47: System objects for the queue manager TBUSSE.NACT ................................................... 124
Figure 48: NACT objects for the queue manager TBUSSE.NACT .................................................... 125
Figure 49: All defined queues for TBUSSE.NACT ................................................................................. 130
Figure 50: All created channels for TBUSSE.NACT ............................................................................. 131
Figure 51: Input screen of the MQSeries JAVA-application ............................................................... 137
Figure 52: Displaying the active users on OS/390 ............................................................................. 142
Figure 53: Displaying the started MQSeries Services – Channel initiator and TCP/IP listener – 01 .... 143
Figure 54: Displaying the started MQSeries Services – Channel initiator and TCP/IP listener – 02 .... 143
Figure 55: Starting the channel sender TBUSSE.NACT.WIN.OS ....................................................... 144
Figure 56: Message that the request to start the channel was accepted ........................................... 145
Figure 57: Channel was successfully connected to the server queue manager .................................. 145
Figure 58: Connecting the channel sender with the client queue manager ......................................... 147
Figure 59: Channel status after pressed the refresh key ...................................................................... 147
Figure 60: Channel receiver was activated after successful call from the server queue manager ....... 148
Figure 61: The Royal Bank of KanDoIT – Account Enquiry Client ....................................................... 149
Figure 62: The output messages on the DOS-console .......................................................................... 150
Figure 63: List Channels panel – Stopping a channel sender of the OS/390-server queue manager .... 151
Figure 64: List Channels panel – Channel sender is stopped .............................................................. 152
Figure 65: Stopping the channel sender on the WINDOWS2000 client queue manager ................. 152
Figure 66: Stopping the channel sender on the WINDOWS2000 client queue manager ................. 153
Figure 67: Stopping the channel sender on the WINDOWS2000 client queue manager ................. 153
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>68</td>
<td>Stopping the channel sender on the WINDOWS2000 client queue manager</td>
<td>154</td>
</tr>
<tr>
<td>69</td>
<td>List Channels panel – Ghost connection</td>
<td>155</td>
</tr>
<tr>
<td>70</td>
<td>List Channels panel – List Channels panel – Perform a function</td>
<td>157</td>
</tr>
<tr>
<td>71</td>
<td>List Channels panel – Perform a Channel Function panel – Reset message</td>
<td>157</td>
</tr>
<tr>
<td>72</td>
<td>MQSeries for WINDOWS2000 – Reset the message sequence number</td>
<td>158</td>
</tr>
<tr>
<td>73</td>
<td>System Authorisation Facility (SAF)</td>
<td>160</td>
</tr>
<tr>
<td>74</td>
<td>RLIST STARTED CICS*.** NORACF STDATA</td>
<td>174</td>
</tr>
<tr>
<td>75</td>
<td>The CICS SignOn panel</td>
<td>175</td>
</tr>
<tr>
<td>76</td>
<td>LISTUSER C001DEF NORACF CICS</td>
<td>177</td>
</tr>
<tr>
<td>77</td>
<td>Terminal screen after log on to CICS (without security)</td>
<td>183</td>
</tr>
<tr>
<td>78</td>
<td>Search for the transaction CESN – CEDA TRANS(CESN) DISP GR(*)</td>
<td>209</td>
</tr>
<tr>
<td>79</td>
<td>Display the transaction CEDA using DISP TRANS(CESN) GR(DFHSIGN)</td>
<td>209</td>
</tr>
<tr>
<td>80</td>
<td>Copy the transaction definition to the new group TYPENEU</td>
<td>210</td>
</tr>
<tr>
<td>81</td>
<td>Copy the profile definition to the new group TYPENEU as AAACICST</td>
<td>210</td>
</tr>
<tr>
<td>82</td>
<td>Display the contents of the group TYPENEU</td>
<td>211</td>
</tr>
<tr>
<td>83</td>
<td>Modify UCTRAN from NO to YES</td>
<td>211</td>
</tr>
<tr>
<td>84</td>
<td>Modify the profile name to the new one</td>
<td>212</td>
</tr>
<tr>
<td>85</td>
<td>CUSTOMPAC MASTER APPLICATION MENU on the JEDI OS/390-server</td>
<td>226</td>
</tr>
<tr>
<td>86</td>
<td>Data Set Utility screen in ISPF/PDF</td>
<td>226</td>
</tr>
<tr>
<td>87</td>
<td>“Allocate New Data Set” screen in ISPF/PDF</td>
<td>227</td>
</tr>
<tr>
<td>88</td>
<td>FTP session – Listing the own files on OS/390</td>
<td>227</td>
</tr>
<tr>
<td>89</td>
<td>FTP-session – Copying one source file to OS/390</td>
<td>228</td>
</tr>
<tr>
<td>90</td>
<td>Open the DA-Panel within SDSF</td>
<td>233</td>
</tr>
<tr>
<td>91</td>
<td>Placing a question mark in the column NP to display the logs</td>
<td>234</td>
</tr>
<tr>
<td>92</td>
<td>Placing the character &quot;S&quot; in the column NP to display the MSGUSR log</td>
<td>234</td>
</tr>
<tr>
<td>93</td>
<td>Searching for the MQSeries CICS Bridge message</td>
<td>235</td>
</tr>
<tr>
<td>94</td>
<td>The MQSeries CICS Bridge has been successfully started</td>
<td>235</td>
</tr>
</tbody>
</table>
# LIST OF TABLES

Table 1: Contents of the CD-ROM ................................................................. xxii
Table 2: The base elements and the optional features of OS/390 V2R7 ................ 31
Table 3: Files transferred from the accompanied CD-ROM onto OS/390 ............ 42
Table 4: CICS COBOL commands used in the application programs ................ 56
Table 5: Storage hierarchy in S/390 computers ............................................. 58
Table 6: Allowed operations on the CICS file objects ..................................... 75
Table 7: Named queue manager objects used for the MQSeries CICS application MQNACT .......... 92
Table 8: RACF commands ........................................................................ 164
Table 9: RACF commands ........................................................................ 166
Table 10: Authorisation Levels .................................................................... 166
Table 11: SIT-parameters set resp. modified for the CICS region A06C001 .......... 170
Table 12: Access permission to the CICS data sets ....................................... 181
Table 13: Profiles containing CAT2-transactions specified within the CLIST CAT2JEDI .......... 194
Table 14: CAT3-transactions secured by the CLIST CAT3JEDI ....................... 195
Table 15: MQSeries-supplied CICS transactions secured by the CLIST MQSJEDI .......... 196
Table 16: Access required for SP-type commands .......................................... 198
Table 17: The CICS resources accessible by the SP-type commands ............... 205
LIST OF LISTINGS

Listing 1: Extract from the COBOL copybook NACWCRUD (cf. Listing 17, page 228) ............... 63
Listing 2: Extract from the CICS COBOL program NACT01 (cf. Listing 13, page 227) ............. 63
Listing 3: Extract from the CICS COBOL program NACT02 (cf. Listing 14, page 227) .......... 65
Listing 4: Extract from the COBOL copybook NACCCRUD (cf. Listing 19, page 228) .......... 66
Listing 5: Extract from the member VSAM stored in TBUSSE.CICSADP.JCLLIB (part 1) ....... 70
Listing 6: Extract from the member VSAM stored in TBUSSE.CICSADP.JCLLIB (part 2) ..... 72
Listing 7: Extract from the member VSAM stored in TBUSSE.CICSADP.JCLLIB (part 3) ..... 73
Listing 8: Extract from the data member SYS1.COMMON(CICSC001) ........................................... 169
Listing 9: Extract from the CLIST CAT1JEDI stored in the data set CICS.COMMON.RACF ...... 192
Listing 10: Extract from the CLIST COM1JEDI stored in the data set CICS.COMMON.RACF ... 199
Listing 11: The physical map of the map set NACTSET ................................................................. 227
Listing 12: The symbolic description map of the map set NACTSET .............................................. 227
Listing 13: The 3270 presentation logic of the NACT application – NACT01 .......................... 227
Listing 14: The CRUD business logic of the NACT application – NACT02 ........................... 227
Listing 15: The Browse business logic of the NACT application – NACT05 .......................... 227
Listing 16: The Error Handling business logic of the NACT application – NACT04 ............. 227
Listing 17: The COBOL copybook NACWCRUD ........................................................................... 228
Listing 18: The COBOL copybook NACCTREC ............................................................................. 228
Listing 19: The COBOL copybook NACCCRUD ............................................................................ 228
Listing 20: Storing the data of the NACT application – Installing the account, locking, and name files on the OS/390-server ............................................................................................................... 228
Listing 21: The CICS resource definitions for the NACT application ........................................ 228
Listing 22: The additional CICS SIT definition script C001 (besides COMMON and END) ...... 228
Listing 23: The CICS startup script CICSC001 ................................................................................. 229
Listing 24: The script IEFSSN00 for the OS/390 subsystem name table ...................................... 233
Listing 25: The start script MQA1MSTR for the queue manager MQA1 ....................................... 233
Listing 26: Non-recoverable objects for the queue manager MQA1 defined within the script CSQ4INP1 ........................................................................................................................................... 233
Listing 27: Must have system objects for the queue manager MQA1 defined within the Script CSQ4INSNG ........................................................................................................................................... 233
Listing 28: System objects for distributed queuing and clustering (not CICS) for the queue manager MQA1 defined within the script CSQ4INSX ........................................................................................................ 233
Listing 29: The script CSQ4STRT starts the Channel Initiator and the Channel Listener  ................. 233
Listing 30: CICS objects for the MQSeries CICS adapter defined within the script CSQ4B100 ......... 234
Listing 31: The script DFHCSD01 to update the CSD ................................................................. 234
Listing 32: The script CSQ4INYG defines additional general objects for the queue manager MQA1
......................................................................................................................................................... 234
Listing 33: The updated CICS SIT definition script C001 (besides COMMON and END) .............. 234
Listing 34: The script CSQ4CKBM defines the MQSeries CICS Bridge queue and its trigger process
......................................................................................................................................................... 234
Listing 35: CICS objects for the MQSeries CICS Bridge defined within the script CSQ4CKBC ... 234
Listing 36: The MQSeries CICS Bridge – The script STRTCKBR to start the bridge automatically during
CICS start up ...................................................................................................................................... 235
Listing 37: The MQSeries CICS Bridge – The CICS COBOL compiling script DFHYIITVL ........... 235
Listing 38: The MQSeries CICS Bridge – The PLT for programs loaded during CICS startup ...... 235
Listing 39: The MQSeries CICS Bridge – The DFHAUPLE script that compiles PLT scripts ...... 235
Listing 40: The MQSC file “mqadmvs.tst” defines the objects for the WINDOWS2000-client queue manager
TBUSSE.NACT ................................................................................................................................. 235
Listing 41: The batch file loads the MQSC script “mqadmvs.txt” .................................................... 235
Listing 42: The MQSeries communication logic of the JAVA application – MQCommunicator.java
.............................................................................................................................................................. 236
Listing 43: The presentation logic of the JAVA application – MQClient.java .................................. 236
Listing 44: Error messages explain that the primary CICS system log cannot be accessed ......... 239
Listing 45: The script LISTLOG displays information about the log stream and its definition .... 239
Listing 46: The output of the LIST LOGSTREAM request .............................................................. 240
Listing 47: The script DELCLOG deletes the log stream entry from the LOGR policy ............. 241
Listing 48: The script DELCLOG deletes the log stream entry from the LOGR policy ............. 241
Listing 49: The default CICS region's SIT script COMMON ........................................................... 243
Listing 50: The CICS region's 2nd SIT script C001 – temporary version ......................................... 243
Listing 51: The CICS region's 2nd SIT script C001 – final version .................................................. 243
Listing 52: The CLIST CAT1JEDI secures Category-1 transactions ............................................... 243
Listing 53: The CLIST CAT2JEDI secures Category-2 transactions ............................................... 243
Listing 54: The CLIST CAT3JEDI secures Category-3 transactions ............................................... 244
Listing 55: The CLIST MQSJEDI secures MQSeries CICS transactions ....................................... 244
Listing 56: The CLIST USERJEDI secures CICS User-transactions ............................................... 244
Listing 57: The CLIST COM1JEDI secures CICS Resources subject to SP-type commands ....... 244
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Paraphrase</th>
<th>Chapter</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACID</td>
<td>Atomicity, Consistency, Isolation, Durability</td>
<td>2</td>
</tr>
<tr>
<td>AIX</td>
<td>alternate index</td>
<td>3.4.1</td>
</tr>
<tr>
<td>AMI</td>
<td>Application Messaging Interface</td>
<td>4.2.5</td>
</tr>
<tr>
<td>AMS</td>
<td>Access Method Services</td>
<td>3.4.1</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
<td>4.2.5</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interfaces</td>
<td>4.2.1</td>
</tr>
<tr>
<td>CICS</td>
<td>Customer Information Control System</td>
<td>2</td>
</tr>
<tr>
<td>CMAM</td>
<td>CUSTOMPAC MASTER APPLICATION MENU</td>
<td>3.2</td>
</tr>
<tr>
<td>COMMAREA</td>
<td>communication area</td>
<td>3.3.4</td>
</tr>
<tr>
<td>CRS</td>
<td>Cross-Region Sharing</td>
<td>3.4.2</td>
</tr>
<tr>
<td>CRUD</td>
<td>Create, Read, Update, Delete</td>
<td>3.3.3</td>
</tr>
<tr>
<td>CSD</td>
<td>CICS System Definition</td>
<td>3.5.1</td>
</tr>
<tr>
<td>CSS</td>
<td>Cross-System Sharing</td>
<td>3.4.2</td>
</tr>
<tr>
<td>CWA</td>
<td>Common Work Area</td>
<td>3.3.5</td>
</tr>
<tr>
<td>DASD</td>
<td>Direct Access Storage Devices</td>
<td>3.3.5</td>
</tr>
<tr>
<td>DCT</td>
<td>Destination Control Table</td>
<td>3.3.5</td>
</tr>
<tr>
<td>DSA</td>
<td>Dynamic Storage Area</td>
<td>3.3.5</td>
</tr>
<tr>
<td>DSLU</td>
<td>DATA SET LIST UTILITY</td>
<td>3.1</td>
</tr>
<tr>
<td>DSU</td>
<td>DATA SET UTILITY</td>
<td>3.1</td>
</tr>
<tr>
<td>EIBAID</td>
<td>EXEC Interface Block Attention Identifier</td>
<td>3.3.2</td>
</tr>
<tr>
<td>ESDS</td>
<td>Entry-Sequenced Data Sets</td>
<td>3.4.1</td>
</tr>
<tr>
<td>ETQ</td>
<td>Extrapartition TDQ</td>
<td>3.3.5</td>
</tr>
<tr>
<td>FCT</td>
<td>File Control Table</td>
<td>3.3.1</td>
</tr>
<tr>
<td>FIFO</td>
<td>First in, First out</td>
<td>4.2.3</td>
</tr>
<tr>
<td>HTTP</td>
<td>HyperText Transfer Protocols</td>
<td>4.2.1</td>
</tr>
<tr>
<td>IMS</td>
<td>Information Management System</td>
<td>3.4.1</td>
</tr>
<tr>
<td>ISPF/PDF</td>
<td>Interactive System Productivity Facility/Program Development Facility</td>
<td>3.1</td>
</tr>
<tr>
<td>ITDQ</td>
<td>Intrapartition TDQ</td>
<td>3.3.5</td>
</tr>
<tr>
<td>JCL</td>
<td>Job Control Language</td>
<td>3.3.2</td>
</tr>
<tr>
<td>JMS</td>
<td>JAVA Message Service</td>
<td>4.2.5</td>
</tr>
<tr>
<td>KSDDS</td>
<td>Key-Sequenced Data Set</td>
<td>3.4.1</td>
</tr>
<tr>
<td>LSR</td>
<td>local shared pool</td>
<td>3.5.1</td>
</tr>
<tr>
<td>MCA</td>
<td>Message Channel Agent</td>
<td>4.2.6.2</td>
</tr>
<tr>
<td>MMC</td>
<td>Microsoft Management Console</td>
<td>4.5.1</td>
</tr>
<tr>
<td>MQ</td>
<td>Message Queuing</td>
<td>4.2.1</td>
</tr>
<tr>
<td>MQI</td>
<td>Message Queue Interface</td>
<td>4.1</td>
</tr>
<tr>
<td>MQSC</td>
<td>MQSeries commands</td>
<td>4.4.2.2</td>
</tr>
<tr>
<td>MVX</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>OS/390</td>
<td>Operating System for a 390 processor</td>
<td>1</td>
</tr>
<tr>
<td>PDS</td>
<td>partitioned data set</td>
<td>3.1</td>
</tr>
<tr>
<td>PLT</td>
<td>Program List Table</td>
<td>4.4.2.4</td>
</tr>
<tr>
<td>QM</td>
<td>Queue manager</td>
<td>4.2.2</td>
</tr>
<tr>
<td>QMC</td>
<td>QM Clusters</td>
<td>4.2.4</td>
</tr>
<tr>
<td>RLS</td>
<td>Record Level Sharing</td>
<td>3.4.2</td>
</tr>
<tr>
<td>RPC</td>
<td>remote procedure calls</td>
<td>4.2.1</td>
</tr>
<tr>
<td>RRDS</td>
<td>Relative-Record Data Sets</td>
<td>3.4.1</td>
</tr>
<tr>
<td>SAM</td>
<td>Sequential Access Method</td>
<td>3.3.5</td>
</tr>
<tr>
<td>SDS</td>
<td>sequential data set</td>
<td>3.1</td>
</tr>
<tr>
<td>SDSF</td>
<td>System Display and Search Facility</td>
<td>3.4.2</td>
</tr>
<tr>
<td>SIT</td>
<td>System Initialisation Table</td>
<td>3.5.2</td>
</tr>
<tr>
<td>TCP/IP</td>
<td>Transmission Control Protocol/Internet Protocol</td>
<td>4.1</td>
</tr>
<tr>
<td>TD</td>
<td>Transient Data</td>
<td>3.3.5</td>
</tr>
<tr>
<td>TDQ</td>
<td>Transient Data Queues</td>
<td>3.3.5</td>
</tr>
<tr>
<td>TRANSID</td>
<td>transaction identifier</td>
<td>3.3.2</td>
</tr>
<tr>
<td>TS</td>
<td>Temporary Storage</td>
<td>3.3.5</td>
</tr>
<tr>
<td>TSQ</td>
<td>Temporary Storage Queues</td>
<td>3.3.5</td>
</tr>
<tr>
<td>TUA</td>
<td>Terminal User Area</td>
<td>3.3.5</td>
</tr>
<tr>
<td>TWA</td>
<td>Transaction Work Area</td>
<td>3.3.5</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>UoW</td>
<td>Unit of Work</td>
<td>3.3.1</td>
</tr>
<tr>
<td>VSAM</td>
<td>Virtual Storage Access Method</td>
<td>3.4.1</td>
</tr>
<tr>
<td>PDF</td>
<td>Portable Document Format</td>
<td>Notices</td>
</tr>
</tbody>
</table>
NOTICES

Conventions in this Master Thesis

Throughout this master thesis, the following conventions are used:

**Bold**

Indicates important words or word phrases.

*Italics*

Used for WINDOWS2000 file names, for proper names of MQSeries for WINDOWS2000, for JAVA statements, and for menu items and their sub items.

*Italics bold*

Indicates JAVA class names.

**UPPERCASE ITALICS**

Indicates data set, file, program, and script names used on the OS/390-server. Furthermore, this style is used for proper names of the CICS RACF security management (User IDs, Group IDs, security profile names, security classes, and so on).

“*Italics*”

Italics surrounded by non-italics quotes indicate citations.

UPPERCASE

Used for CICS object names, MQSeries object names, abbreviations, proper names in general, and some other special names.

**Lucida Console**

This format type indicates commands outside the continuous text. Also used in the listings and in the clip-pings of them.

*Lucida Console Italics*

Using this format style indicate commands and their parameters used in the continuous text.
Generation of a Java front end for a standalone CICS application accessed through MQSeries and securing CICS with RACF

[...]

These brackets enclose the shortcuts for sources listed in on page.

About the CD-ROM

The CD-ROM accompanying this master thesis contains the source and compiled code of the developed applications (COBOL and JAVA code) and the scripts that are required to run these applications. The source code and scripts are stored into folders of the CD-ROM in reference to the chapters where they are named (Table 1). Additionally, all the code referring to OS/390 is stored on the JEDI OS/390-server. The code resp. scripts applying to OS/390 data sets are stored on the CD-ROM into subdirectories of the listed folders in Table 1. Each subdirectory represents a high qualifier of an OS/390 data set. For example, the script C001 is stored to following directory on the CD-ROM: listings\chapter3\os390\scripts\cics\common\sysin\c001. This script is also stored to the JEDI OS/390-server into the data set CICS.COMMON.SYSIN as a member called C001. Each script is linked from the chapter to an entry in its associated appendix. This entry links directly to the scripts on the CD-ROM.

Besides the source code, the CD-ROM contains the referenced textbooks, if available as an electronic book. They are accompanied in IBM’s own format readable by the IBM Softcopy Reader. Some of them are also available as Portable Document Format (PDF) files. The documents can be directly downloaded from the CD-ROM’s directory books or accessed through a web browser when the file index.html is opened. Within this file there can also be linked to downloaded electronic documents referenced to in the text, that have been stored within the CD-ROM because of future unavailability.

<table>
<thead>
<tr>
<th>Code / scripts named in</th>
<th>Code / scripts related to</th>
<th>Code / scripts stored to folder:</th>
</tr>
</thead>
<tbody>
<tr>
<td>chapter 4 “The CICS Business Application NACT”</td>
<td>COBOL code Other OS/390 scripts</td>
<td>listings\chapter3\os390\cobol listings\chapter3\os390\scripts</td>
</tr>
<tr>
<td>chapter 5 “The MQSeries CICS Business Application MQNACT”</td>
<td>OS/390 scripts JAVA code MQSeries batch files</td>
<td>listings\chapter4\os390\scripts listings\chapter4\windows\java listings\chapter4\windows\mqseries</td>
</tr>
<tr>
<td>chapter 6 “Securing CICS with RACF”</td>
<td>OS/390 scripts</td>
<td>listings\chapter5\os390\scripts</td>
</tr>
</tbody>
</table>

Table 1: Contents of the CD-ROM

1 JEDI OS/390-server is one of the OS/390-servers of the university of Leipzig, Germany. For an apply of a login refer to the homepage jedi.informatik.uni-leipzig.de.
1 INTRODUCTION TO OS/390

1.1 Introduction

For the computers of the IBM System/390 architecture exist a variety of operating systems, which are implemented in dependence of the size of the installation. That are Transaction Processing Facility (TPF), Multiple Virtual Storage/Enterprise Systems Architecture, System Product Version 4 and Version 5 (MVS/ESA SP Version 4 & 5), Virtual Machine/Enterprise Systems Architecture (VM/ESA), Virtual Storage Extended/Enterprise Systems Architecture (VSE/ESA), and OS/390. All these operating systems have been developed by IBM. Further, there exist some other operating systems which also run on S/390 computers, for example Amdahl's operating system Universal Time Sharing Release 4 (UTS R4), and Hitachi's operating system Hi-OSF/1-MJ a UNIX variation. All the mentioned operating systems create “a different application execution environment with its own set of advantages and disadvantages.” ([HAF01]) Primarily, these environments vary in the interfaces between the applications and the operating systems. Further, they use central storage and create its virtual storage structure differently. Using both storage areas were an important reason to develop periodical enhancements of the operating systems for the S/3xx hardware.

The components of this master thesis are implemented on an S/390 computer system running with the operating system OS/390 Version 2 Release 7 (V2R7).

---

2 “OSF/1 is a variant of the Unix operating system that was largely a product of the so-called ‘Unix wars’ of the mid- and late 1980s. OSF/1’s roots lie in being one of the first operating systems to use the Mach kernel developed at Carnegie Mellon University.” from http://encyclopedia.thefreedictionary.com
1.2 History of OS/390

All the operating systems belonging to OS/390 have been evolved through several generations of large-computer systems (Figure 1). The evolution starts with the OS/360 family developed in 1964 on behalf of the introduction of the S/360 computers. It consists of the operating systems Primary Control Program (OS/PCP) as the simplest one, Multiprogramming with a Fixed number of Tasks (OS/MFT), and Multiprogramming with Variable number of Tasks (OS/MVT). OS/PCP had a sequential scheduler and could only handle 1 task at a time using maximum 12 KB central storage, whereas OS/MFT allowed up to 15 tasks to be processed at a time with a pre set memory

<table>
<thead>
<tr>
<th>Year</th>
<th>Operating System Family</th>
<th>Operating Systems belonging to OS/390</th>
</tr>
</thead>
<tbody>
<tr>
<td>1964</td>
<td>OS/360</td>
<td>OS/PCP, OS/MFT, OS/MVT</td>
</tr>
<tr>
<td>1972</td>
<td>OS/VS</td>
<td>OS/VS2 SVS</td>
</tr>
<tr>
<td>1974</td>
<td>OS/VS</td>
<td>OS/VS2 MVS R2</td>
</tr>
<tr>
<td>1975</td>
<td>OS/VS</td>
<td>OS/VS2 MVS R3</td>
</tr>
<tr>
<td>1979</td>
<td>OS/VS</td>
<td>OS/VS2 MVS R3.8</td>
</tr>
<tr>
<td>1981</td>
<td>MVS/370</td>
<td>MVS/370</td>
</tr>
<tr>
<td>1983</td>
<td>MVS/XA</td>
<td>MVS/XA, MVS/SP Version 1, DFP/370</td>
</tr>
<tr>
<td>1985</td>
<td>MVS/XA</td>
<td>MVS/XA, MVS/SP Version 2, MVS/XA DFP Version 1, MVS/XA DFP Version 2</td>
</tr>
<tr>
<td>1990</td>
<td>MVS/ESA</td>
<td>MVS/ESA SP Version 4</td>
</tr>
<tr>
<td>1994</td>
<td>MVS/ESA</td>
<td>MVS/ESA SP Version 5</td>
</tr>
<tr>
<td>1995</td>
<td>OS/390</td>
<td>OS/390 Version 1 R1</td>
</tr>
<tr>
<td>1996</td>
<td>OS/390</td>
<td>OS/390 Version 1 R3</td>
</tr>
<tr>
<td>1997</td>
<td>OS/390</td>
<td>OS/390 Version 2 R4</td>
</tr>
<tr>
<td>1999</td>
<td>OS/390</td>
<td>OS/390 Version 2 R7</td>
</tr>
<tr>
<td>2000</td>
<td>OS/390</td>
<td>OS/390 Version 2 R10</td>
</tr>
<tr>
<td>2000</td>
<td>z/OS</td>
<td>z/OS Version 1 R1</td>
</tr>
<tr>
<td>2000</td>
<td>z/OS</td>
<td>z/OS Version 1 R2</td>
</tr>
<tr>
<td>2004</td>
<td>z/OS</td>
<td>z/OS Version 1 R3</td>
</tr>
</tbody>
</table>

Figure 1: Pedigree of the operating systems belonging to OS/390
allocation of central storage (up to 64 KB). The same amount of tasks could be processed by OS/MVT, but these tasks could share whatever central storage was installed on the computer system (maximum 128 KB).

Eight years later, IBM announced System/370 hardware, that provided virtual storage capability. At the same time, the next OS family OS/VirtualStorage (OS/VS) has been introduced to utilise the new hardware. OS/360 has to be transformed into a virtual storage system. Because many customers preferred OS/MFT and others OS/MVT both should be satisfied. OS/MFT was improved to become OS/VS1 and OS/MVT became OS/VS2, taking account of virtual storage in only a single partition up to 16MB (2^24 bits = 16 MB). Therefore, OS/VS2 has also be named as OS/VS2 Single Virtual Storage (SVS).

In 1974 the next offered release supported multiple virtual storage for each address spaces up to 16 MB and was named OS/VS2 Multiple Virtual Storage (MVS) Release 2 (R2). The third release of OS/VS2 MVS (1975) provided so-called Selectable Units (SUs) to allow the customer to tailor their operating system only to functions they needed. In 1979, the last update OS/VS2 MVS R3.8 was published. Exceptionally, it was required as an installation base for the next developed operating system MVS/370 introduced with the new S/370 hardware. What has been named as MVS/370 were in real a combination of two products – MVS/Service Product Version 1 (MVS/SP V1) and Data Facility Product/370 (DFP/370). This OS supported the 26 address bits of the central storage provided by the processor architecture, hence, the MVS/370 could address up to 64 MB of this memory. Although, the operating system had could utilise these 26 bits (2^26 bits = 64 MB) instead of the 24 bits for its virtual storage, IBM did not wanted it because many existing MVS/370 applications would not properly work. Therefore, only up to 16 MBs for virtual storage could be addressed per address space by MVS/370.

This problem has been resolved by a new hardware – the S/370 eXtended Architecture (370-XA), firstly shipped in 1983. For the first time, the computer system had have 31 bits (2^31 bits = 2 GB) of central storage and a dynamic channel capability for I/O. The 370-XA has been utilised by a new version of MVS/370, called MVS/eXtended Architecture (MVS/XA). This OS package also consisted of two products that made up the operating system – MVS/SP V2 and MVS/XA DFP V1. MVS/SP V2 supported both storage areas, whereas MVS/XA DFP V1 contained five updated data-management function of OS/VS2 MVS R3.8 besides the functions of DFP/370. Hence, OS/VS2 MVS R3.8 was not more required as a base for the new operating system. Dynamic channel architecture could access up to eight I/O channels, although the hardware had only four. This was the most visible feature of the new OS and I/O boosted dramatically. Furthermore, MVS/XA supported the whole 31 bits to build the virtual storage up to 2 GB per address space. However, central storage could only be addressed by MVS/XA in the range of maximum 256 MB (28 bits). Additionally, to provide compatibility with previous versions, MVS/XA still supported the 24 bit mode for addressing virtual storage areas. In 1985 the second version of MVS/XA DFP was published.

MVS/ESA was the successor of the MVS/XA operating system and has been introduced in 1988. This operating system run on a new hardware called Enterprise Systems Architecture/370 (ESA/370). It contained the third versions of MVS/SP and MVS/XA DFP. Two years later (1990), a new hardware has been introduced – the ESA/390 and ES9000 S/390 family. For both systems only a new version of one product of MVS/ESA has been
provided – MVS/ESA SP V4. The DFP version was not more needed. All its functions were integrated in the
new OS. In 1993, the fifth and last version of MVS/ESA SP was delivered. When in 1995 a new version was put
on the market, it was renamed to OS/390 Version 1 Release 1. Until October 2000 there were published every
half year a new release of OS/390 – the last was release 10. These operating systems for the S/390 architecture
are elaborated in the next chapter “OS/390 in general”.

However, OS/390 has also been replaced in 2000 by the z/Operating System (z/OS) family for the zSeries
computers\(^3\). These computer systems now provide the 64 bits addressing of the central storage, which the OS also
supports \((2^{64} \text{ bits} = 17.179.869.184 \text{ GB} = 16.777.216 \text{ Terabytes (TB)} = 16.384 \text{ Petabytes (PB)} = 16 \text{ Exabytes (EB)})\).

Please look for a brief history for the operating systems of S/3xx computers in [HAF01] and on the interesting
homepage of T. Falissard [FAL01].

\(^3\) Is it a myth? The “z” in z/OS and zSeries paraphrases the meaning of “zero down-time” of the zSeries computer systems, as IBM intern-
al propagates. It means, that the zSeries computers, on which z/OS runs, do not have any down-time. Understand it as a homage to their
stable computer systems. Of course, it is a marketing philosophy.
1.3 OS/390 in general

An operating system like OS/390 is the intermediate layer between the application software and the hardware; it works behind the scene. An OS consists of a set of programs to create and manage the environment in which other programs resp. applications can execute. Additionally, it allows to run several of those programs at the same time. Each OS (OS/390, too) can be categorised into following operating-system functions (according to [ES-1011]):

- Services (for application programs): contain functions provided by the OS that release the programs from managing them.
- Resource Management: distributes resources to the application programs as they are running. Such resources are the processor storage, processors, and I/O channels.
- Storage Management: divides the main storage proportionally and sufficiently to run the programs in the system.
- Timer Services: supports time-dependent functions in application program logic.
- Serialisation Services: coordinates the shared resources and maintain the integrity of updated resources.

In contrast to personal computer systems, large computer systems like S/390 can have multiple processors. Managing these processors in a so-called multi-processing system is another must for such an operating system. It must also handle interrupts generated by the S/390 hardware to allow pre-emptive multitasking (giving the programs execution priorities). As last but important feature, the OS must identify a hardware-detected error (machine or program check) to react appropriate.

OS/390 has been designed for enterprise computing to change rapidly highly competitive business environments. This operating system is described in [HAF01] as “a network-ready, integrated operational environment for S/390.” An important advantage of OS/390 is that it remains operational through system difficulties and it can be quickly recovered after a system disaster. According to [ES1011] OS/390 has the following characteristics:

- High-availability and high-performance
- Air-tight security
- Connectivity diversity
- Flexibility to handle ever growing complex integrated work flows
- Affordability in costs for the user.

Actually, OS/390 was a new name for the MVS/ESA SP Version 5 Release 2.2 (V5R2.2) (re-branding) introduced in 1995 (see Figure 1 on page 24, too). The name OS/390 belongs to the S/390 architecture. However, this system was introduced in 1990 with the Enterprise System Architecture/390 (ESA/390) and the ES/9000 S/390.
From this architecture the present name S/390 has been derived. The operating system running on the ESA/390 and S/390 systems was MVS/ESA until 1995. To honour the S/390 system MVS/ESA was renamed to OS/390.

In contrast to MVS/ESA, OS/390 has been delivered within a basic installation package containing a set of separate but required products needed for S/390 operation. Optional functions could be additionally added to the installation. All in all, OS/390 can be a bundling of up to 70 software components. To distinguish the operating system from the OS/390 package, the older name MVS was used by the system operators to identify the OS. Because that the base OS of the OS/390 package is actually not more than MVS/ESA, the main operating system features are described by means of MVS/ESA.

As mentioned above, MVS/ESA was the successor of the MVS/XA operating system. The evolution of operating systems is closely related with the storage management. As same as MVS/XA, MVS/ESA supports the 24-bit and 31-bit mode of operation. But in contrast to MVS/XA, which only supported 256 MB of central storage (contrary, architecture allowed up to 2 GB), MVS/ESA supported the whole $2^{31}$ bits = 2 GB of central storage. Virtual storage could be maximum of 2 GB per address space (often called region) created by both operating systems. However, MVS/ESA can build another virtual storage area called data spaces ranging in size from 4 KB to 2 GB. This storage area is byte addressable by an application program to store and access application relevant data, nothing more. The normal virtual address spaces were thenceforth called primary address spaces. Once again, applications can only run in a separate primary address spaces, not in data spaces. The data spaces are accessible by the applications through a facility called Access-Register Addressing. Of course, an application running in a primary address space can furthermore access another application running in another allocated primary address space to pass control or to access data. Primary address spaces are linked together using the Cross Memory Facility linkage as already used in earlier MVS versions.

MVS/ESA provides a better expanded storage utilisation than MVS/XA. This storage technique has been introduced in 1985 and was firstly supported by MVS/XA at the same time. On S/390 systems is maximum 8 GB expanded storage available. Access to expanded storage has only the central storage to transfer still needed information that is just not used or has a low priority. If this information is needed, it must be moved back to the central storage. This transfer is managed by the operating system using paging instead of bytes-addressing.

What the expanded storage is for the central storage, is another address space called hiperspace for the primary address space firstly supported by MVS/ESA. However, hiperspace normally resides in the expanded storage to use its storage method (paging). Each data stored under control of the application program into the hiperspace must be moved to the primary address space before the application program it can use. Today, OS/390 uses the same storage management products and methods as MVS/ESA resp. MVS/XA.

With the introduction of MVS/ESA SP V4 in 1990 were delivered some resource enhancements for new and old facilities. There has been added a support to connect a few processors to a single logical unit called a system complex (Sysplex). Synchronising the time-of-day clocks on each sysplex is supported by a new facility called Sysplex Timer. The new ESCON I/O channels have also been now supported. Another software function, called
Cross System Coupling Facility (XCF), has been added to the new operating system. "XCF provides the application programming interface and services to allow communication among application groups on the separate processors and provides a monitoring ("heartbeat") and signaling capabilities." ([HAF01]) XCF is used by the Job Entry Subsystem 2 (JES2), CICS eXtended Recovery Facility (XRF), among others. Advanced Program-to-Program Communications/MVS (APPC/MVS) has been included in later releases of MVS/ESA SP V4. There have been also added some new functions to manage the storage environment. MVS/ESA SP V4 supports as first the open systems environment with a POSIX-compliant API.

One of the main enhancements of MVS/ESA SP V5 introduced in 1994 was the support for Parallel Sysplex services for up to 32 MVS/ESA systems. This method is called Coupling Facility. A new software product called the WorkLoad Manager (WLM) has been introduced to define processing goals reflecting to specific business needs. The most important package expansion was the MVS Open Edition, today called UNIX System Services (USS). For a better batch handling there was included a product called BatchPipes/MVS.

MVS/ESA SP V5 was the last operating system version having the name MVS/ESA. OS/390 Version 1 Release 1 has been delivered in 1995. The base operating system, also called Base Control Program (BCP) was still named MVS/ESA V5 R2.2. OS/390 V1R1 includes some base and optional elements. These have been also delivered as further enhancements within the OS/390 V2R7 described in the next chapter. After OS/390 V1R3 there has been announced in 1997 as next operating system OS/390 Version 2. It started with the release 4 to continue the release numbers. Within OS/390 V2 it has been firstly included the Domino Go Webservice (now named WebSphere Application Server (WAS)). Furthermore, Tivoli’s TME 10 Framework has then integrated into OS/390. Distributed Computing is completed by the addition of the ENCINA Toolkit Executive, DCE Application Support providing RPC, and the Lightweight Directory Access Protocol (LDAP) client support. There have been added some enhancements to WLM and to the USS (UNIX Application Services as Shell, Utilities, and Debugger). USS now supports the UNIX 95 resp. XPG4.2 X/Open Company's Single UNIX Specification including all the commands and utilities referred to it. Within the next release 5, OS/390 included as main part an updated Domino Go Webservice. When a new hardware function support (here: FICON channel support) was available, all OS/390 releases down to release 3 have to be updated by a new software function. OS/390 V2R6 introduced in September 1998 included some important enhancements for the eNetwork Communications Server support and for the Parallel Sysplex function. Six months later, in March 1999, OS/390 V2R7 was available – the OS package that is installed on the JEDI OS/390-server.

Beyond OS/390 V2R7, there have been published 3 more releases of OS/390 before the new operating system z/OS has been announced in 2000 for the new zSeries computer systems. The features of the elements of these operating systems will not be reflected here, please refer to [HAF01]. Also, use this book for a comprehensive overview of the referenced products for the operating systems.
1.4 OS/390 V2R7

On the JEDI OS/390-server is installed OS/390 V2R7 which has been offered in March 1999. Besides the base operating system it contains a set of so-called OS services, distinguished into base elements always delivered and optional features delivered on demand. Furthermore, USS has been also packaged as a base OS service into OS/390. The important base Systems Services are the Base Control Program (BCP), JES2, Interactive System Productivity Facility (ISPF), and TSO/E. System Modification Program/Extended (SMP/E) and the optional System Display Search Facility (SDSF) are Systems Management Services, among others, that have been installed on the JEDI OS/390-server. Another important product that has been firstly included into OS/390 V2R5 is the WebSphere Application Server (WAS). The optional Security Services including the Security Server and the Resource Access Control Facility (RACF) have been also installed on OS/390-server. Some additional components contain the language compilers for COBOL, Object-Oriented COBOL, PL/I, C/C++, JAVA Development Kit (JDK) 1.1.8 with the SWING package and a JAVA Virtual Machine (JVM). Table 2 gives an overview of all base and optional elements of OS/390 V2R7.

On the JEDI OS/390-server are installed furthermore two application managers that support the high-performance transaction processing: Information Management System (IMS) and Customer Information Control System (CICS).

<table>
<thead>
<tr>
<th>OS Services</th>
<th>Base Elements</th>
<th>Optional Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Systems Services</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• BCP</td>
<td>• ISPF</td>
<td>• JES3</td>
</tr>
<tr>
<td>• Bulk Data Transfer (BDT)</td>
<td>• JES2</td>
<td>• Bulk Data Transfer (BDT) File-to-File</td>
</tr>
<tr>
<td>• DFSMSdfp</td>
<td>• MICR/OCR Support</td>
<td>• Bulk Data Transfer (BDT) SNA NJE</td>
</tr>
<tr>
<td>• EREP</td>
<td>• TSO/E</td>
<td></td>
</tr>
<tr>
<td>• ESCON Director Support</td>
<td>• 3270 PC File Transfer Program</td>
<td></td>
</tr>
<tr>
<td>• High Level Assembler (HLASM)</td>
<td>• FFST</td>
<td></td>
</tr>
<tr>
<td>• ICKDSF</td>
<td>• TIOC</td>
<td></td>
</tr>
<tr>
<td>Systems Management Services</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Cryptographic Services (includes ICSF) (new)</td>
<td>• DFSMSdss, DFSMSrmm, DFSMSHsm</td>
<td></td>
</tr>
<tr>
<td>• HCD</td>
<td>• HCM</td>
<td></td>
</tr>
<tr>
<td>• SMP/E</td>
<td>• Open Cryptographic Services Facility (OCSF France) (new)</td>
<td></td>
</tr>
<tr>
<td>• Tivoli Management Framework (new)</td>
<td>• OCSF Security Level 1, 2, 3 (new)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• RMF</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• SDSF</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• System Secure Sockets Layer (SSL) Crypto (new)</td>
<td></td>
</tr>
</tbody>
</table>

*Table 2: The base elements and the optional features of OS/390 V2R7 (continued on the next page)*

---

4 Speak “RAAK-EFF”.
5 Speak “KICKS” or “C–I–C–S”.
<table>
<thead>
<tr>
<th>OS Services</th>
<th>Base Elements</th>
<th>Optional Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Enablement</td>
<td>• DCE Application Support</td>
<td>• C/C++ with Debug Tool</td>
</tr>
<tr>
<td>Services</td>
<td>• Encina Toolkit Executive</td>
<td>• C/C++ without Debug Tool</td>
</tr>
<tr>
<td></td>
<td>• GDDM (includes PCLK and OS/2 Link)</td>
<td>• DFSORT</td>
</tr>
<tr>
<td></td>
<td>• Language Environment (LE)</td>
<td>• GDDM-PGF</td>
</tr>
<tr>
<td></td>
<td>• Application Enabling Technology</td>
<td>• GDDM-REXX</td>
</tr>
<tr>
<td></td>
<td>• SOMobjects Runtime Library</td>
<td>• High Level Assembler (HLASM) Toolkit</td>
</tr>
<tr>
<td></td>
<td>• VisualLift Runtime Library</td>
<td>• Language Environment Data Decryption</td>
</tr>
<tr>
<td></td>
<td>• C/C++ IBM Open Class Library</td>
<td>• SOMobjects Application Development Environment</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• VisualLift Application Development Environment for MVS, VSE, VM</td>
</tr>
<tr>
<td>Distributed Computing</td>
<td>• DCE Base Services (OSF DCE level 1.1)</td>
<td>• DCE User Data Privacy (DES and CDMF) - OSF DCE 1.1 level</td>
</tr>
<tr>
<td>Services</td>
<td>• Distributed File Service (OSF DCE level 1.2.2)</td>
<td>• DCE User Data Privacy (CDMF) - OSF DCE 1.1 level</td>
</tr>
<tr>
<td></td>
<td>• Network File System</td>
<td>• OS/390 Print Server</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• OS/390 Print Interface</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Windows 95/NT client</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• IP PrintWay</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• NetSpool</td>
</tr>
<tr>
<td>eNetwork Communications</td>
<td>• IP (formerly TCP/IP)</td>
<td>• eNetwork Communications Server Security Level 1, 2, 3 (new)</td>
</tr>
<tr>
<td>Server</td>
<td>• SNA (includes AnyNet) (formerly VTAM)</td>
<td>• eNetwork Communications Server Network Print Facility (NPF)</td>
</tr>
<tr>
<td>Network Computing Services</td>
<td>• WebSphere Application Server</td>
<td>• IBM HTTP Server Export Secure</td>
</tr>
<tr>
<td></td>
<td>• NetQuestion</td>
<td>• IBM HTTP Server France Secure</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• IBM HTTP Server NA Secure</td>
</tr>
<tr>
<td>Softcopy Services</td>
<td>• BookManager READ</td>
<td>• BookManager BUILD</td>
</tr>
<tr>
<td></td>
<td>• BookManager BookServer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Softcopy Print (includes Softcopy Print for DBCS Languages)</td>
<td></td>
</tr>
<tr>
<td>UNIX System Services</td>
<td>• OS/390 UNIX System Services Application Services</td>
<td></td>
</tr>
<tr>
<td>(X/Open UNIX 95</td>
<td>• Shell</td>
<td></td>
</tr>
<tr>
<td>functions)</td>
<td>• Utilities</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Debugger</td>
<td></td>
</tr>
<tr>
<td>LAN Services</td>
<td>• OS/390 UNIX System Services Kernel</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• LANRES</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• LAN Server</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• OSA Support Facility</td>
<td></td>
</tr>
<tr>
<td>Security Server</td>
<td></td>
<td>• Security Server</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• RACF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DCE Security Server at OSF DCE level 1.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• LDAP Server</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Firewall Technologies</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Security Server LDAP Server DES</td>
</tr>
</tbody>
</table>

Table 2: The base elements and the optional features of OS/390 V2R7
(according to [IRG99])
2 INTRODUCTION TO CICS

2.1 Introduction

Wherever today a day goes by, someone is involved in a transaction, somewhere in the world. It could be a private or business purpose – for example paying at the cash desk in a supermarket, taking money from a bank account using an automatic teller machine (ATM), delivering packages or checking and updating personal records in a business company, comparing customer contracts in an insurance company or taking a reservation in a car rental or airline company. Stored in a database, this information is retrieved when a business application program is started using a transaction that delivers the required data. Such transactions are business operations on behalf of a company. Today the market, whether it is traditional, E-business or mixed, lives or dies due to connected to On-Line Transaction Processing (OLTP) systems. CICS is such an OLTP product developed by IBM, that manages those business applications.

However, on-line should not be confused with the today meaning – using a terminal for doing all interactions on a server logically situated away from but connected with the terminal. Originally, on-line meant “that a file could be read or manipulated in some manner other than a batch job.” ([YOU01], page 43) In contrast, early on-line processes allowed only one user accessing one record at a time. Such a procedure has been usually called as a transaction. But, the more users wanted to access a record concurrently the more a system has been needed to manage such transaction traffic.
2.2 History of CICS

IBM introduced in 1968 for its mainframe computer system S/360 a transaction management system called Public Utility Customer Information Control System (PUCICS) to manage the increasing transaction processing. Surprisingly, this initial version of CICS became available as a free distribution. One year later, CICS was packaged as a “Class A Program Product” (Class A PPs provides software defect support), now no more free of charge. The first release of CICS for the OS/360 family supported only the ASSEMBLER programming language, however, in 1970, CICS supported also the programming languages COBOL and PL/I. In 1972, CICS supported the new technology of the IBM 3270 Display System, right away, including the Basic Mapping Support (BMS) as well as sending and receiving native 3270 data streams directly to application programs. CICS also supported from that year on the Virtual System (VS) environment introduced within the operating system VM/370 for S/370 computer systems. Within the next years “a number of new technologies were added to CICS, including support for online update and recovery/restart” [YEL01] as well as supporting the virtual storage and the Data Language/I (DL/I). Furthermore, CICS got its command level programming interface in 1977. “Command level program isolates the application from its physical environment and has enabled customers to implement distributed systems (client/server)” [YEL01] Until 1980 CICS supported additionally the Multiple Region Operation (MRO) and the InterSystem Communication (ISC). Both support the easy distribution and running of user applications and data resources across multiple computer systems having multiple processors. During the 1980s CICS was ported onto the personal computer having installed IBM's operating system OS/2, as same as ported onto UNIX machines. Further, CICS was improved to run on the XA architecture to support the 31 bit addressing; hence, the virtual storage CICS could address increased now above the so-called 16 megabit line. RACF became more important as an external security manager for CICS to secure the access to CICS resources. The introduction of Resource Definition Online (RDO), the support for Advanced Program-to-Program Communication (APPC) which involved the use of LU6.2 protocols, and the Extended Recovery Facility (XRF) to enable users to achieve higher levels of system and application availability were another important features developed for CICS during the early 1980s. With the introduction of parallel computer systems in the 1990s CICS was upgraded to support these computers. CICS supports since 1992 IBM's Message Oriented Middleware (MOM) product MQSeries. “By 1993, there were tens of thousands of CICS licenses for use of the product in MVS, ESA, AIX, AS/400 and OS/2 environments. CICS had become the most widely used transaction processing system.” [YEL02] In 1996, the first version of the CICS Transaction Server (TS) for OS/390, for Windows NT, and for AIX resp. other UNIX systems became available. This new product supported Assembler, COBOL, C, PL/I, JAVA plus support of APPC and the Transmission Control Protocol/Internet Protocol (TCP/IP). Within this year, CICS resp. CICS TS was prepared for the Internet era supporting some new CICS-accessing methods, for example the CICS Internet Gateway, the CICS Gateway for JAVA, the CICS Gateway for Lotus Notes, or the CICS Web Interface (CWI). The next release of the CICS TS version 1.2 has included a major enhancement –

\[YEL01\] For an interesting homepage for CICS references and CICS related documents see “Follow The Yelavich Road”: http://www.yelavich.com/
the CICS 3270 bridge. Hence, CICS programmers were able to enable legacy CICS 3270 applications to access them through the Internet without rewriting the original code.

In 1998, the CICS TS v 1.3 for OS/390 was announced and delivered within 1999. Following enhancements and improvements were realised resp. developed:

- Supporting C++,
- CICS Transaction Gateway (CTG) containing the functions of the CICS Internet Gateway and the CICS Gateway for Java,
- Open Transaction Environment (OTE) to execute CICS JAVA applications in an own CICS region, but under an OTE Task Control Block (TCB) and not the primary CICS Quasi-Reentrant (QR) TCB, a special MVS TCB,
- Business Transaction Services (BTS) allow users to define an overall business process and its related transactions,
- Supporting the Common Object Request Broker Architecture (CORBA), and
- CICS Web Support (CWS), the restructured code of the CWI, to support the CICS domain architecture.

On the JEDI OS/390-server is installed the CICS TS 1.3 because this version is the last available one for OS/390. Therefore, the master thesis bases on this OLTP system.

In the next years until 2004, three new versions of CICS TS for z/OS were introduced – CICS TS 2.1, 2.2, and the last version 2.3. The most important enhancement of the CICS TS 2.1 for z/OS was the support of Enterprise JavaBeans (EJB). These three CICS versions improve each time the performance in using JVMs on z/OS. For example, more than one CICS JAVA program can be used within a CICS task.

Besides CICS, there exist today a variety of other OLTP systems running on a huge number of operating systems – mainframe, open, and also on personal computer systems. However, a major part of transaction processing takes still place on mainframe systems. For example, Siemens provides for its mainframe operating system BS2000 an own TP monitor called Universal Transaction Monitor (UTM). Other known TP monitors for open systems as openUTM from Siemens, Tuxedo from BEA Systems, or Encina from Transarc are used as for example on IBM AIX, DCE DC/OSx, Hewlett-Packard HP-UX, or Novell Unixware resp. Netware.
2.3 CICS in general

However, CICS is one of the most important and most known TP monitors running on z/OS, OS/390, MVS/ESA, OS/400, OS/2, Windows NT and 2000, and on the UNIX Systems AIX (IBM), Solaris (SUN), and HP-UX (Hewlett-Packard). It is used for procedural development organisations and in large-scale applications that expect to have many users and high transactions rates. That is why, CICS is also called a cross industry product. Supported by more than 5000 software packages from more than 2000 vendors, CICS still defends its leading position regarding to performance, reliability, and availability. This TP-monitor is used in business critical applications which demand a high grade of these requirements. A total outage of such applications would terminate the whole business process. Nothing may be lost, nothing may be falsified, and an unauthorised person must not see or change any parts of the data. That means, that transactions must be routed securely, reliably, and efficiently through the whole network of all involved computers. It takes care of the security and integrity of the data while looking after resource scheduling. Transaction processing has been evolved to a heterogeneous application and hence, “CICS is an «application server».” [YEL01] It integrates all the basic software services required by OLTP applications. However, CICS has never been used this term.

CICS’ function to allow reliable, concurrent access and manipulation of data is still fundamentally. Each interaction between the user and the CICS TS is done by a client. Such a client/server scenario is commonly known as middleware. Middleware is described as a general purpose service between the heterogeneous computing platform (hardware and operating system) and its transaction-based applications and to support their distribution within the network. There are some different classifications of what middleware is: Transaction Processing (TP) monitors, DataBase Access (DBA) modules, Message-Oriented Middleware (MOM), Object Request Brokers (ORB), Remote Procedure Calls (RPC), security management products, and web servers, too.

Following description of CICS is given in chapter 1.1.1 of [APP90]:

“CICS (Customer Information Control System) is a general-purpose data communication system that can support a network of many hundreds of terminals. You may find it helpful to think of CICS as an operating system within your own operating system ... In these terms, CICS is a specialized operating system whose job is to provide an environment for the execution of your online application programs, including interfaces to files and database products.

The total system is known as a database/data-communication system ...” (DB/DC system). “Your host operating system, of course, is still the final interface with the computer; CICS is «merely» another interface, this time with the operating system itself.”

CICS includes following features: End-to-End Integrity, Listener, Security, Scheduling, Naming, Performance, Exception Handling, Locking, Connectivity, Administration, Logging, Time Control, Multithreading, Recovery, and Authorisation. The TP monitor runs as a batch processing job in a separate virtual address space for instance on an OS/390-server (Figure 2). Such a virtual address space is also called region. CICS application pro-
grams run in the same virtual address space as the CICS kernel. In contrast, the database process runs in another separate virtual address space of OS/390. The CICS kernel consists of the following components: Terminal Control, Program Control, Storage Control, File Control, and Interval Control. Each of these modules has an associated table in the main storage of OS/390 (Figure 3). Besides, running CICS as a single copy on an S/390 computer, there can also be installed multiple CICS copies on a single S/390 computer, and further, multiple copies on different S/390 computer systems, which communicate freely with each other. That is, share data!

Figure 2: OS/390 – Basic architecture – Virtual address spaces (regions)

Figure 3: CICS – Basic architecture
Some more interesting facts about CICS are assembled from [HKS04] and [ES1011]:

- Installed on 70,000 servers in 15,000 companies worldwide it is used by almost one million CICS application engineers. Compared to this, about 30 millions CICS terminals are connected to these CICS servers. In contrast, there exist 500 millions personal computers and 200 millions Internet connections. About 90% out of the 2000 biggest companies use CICS.
- More than 150000 users can use a CICS application server concurrently at a time.
- CICS can serve up to 30 billion transaction a day and over 300 billion dollars are transferred in transactions each week.
- CICS can handle thousands of transactions per second.
- More than 1 billion US-dollars have been invested in CICS based applications.
- CICS transactions can run programs written in a variety of programming languages, for example JAVA, C, C++ COBOL, PL/I, Assembler, REXX, etc.
- CICS commands are the same on all platforms, beginning with an EXEC CICS statement.
- “CICS changes, the user's applications need not.” [YEL03]
- CICS works on basis of the ACID criteria (ACID = Atomicity, Consistency, Isolation, and Durability of the data).

For a more comprehensive understanding about CICS, see chapter 7 of [HKS04] and chapter 1.1 of [APP90].
THE INITIAL ARCHITECTURE OF THE CICS BUSINESS APPLICATIONS

Within the next two chapters we create two CICS business applications – describing two possibilities to use CICS as an OLTP system and as an application server (see Figure 4 on next page). As the key components of each application, a business logic component shared by both applications and a presentation logic component for each application are created. They represent a bank program for debit and cash card holders to manage their customer accounts stored to a database-like file by the clerical staff. The customer account file is stored on an OS/390-server and is accessed by the CICS TS. The applications provide procedures to update the customer information, to create and delete customer accounts, and to read them.

Chapter 4 “The CICS Business Application NACT” on page 41 describes in detail the contents of the business and presentation logic component of the CICS application NACT. All programs of the application are written in COBOL. The application is invoked by an associated CICS transaction. Executed on a 3270 terminal screen running on a WINDOWS2000 computer system, it inquires a customer record from the OS/390 database-like file and displays the result on the screen. The services of this CICS application are provided by the legacy 3270 interface. This interface is used to send and receive special 3270 data streams between the CICS terminal and the application programs where they are interpreted. To transport the data from the terminal to the CICS region, the 3270 interface uses the TCP/IP protocol.

A second method to inquire the customer records is explained in chapter 5 “The MQSeries CICS Business Application MQNACT” on page 81. The created business application MQNACT also uses the COBOL business logic created for the CICS application NACT, but has another presentation logic component written in JAVA. We take advantage of JAVA's principle: Write once, run anywhere. That means, that anywhere, where a JVM is installed, the MQSeries CICS business application can be executed. In case, the code is transformed into a JAVA applet or servlet, it can be included as part of a web-based application running in a web browser. Exchanging the inquired data between the presentation logic and the business logic is done by the Messaging and Queuing method also referred to as commercial messaging. IBM has developed a product called MQSeries. We use this product in our sample application to explain the mechanisms of sending and receiving messages which transport data streams from a client computer system (WINDOWS2000) to a server computer system (OS/390). On both systems, MQSeries is installed as a server version connected through the TCP/IP protocol. When the JAVA program inquires a customer record stored to the OS/390 database-like file, it sends the data to the MQSeries Server installed on the client computer system by using the Message Queue Interface (MQI). The data is then packaged in a message and send to the OS/390 MQSeries Server. Between this MQSeries Server and the CICS TS another
interface is used to transfer the inquire data to the business logic programs and to transfer the response data back to MQSeries. This interface is called as the MQSeries CICS Bridge, sometimes also called as the MQSeries CICS DPL Bridge. After the response data is transferred from the OS/390 MQSeries Server to the MQSeries Server on the client computer system, it is picked up by the JAVA program using the MQI, too, and the information is displayed.

In contrast to the CICS application NACT, the MQSeries CICS application MQNACT only read the information of the customer account!

Figure 4: The initial architecture of the CICS business applications NACT and MQNACT
THE CICS BUSINESS APPLICATION NACT

A Bank Customer Account Program for the 3270 Terminal

4.1 Introduction

We want to create a CICS application for a bank that should perform operations on a customer account stored in a data base. All employees of the bank should be able to create, read, update, delete, and browse such an account. Furthermore, the customer accounts should be accessible through a menu-controlled program running on a CICS terminal supporting IBM's 3270 Display System, in that case the 3278 Display Station Model 2.

Such a CICS business application consists always of three key elements – components, transactions, and error handling. There are two key components of any business application – the business logic and the presentation logic. The presentation logic is an interface to use the business logic. In contrast, the business logic does the real work. Presentation and business logic can be written in any programming language the operating system understands and subsystems can call. The programs for the CICS application NACT are written in COBOL and use BMS and can be accessed by the CICS subsystem using transactions – the second element. Error handling is started if an error occurs during processing. This third key element is represented by a distinct COBOL program which is part of the business logic.

This chapter covers the description and the installation of both the business logic and the presentation logic. It explains data structures used for administration of the customer accounts and how to install these data on the OS/390-server. Furthermore, it explains how to deploy the application to CICS and how to display the result on a terminal screen when using the 3270 interface.

All data, the source code and the compiled code, are available as raw material on a CD as a part of Horswill's textbook [HOR00].
4.2 Uploading the files

The approach to transfer the files from the CD to the OS/390-server as described in Appendix A of [HOR00] does not work using the standard FTP-terminal included as part of Windows 2000. This terminal does not understand the suggested command `bin fixed 80` to transfer files that have a length of 80 bytes per record stored in fixed blocks. Partitioned data sets (PDS) are always consist of fixed record blocks with a length of 80 bytes per record, but data cannot be transferred directly into PDSs. Those files have to be firstly transferred into sequential data sets (SDS) to receive them later into PDS. Therefore, the SDS has to be set to fixed record blocks. Instead of this command, following command procedure can be used on the FTP-terminal to transfer the data onto OS/390. Firstly, the command `bin` has to be executed. Afterwards, an additional command is to be entered to create the fixed record block SDSs on the OS/390-server (Figure 5, page 44):

```
quote site recfm=fb lrec=80
```

The parameter `recfm` with the attribute `fb` specifies to create the fixed blocks. Within the parameter `lrec` is specified the length of 80 bytes per records. The whole block size is automatically set to 6160 bytes (refer to the messages shown in Figure 5). The file packages are copied from the CD (for folders stored to see Table 3) using the `put` command into a SDS template named as `TBUSSE.CICSADP.NEWSEQ` (Figure 5):

```
put cicsadp.loa cicsadp.newseq
```

It is not required to specify `tbusse.cicsadp.newseq`; the shorter form `cicsadp.newseq` can be used, because after logging on to the FTP-terminal, OS/390 always adds the user library name as high-level qualifier to a data set.

<table>
<thead>
<tr>
<th>Windows2000</th>
<th>OS/390</th>
</tr>
</thead>
<tbody>
<tr>
<td>File name</td>
<td>Stored on the CD-ROM into folder:</td>
</tr>
<tr>
<td>cicsadp.cpy</td>
<td>listings\chapter4\os390\cobol\compiled</td>
</tr>
<tr>
<td>cicsadp.csd</td>
<td></td>
</tr>
<tr>
<td>cicsadp.jcl</td>
<td></td>
</tr>
<tr>
<td>cicsadp.loa</td>
<td></td>
</tr>
<tr>
<td>cicsadp.mac</td>
<td></td>
</tr>
<tr>
<td>cicsadp.src</td>
<td></td>
</tr>
<tr>
<td>cicsadp.txt</td>
<td></td>
</tr>
</tbody>
</table>

Table 3: Files transferred from the accompanied CD-ROM onto OS/390

After a file is successfully transferred into the template data set it is switched to the OS/390 terminal to receive the data into the correct PDS. Entering the command `P.3.4` in the CUSTOMPAC MASTER APPLICATION MENU (CMAM) leads to the Data Set List Utility (DSLU) of the Interactive System Productivity Facility/Program Development Facility (ISPF)/(PDF) (Figure 6, page 44). Entering the user name `TBUSSE` into the field `Dsname Level` and pressing the enter key list the data sets owned by TBUSSE (Figure 7, page 45).
When placing the cursor on the first position of the line beginning with the data set name \textit{TBUSSE.CICSADP.NEWSEQ}, the following command is entered to receive the data (Figure 8, page 45):

\begin{verbatim}
receive indname(/)
\end{verbatim}

The token / in the emphasis marks the data set in this line. It is noticed that the INSERT key on the keyboard must be activated to overwrite the line. Afterwards, the message INMR901I indicates the data set is ready to be restored into a new data set. The next message INMR906A demands to enter the restore parameters – this refers to the data set in which the files should be copied (Figure 9, page 46):

\begin{verbatim}
dsn('TBUSSE.CICSADP.LOADLIB')
\end{verbatim}

During executing the command a few messages are listed. After pressing the enter key and returning back to the DSLU the first data set has been successfully uploaded from \textit{cicsadp.loa} into the data set \textit{TBUSSE.CICSADP.LOADLIB}. Subsequently, the SDS template may be overwritten by the other files that have to be transferred to OS/390 (Table 3). For transferring and receiving the file the same procedure is used, starting with the package \textit{cicsadp.cpy}. This is to be restored into its target data set as it is listed in Table 3.

All files are received into PDSs, except one file that remains as an SDS named as \textit{TBUSSE.CICSADP.VSAMDATA} (Table 3). For it, the \textit{quote site} command needs to be executed again on the FTP-terminal. A length of 383 bytes per record has to be set in the \textit{lrec1} attribute:

\begin{verbatim}
quote site recfm=fb lrec=383
\end{verbatim}

The records (account details) specified in the file \textit{cicsadp.txt} have a length of 383 characters. Therefore, the file is uploaded into an SDS having 383 bytes per record in one block. At the end, six new PDSs and one SDS are listed in the Dslist Utility (Figure 10, page 46).

Following files are stored in the partitioned data sets:

\begin{itemize}
  \item \textbf{TBUSSE.CICSADP.COBCOPY}: NACCBRWS, NACCCRUD, NACCERRH, NACCTREC, NACTSET, NACWBRWS, NACWCRUD, NACWERRH, NACWLITS, NACWLOCK, NACWTREC
  \item \textbf{TBUSSE.CICSADP.COBSRCE}: NACTSET, NACT01, NACT02, NACT03, NACT04, NACT05
  \item \textbf{TBUSSE.CICSADP.CSDEFS}: CICS0ADP, CICSJADP
  \item \textbf{TBUSSE.CICSADP.JCLLIB}: VSAM
  \item \textbf{TBUSSE.CICSADP.LOADLIB}: NACTSET, NACT01, NACT02, NACT03, NACT04, NACT05
\end{itemize}

After all files are transferred and received correctly, the SDS \textit{TBUSSE.CICSADP.NEWSEQ} may be deleted (Figure 11-Figure 14, page 47-48).
Generation of a Java front end for a standalone CICS application accessed through MQSeries &
Securing CICS with RACF

Figure 5: FTP session – Uploading a file into a SDS

```
C:\>ftp 139.18.4.97
Connected to 139.18.4.97.
220-FIPDM IBM FTP CS V2.97 at TIUSW07, 04:56:13 on 2004-02-05.
200 Connection will close if idle for more than 5 minutes.
User (139.18.4.97:(none))= TBUSSE
331 Send password please.
Password:
230 TBUSSE is logged on. Working directory is "TBUSSE."
ftp> bin
200 Representation type is Image
ftp> quote site vendor=Th lmac=30
200-BLOCKSIZE must be a multiple of LRECL for RECFM FB
200-BLOCKSIZE being set to 6160
200 Site command was accepted
ftp> lcd /new.
Local directory now /new.
ftp> put cicsadj.log cicsadj.newseq
200 Put request OK.
125 Storing data set TBUSSE.CICSADP.NEWSEQ
250 Transfer completed successfully.
ftp> 57680 bytes sent in 2.92 seconds 23.15Kbytes/sec.
```

Figure 6: DATA SET LIST UTILITY screen in ISPF/PDF
### Figure 7: DSLIST-screen in ISPF/PDF

<table>
<thead>
<tr>
<th>Command</th>
<th>Message</th>
<th>Volume</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBUSSE.CICSADP.NEWSEQ</td>
<td>SMS002</td>
<td></td>
</tr>
<tr>
<td>TBUSSE.DDIR</td>
<td><em>VSAM</em></td>
<td></td>
</tr>
<tr>
<td>TBUSSE.DDIR.D</td>
<td>DAVR7A</td>
<td></td>
</tr>
<tr>
<td>TBUSSE.DDIR.I</td>
<td>DAVR7A</td>
<td></td>
</tr>
<tr>
<td>TBUSSE.ISPF.ISPPROF</td>
<td>SMS001</td>
<td></td>
</tr>
<tr>
<td>TBUSSE.SPFLIST</td>
<td>SMS002</td>
<td></td>
</tr>
<tr>
<td>TBUSSE.SPFTemp1.CNTL</td>
<td>SMS002</td>
<td></td>
</tr>
<tr>
<td>TBUSSE.SPFLIST</td>
<td>SMS002</td>
<td></td>
</tr>
</tbody>
</table>

******************* End of Data Set list *******************

### Figure 8: Receiving the files from the data set “TBUSSE.CICSADP.NEWSEQ”

<table>
<thead>
<tr>
<th>Command</th>
<th>Option</th>
<th>F1=Help</th>
<th>F3=Exit</th>
<th>F5=Rfind</th>
<th>F12=Cancel</th>
</tr>
</thead>
<tbody>
<tr>
<td>receive indsname(/)SADP.NEWSEQ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TBUSSE.DDIR</td>
<td><em>VSAM</em></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TBUSSE.DDIR.D</td>
<td>DAVR7A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TBUSSE.DDIR.I</td>
<td>DAVR7A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TBUSSE.ISPF.ISPPROF</td>
<td>SMS001</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TBUSSE.SPFLIST</td>
<td>SMS002</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TBUSSE.SPFTemp1.CNTL</td>
<td>SMS002</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TBUSSE.SPFLIST</td>
<td>SMS002</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

******************* End of Data Set list *******************
Generation of a Java front end for a standalone CICS application accessed through MQSeries & Securing CICS with RACF

![Figure 9: Receiving the files into the new data set "TBUSSE.CICSADP.LOADLIB"](image)

![Figure 10: List all received data sets (in blue colour)](image)
Figure 11: Deleting the SDS template “TBUSSE.CICSADP.NEWSEQ”
Generation of a Java front end for a standalone CICS application accessed through MQSeries &
Securing CICS with RACF

---

**Confirm TSO Delete**

- **Data Set Name:** TBUSSE.CICSADP.NEWSEQ
- **Volume:** SMS002

**Caution**
If TSO delete command was issued against an uncataloged data set, a cataloged version on a volume other than the one listed here may be deleted.

- **Creation date:** 2002/09/15
- **Command:** DELETE 'TBUSSE.CICSADP.NEWSEQ'
- **Instructions:**
  - Press ENTER key to confirm delete request.
  - Press CANCEL or EXIT to cancel delete request.

**IDC0550I ENTRY (A) TBUSSE.CICSADP.NEWSEQ DELETED**

---

**Figure 14:** Message also appeared in the DLIST utility
4.3 The NACT COBOL programs and their commands

4.3.1 Overview

The business application programs running on a CICS Transaction Server v1.3 are *NACT01, NACT02, NACT03, NACT04*, and *NACT05* (Figure 15, next page). These CICS COBOL programs are transferred as compiled and as source versions onto the OS/390-server. Besides these five CICS COBOL programs, a compiled and a source file named *NACTSET*, containing one part of the presentation logic in BMS language, have also been transferred (refer to page 43).

A set of such application programs is also known as a Unit of Work (UoW) and can transfer an unlimited number of messages. The programs are not standard COBOL programs but so-called CICS COBOL programs. Differing from regular COBOL programs, the files to be accessed are not defined in the CICS COBOL programs themselves. They are stored in a CICS table – the File Control Table (FCT). This table links any access from the terminal to the right file stored on OS/390. In addition, some COBOL file processing verbs, console I/O verbs, statements, and compiler functions may not be used. A CICS COBOL program is usually short-running and processes only a few requests. All used program commands are explained in the next sections and are listed together in the Table 4 on page 56.

The COBOL copybooks, that belong to the COBOL programs, are also transferred onto the OS/390-server (refer to Table 3, page 42 and to the data set *TBUSSE.CICSADP.COBCOPY*, page 43). COBOL copybooks are libraries and act somewhat like functions or subroutines. They contain a segment of often used COBOL code for use by the complete application. During execution of the COBOL compiler these copybooks are introduced into COBOL programs like C and C++ include files into C or C++ programs.

4.3.2 The presentation logic component

When the transaction identifier (TRANSID or TRID) NACT is entered on the CICS terminal, the program *NACT01* is started. This program transfers data to the business logic, and, when it gets return data, it sends them to the map set *NACTSET*, which displays the matching screens on the terminal using the 3270 interface (Figure 15). There are three screens, called the ACCOUNTS MENU screen (Figure 16, page 51), the ACCOUNTS DETAILS screen (Figure 17, page 51), and the ERROR REPORT screen (Figure 18, page 52). Those screens are created using the Basic Mapping Support (BMS) services – the CICS terminal input/output services. These services constitute the native CICS presentation logic. The map set *NACTSET* contains for the each screen a map that contains multiple fields. The map set has been written in a special BMS language (which is in reality an S/390 assembler language macro facility). It is assembled with the help of the Job Control Language (JCL) into a physical map and into a symbolic description map. The physical map named as *NACTSET* is stored in the execution library *TBUSSE.CICSADP.LOADLIB* but contains no executable code. This part of the map set holds attrib-
utes, labels, and titles for the screen image and adjusts the variable data to display all information correctly on the screen (cf. Listing 11, page 227). The symbolic description map defines variable fields to refer them by name. It must have the same name as the physical map and is compiled as a COBOL copybook into the data set TBUSSE.CICSADP.COBCOPY. It is introduced into NACT01 as an include structure (cf. Listing 12, page 227).

Figure 15: Summary of the components of the bank customer account application NACT
ACCOUNTS MENU

TO SEARCH BY NAME, ENTER SURNAME AND IF REQUIRED, FIRST NAME

SURNAME : (1 TO 18 ALPHABETIC CHRS)
FIRST NAME : (1 TO 12 ALPHABETIC CHRS OPTIONAL)

TO PROCESS AN ACCOUNT, ENTER REQUEST TYPE AND ACCOUNT NUMBER

REQUEST TYPE: (D-DISPLAY, A-ADD, M-MODIFY, X-DELETE, P-PRINT)
ACCOUNT : (10000 TO 79999)
PRINTER ID : (1 TO 4 CHARACTERS (REQUIRED FOR PRINT REQUEST))

ENTER DATA AND PRESS ENTER FOR SEARCH OR ACCOUNT REQUEST OR PRESS CLEAR TO EXIT

Figure 16: ACCOUNTS MENU screen

ACCOUNTS DETAILS OF ACCOUNT NUMBER 11100

SURNAME : BUSSE (18 CHRS) TITLE : Mr (4 CHR OPTIONAL)
FIRST NAME : TOBIAS (12 CHRS) MIDDLE INIT: (1 CHR OPTIONAL)
TELEPHONE : 0000245756 (10 DIGS)
ADDRESS LINE1: Hursley Park (24 CHRS)
LINE2: Hants (24 CHRS)
LINE3: UK (24 CHRS OPTIONAL)
CARDS ISSUED : 1 (1 TO 9) CARD CODE : C (1 CHR)
DATE ISSUED : 01 12 01 (MM DD YY) REASON CODE: N (N,L,S,R)
APPROVED BY : IBM (3 CHR)
UPTO 4 OTHERS WHO MAY CHARGE (EACH 32 CHRS OPTIONAL)
O1: FRAU ELSTER
O2: RITTER RUNKEL
O3: PITTIPLATSCH
O4: DIE DIGEDAGS
SPECIAL CODE1: 1 CODE2: 2 CODE3: 3 (EACH 1 CHR OPTIONAL)
NO HISTORY AVAILABLE AT THIS TIME CHARGE LIMIT 1000.00 STATUS N

NOTE:- DETAILS IN BRACKETS SHOW MAXIMUM NO. CHARACTERS ALLOWED AND IF OPTIONAL
PRESS "CLEAR" OR "ENTER" TO RETURN TO THE MENU WHEN FINISHED

Figure 17: ACCOUNTS DETAILS screen (example)
ACCOUNT FILE: ERROR REPORT

ERROR AT 27/09/2002 12:40:18

Error in transaction NACT, program NACT05.
Type is TRAP. Response is NOTOPEN, Reason is 0060 - STARTBR.

PLEASE ASK YOUR SUPERVISOR TO CONVEY THIS INFORMATION TO THE OPERATIONS STAFF.

THEN PRESS "CLEAR". THIS TERMINAL IS NO LONGER UNDER CONTROL OF THE "NACT" APPLICATION.

Figure 18: ERROR REPORT screen (example)

Three map-definition macros are needed to generate a particular map. One of these is the BMS map definition for a field introduced by the DFHMDF macro, for example (cf. Listing 11, page 227):

```
017 DFHMDF POS=(01,01),ATTRB=(ASKIP,BRT),LENGTH=13, INITIAL='ACCOUNTS MENU'
```

This example defines that the characters 'ACCOUNTS MENU' are to be displayed on the ACCOUNTS MENU screen using the attribute INITIAL. The string is 13 characters long, thus the attribute LENGTH is set to 13. Furthermore, the string is to be displayed at the upper left position of the screen by setting POS=(01,01) (line no. 1 and column no. 1). There are also set predefined characteristics (ASKIP, BRT) in the ATTRB field. The map field macro may be prefixed with a field name (label) if it is needed to refer to it. An example that has the label SUMTTL is shown in line 56 in the same listing:

```
056 SUMTTL DFHMDF POS=(14,01),ATTRB=(ASKIP,BRT),LENGTH=79
```

The DFHMDF macros must be part of a BMS map definition introduced by the DFHMDI macro. It may also have a label, if more than one is defined. This is an example of such a macro:

```
016 ACCTMNU DFHMDI SIZE=(24,80),CTRL=(PRINT,FREEKB)
```

The macro starts with the identifier ACCTMNU. The parameter SIZE(height, length) defines the size of the screen, in this case, the map fills the whole displayable area of a CICS terminal with 24 lines (height 24) and 80
characters per line (length 80). Characteristics of the 3270 terminal are defined by the item \textit{CTRL}. For example, the characteristic \textit{PRINT} has to be specified if a printer is to be started to receive the data. The characteristic \textit{FREEKB} unlocks the keyboard after data is written to the map. All maps are assembled in the BMS map set definition using the DFHMSD macro. This macro must be placed ahead of all the other macros. The DFHMSD macro is shown in line 1 of Listing 11:

\begin{verbatim}
001 NACTSET DFHMSD TYPE=MAP,MODE=INOUT,LANG=COBOL,TIOAPFX=YES,STORAGE=AUTO
\end{verbatim}

A map set is always introduced by a required notation (here: \texttt{NACTSET}) to create the right map set. The attribute \texttt{TYPE=MAP} specifies that a physical map set is firstly to be created. After that, the symbolic map description is assembled when the parameter \texttt{MAP} is replaced by \texttt{DSECT}. Both compiled versions of a map set get always the same name – \texttt{NACTSET}. The physical map is stored in that data set into which all other installed programs are compiled to – in that case into \texttt{TBUSSE.CICSADP.LOADLIB}. In contrast, the symbolic map description is always stored into the copybook data set. Because the map set is used for both directions – input and output – the attribute \texttt{MODE} was set to \texttt{INOUT}. The \texttt{LANG} characteristic is needed to define the source language of the application program into which the symbolic description map should be introduced. Therefore, the map was assembled into a data definition for the COBOL program \texttt{NACT01 (LANG=COBOL)}.

After \texttt{NACT01} creates a menu map using BMS it sends the map, for instance the ACCOUNTS MENU map, to the CICS terminal with the COBOL command \texttt{EXEC CICS SEND MAP} (e.g. line 1217 in Listing 13). In the case of sending control information like erasing all unprotected fields or freeing the keyboard, the command \texttt{EXEC CICS SEND CONTROL} is executed (line 597 in Listing 13). Any data the program \texttt{NACT01} receives from a terminal is collected within the command \texttt{EXEC CICS RECEIVE MAP} (e.g. line 619 in Listing 13). Such data can be entered on the ACCOUNTS MENU screen and on the ACCOUNTS DETAIL screen. Before \texttt{NACT01} transfers data to the business logic programs, it validates the user account and all input data. If an error occurs the error handling program \texttt{NACT04} is called. In this case \texttt{NACT04} identifies the problem and sends an error message back to \texttt{NACT01}. An ERROR REPORT screen containing this error message is created and the map is transmitted to the CICS terminal. Input can also be transferred by pressing special keys on the terminal, for example, the ENTER key, the CLEAR key, or the PA/PF keys. Processing input from a terminal to a program is controlled by the EXEC Interface Block Attention Identifier (EIBAID). For a better understanding on how to use EIBAID it is suggested to browse the Appendix A in [APR03].

Furthermore, \texttt{NACT01} can start the program \texttt{NACT03} (the print presentation logic) when the transaction NACT has requested the NACP transaction (refer to Figure 15). A new task is started by using the command \texttt{EXEC CICS START TRANSID} (line 1040 in Listing 13). The data to be printed is retrieved from a storage area called communication area (COMMAREA) using the command \texttt{EXEC CICS RETRIEVE}. This program is not be used by the application since no printer is installed on the JEDI OS/390-server.
For more information about printing see Horswill's textbook [HOR00], chapter 11 “Local Printing (NACT03): Requests for Printing” on pages 241-242. All mentioned CICS COBOL commands are fully referenced in chapter 1 of [APR03]. Information about BMS, macros, items, and characteristics that can be set is available in [APG03], part 6 “Basic Mapping Support”. For BMS details see also [APR03] Appendix J, and K.

### 4.3.3 The business logic component

The programs NACT02, NACT04, and NACT05 are the business logic component of the CICS application. The NACT02 and NACT05 programs act as stateless tasks on their own reacting to each request. NACT04 is invoked if an error occurs during the program execution. An overview on how the business logic programs cooperate with the presentation logic programs is shown in Figure 15 on page 50.

NACT02 is named as the CRUD program (Listing 14, page 227). This abbreviation stands for its operations on a record of the account file (CREATE, READ, UPDATE, and DELETE). When the program is called it validates and analyses the request to decide to accept it or not. After NACT02 accepts a request the CRUD-operations are performed. If a record is to be added to, or deleted from the account file, or the record is to be changed the file must be locked to prevent concurrent updates. The lock is freed if one of these operations (create, update, and delete) is successfully executed. In case of an error, a message is sent from the error handler to the caller. The operation is aborted.

The CICS COBOL command `EXEC CICS WRITE` listed in NACT02 (e.g. line 359 in Listing 14) adds a new record to the account file (CREATE). If a record is to be updated the command `EXEC CICS REWRITE` (e.g. line 502 in Listing 14) is executed to overwrite this entry (UPDATE). However, at first, it is necessary to perform a “read for update” operation using the `EXEC CICS READ UPDATE` command (e.g. line 482 in Listing 14). This command searches the account file for the record that is to be updated. The CICS command `EXEC CICS DELETE` (e.g. line 438 in Listing 14) removes a record from the account file (DELETE). The READ operation `EXEC CICS READ` (e.g. line 408 in Listing 14) is only an inquiry function and does not initiate locking of the account file. In most cases the related command `EXEC CICS READ UPDATE` expects an update of the account file and thus requires locking the file.

NACT05 browses the account file for a matching record using the sequential method. It is searched by name, based on a user input (Listing 15, page 227). The browse process starts with the CICS command `EXEC CICS STARTBR` and terminates with the command `EXEC CICS ENDBR` (e.g. line 376 resp. Line 487 in Listing 15). After the start browse command was invoked only one match is reported, assuming a match exists in the file. NACT05 however assumes that more than one match is stored in the account file. Therefore, the command `EXEC CICS READNEXT` (e.g. line 529 in Listing 15) is executed to search for additional matches. The results are displayed on the screen. If no record matches the search criteria, a message indicating this fact is displayed. NACT05 does not require locking the account file because the browse process uses only the READ operation and expects no UPDATE, CREATE, or DELETE operation.
The error handler program *NACT04* is called when an error or problem occurs during executing the CICS application NACT (Listing 16, page 227). It must handle a comprehensive scenario that can happen or might happen in future. Nobody can predict every error but if such an unpredictable error occurs a message must sent to the user to inform him or her. For example, problems could be caused be either the application or by CICS itself. Consequently, *NACT04* obtains any information about the current problem, builds and sends messages to the user appropriate to the type of the error, and terminates the task. In any event, to meet the ACID criteria, the intended modification is rolled back.

The first command *NACT04* executes is *EXEC CICS ASSIGN* (line 954 in Listing 16). It passes information from CICS about the environment and the problem back to the program. Besides, it determines which command was used to invoke the error handler. That can happen when a program, for instance *NACT01*, invokes the commands *EXEC CICS XCTL* or *EXEC CICS LINK* and a serious problem occurred thereby (e.g. line 2129 resp. line 2441 in Listing 13). As a result of a CICS abnormal end processing, the programs call *NACT04* using the command *EXEC CICS ABEND* (*NACT01*, Listing 13, line 463). After *NACT04* receives the control it has to confirm that the program which called the error handler is the correct one, and if not, to find the program where the error occurred using the command *EXEC CICS INQUIRE PROGRAM* (e.g. line 1173 in Listing 16). When an error occurs a dump is taken with a code based on the type of the error using the command *EXEC CICS DUMP TRANSACTION DUMPCODE* (e.g. line 1343 in Listing 16). This excludes a conflict with the CICS dump codes. When a transaction fails and an error is produced, all updates on a record must be rolled back, and the previous version is restored with the command *EXEC CICS SYNCPOINT ROLLBACK* (e.g. line 1350 in Listing 16). Displaying the error message on the users terminal screen is done with the command *EXEC CICS WRITEQ TD QUEUE* (e.g. line 1382 in Listing 16). It writes extra-partition transient data (the error message) to a predefined symbolic destination (the queue) and if possible the message is displayed at a terminal. Executing the command *EXEC CICS WRITE OPERATOR TEXT* (e.g. line 1400 in Listing 16) sends the error message to a system console and thus to a system operator according to the CICS system definition. Additionally, a date and a time stamp can be added to the error message header with command *EXEC CICS ASKTIME* resp. *EXEC CICS FORMATTIME*, (e.g. line 1358 or line 1362 in Listing 16). The command *EXEC CICS ISSUE DISCONNECT* (e.g. line 1467 in Listing 16) is invoked to terminate the session on the CICS terminal.

All commands used within these CICS COBOL programs are listed in Table 4 on the next page. The CICS COBOL commands used in the business logic programs are fully referenced in chapter 1 of [APR03].
4.3.4 Other commands that control the programs

There are additional CICS COBOL commands to control the programs of the CICS application. Two commands previously mentioned – EXEC CICS LINK and EXEC CICS XCTL – pass control from one program to another using the scratchpad facility COMMAREA. Such program calls are referred to as “synchronous”. Using this method returns the control to the calling program when the call is completed. The LINK command transfers data to another program in a synchronous manner using the COMMAREA and expects a return to continue the execution. The linked-to program, that is called through the LINK command, gives back the control to the start-
ing point – the instruction that resides after the LINK command. In our sample application, the LINK command is used by NACT01. On the other side, the XCTL command also transfers control in a synchronous manner from one program to another using the COMMAREA, but it expects no return to call another program. This method is very useful if an unexpected error is detected and the program cannot continue its work. This command is used by all programs except NACT04. In connection with the LINK and XCTL commands the command EXEC CICS RETURN used by all programs (e.g. line 523 in Listing 13) returns the control back to the previous program. In addition, the control is given to the next transaction with the command EXEC CICS RETURN TRANSID (e.g. line 2388 in Listing 13).

In case of terminating a transaction because of an abnormal situation the command EXEC CICS HANDLE ABEND (e.g. line 456 in Listing 13) is invoked by all programs except NACT04 to establish the error handler program NACT04. The transaction is terminated by the command EXEC CICS ABEND ABCODE (e.g. line 463 in Listing 13). This command can give the control back to CICS or to the error handler NACT04. It causes to back out all changes made by this unit of work and the record file remains unchanged. Besides, it can produce a dump having the identifier given by the parameter ABCODE.

The command EXEC CICS ASSIGN is not only used in the NACT04 program to collect information about the CICS system and the occurred error, it also collects the system information for all other programs.

All these additional CICS COBOL commands are fully referenced in chapter 1 of [APR03].

### 4.3.5 Excursion: Storing exchange data – When to use the COMMAREA?

#### 4.3.5.1 Overview

The COMMAREA is a CICS standard area used by two CICS programs to exchange data within one transaction or between transactions. This data is saved temporary for a specific, mostly short, time.

Storage hierarchy on S/390 computers is subdivided into different storage levels; the highest at the top, the lowest on the bottom (refer to Table 5 on next page). Relevant for storing CICS exchange data are Processor Storage, consisting of Central Storage plus Expanded Storage, and External Storage. Processor Storage is also called Main Storage or Main Memory, and sometimes erroneously only Central Storage. However, Central Storage is a component of Processor Storage. External Storage is sometimes also called Auxiliary Storage. Processor Storage can be as large as 64 GB (z900), on the JEDI OS/390-server the size is 256 MB. External Storage on JEDI OS/390-server is provided by one internal disc with 18GByte and 16 external discs with 9.1GBytes per disc. The Processor Storage assigned to CICS is called CICS Main Storage, which is usually virtual storage. This virtual storage is subdivided into the CICS Dynamic Storage Area (CDSA) and the Extended CICS Dynamic Storage Area (ECDSA).

<table>
<thead>
<tr>
<th>Storage place</th>
<th>Storage level</th>
<th>Access time</th>
</tr>
</thead>
</table>


Generation of a Java front end for a standalone CICS application accessed through MQSeries & Securing CICS with RACF

<table>
<thead>
<tr>
<th>Inside Central Processor</th>
<th>Registers</th>
<th>few nanoseconds</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Level-1-Cache</td>
<td>nanoseconds</td>
</tr>
<tr>
<td></td>
<td>2nd-Level-Cache</td>
<td>nanoseconds</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inside Processor Unit</th>
<th>Processor Storage</th>
<th>many nanoseconds</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Central Storage (+ Virtual Storage)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Expanded Storage</td>
<td>microseconds</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>External Storage Devices</th>
<th>Cache Disk Storage on a Direct Access Storage Devices (DASD)</th>
<th>milliseconds</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DASD without Cache</td>
<td>milliseconds</td>
</tr>
<tr>
<td></td>
<td>Optical Storage</td>
<td>milliseconds</td>
</tr>
<tr>
<td></td>
<td>Tape Storage</td>
<td>seconds</td>
</tr>
</tbody>
</table>

Table 5: Storage hierarchy in S/390 computers
(see [HAC99], p.52)

There exist two storage services for storing exchange data from the execution of a CICS transaction resp. CICS program and to share it with other transactions/programs – the scratchpad facility and the queuing facility. Both storage services are used in the CICS business application NACT. Saving the state of an interaction between the terminal user and a program results in the use of such facilities. The scratchpad facility is used when the state of program data is to be saved during program execution (without a task exit). When data is store to a file or to a list, that are referenced later (for example for logging), queuing facility is used (a task exit is possible). CICS scratchpad facilities uses scratchpad areas, which can be one of the following: COMMAREA, Transaction Work Area (TWA), Common Work Area (CWA), and Terminal User Area (TUA). Display Screen can also be used as a scratchpad area to store data for a time. Scratchpads use only CICS Main Storage. The other storage service as already mentioned above is called queuing facility. As it, Temporary Storage and Transient Data can be used. Contrary to scratchpad areas, both can use CICS Main Storage and External Storage (Figure 19, next page).

The CICS storage services can further be subdivided into those associated with a single task or within shared tasks. These types are called Task-Private Storage (also known as Task-Private Pool), Task-Shared Storage (also known as Task-Shared Pool), and Private-Shared-Storage (also known as Region Pool) only owned by CICS. Task-Private Storage is private to the task and cannot be addressed by any other CICS task in the system. The COMMAREA and the TWA can use the Task-Private Storage. In contrast, the data stored in the Task-Shared Storage can be shared between all CICS tasks. This storage can be used by the CWA, the TUA, and by the COMMAREA, too. In the CICS Private-Shared-Storage is stored data private to CICS, only information from this area can be obtained.
4.3.5.2 Temporary Storage – queuing and scratchpad facility

Temporary Storage is the primary CICS facility for storing data that must be available for more than one transaction. It can use both – CICS Main Storage and External Storage of the OS/390-server, but using CICS Main Storage requires much more storage for Temporary Storage than External Storage requires. CICS exchange data is kept in Temporary Storage Queues (TSQ) (Figure 19). When using CICS Main Storage the space for the queues is taken from the CDSA, part of the Virtual Storage; when using the External Storage the queue is written as a sequential file stored in a VSAM data set. Data stored in External Temporary Storage implies "storing relatively large amounts of data that have a relatively long lifetime or are accessed infrequently." ([APG03], ch. 3.1.5)

The Temporary Storage Data Sharing holds both TSQ-types in Temporary Storage Pools to support concurrent access to the queues. TSQ stored in data sets can be recovered, in contrast to TSQs stored in CICS Main Storage and to shared TSQs. Temporary Storage requires not to be allocate until it is required and it is only available as long as it is required. Security for TSQs is also available. The name of TSQs can be up to 16 characters long. Each queue requires a CICS resource definition called the Destination Control Table (DCT) entry.

TSQs consist of items that can be thought as a record. For the items on both TSQ types, a CICS internal index is created, maintained in the CICS Main Storage. The items can be added to the queue, read, modified, and deleted from the queue by CICS programs within one task or more tasks. Furthermore, the items can be reread and can be read in any order. The TSQ must not store more than 32.767 items resp. bytes. When an item is longer than 1 byte, for example 12 bytes, the TSQ must not exceeded 32.755 bytes. If only one item is stored onto the queue, the queue can be treated as a scratchpad capability (Figure 19). However, TSQs of more than one item should only be used for direct or repeated access to the items, because Transient Data provides facilities for a
better and efficient handling of sequential files when accessed sometimes. Because the queuing facility is only used in the CICS application NACT to send sometimes error messages to the user's terminal, Temporary Storage has not been used for the CICS application, but Transient Data does.

4.3.5.3 Transient Data – queuing facility

Within the second queuing facility – Transient Data – the CICS exchange data is also stored as items/records onto queues (Transient Data Queues (TDQ)) external or internal to the CICS region (Figure 19). If data is stored onto a Intrapartition TDQ (ITDQ), it can only be accessed by a facility allocated to the CICS region. If data should be sent external to the CICS region, it is stored onto an Extrapartition TDQ (ETDQ). ITDQs are similar to External TSQ, they use also sequential files stored in VSAM data sets, but managed by the CICS region. ITDQs cannot be created dynamically, items can only be read sequentially and only once, and an item cannot be changed. After the item is read, it is removed from the queue. Therefore, intrapartition data can be used for message switching, broadcasting, and database access (if the database is connected to the CICS region). Advantageously, ITDQs can be physically or logically recoverable.

In contrast to ITDQs, EDTQs can also reside on any external sequential devices, such as a DASD, terminal, tape, printer, and so on. Programs can access those queues outside or within the CICS region through the standard Sequential Access Method (SAM). Logging data, statistics, and transaction error messages are examples of data that can be written to EDTQs. Within our sample CICS application (NACT04), extrapartition data is used to display error messages on the terminal screen. The command EXEC CICS WRITEQ TD, as mentioned in chapter 4.3.3 “The business logic component” on page 54, sends the error information to an EDTQ named CSSL.

Both TDQs require a CICS DCT entry and a queue definition. The name of Transient Data queues may only be 4 characters long. Items, as on ITDQs, cannot be reread and modified and they can only be read in order as they were added onto the queues. Read and write operations on ETDQs cannot be done concurrently.

For more information on Temporary Storage and Transient Date please refer to [APG03], for a good explained storage hierarchy and for a complete storage history refer to [HAC99].

4.3.5.4 COMMAREA – a scratchpad facility

As part of the scratchpad facility, the COMMAREA stores temporary data to pass it to a called program within or of the next transaction (refer to Figure 19). A COMMAREA is a special form of user storage. With the use of COMMAREAs, CICS can store the exchange data in DSAs and Extended DSAs to provide them for the same CICS region, for different CICS regions, or for other address spaces external to CICS. When the COMMAREA is used in the EXEC CICS LINK and EXEC CICS XCTL commands, exchange data is transferred between programs of the same transaction or between transactions from the same terminal. Exchange data can be passed between different terminals when the COMMAREA is used in the EXEC CICS RETURN command. When us-
ing this command only the first program of the next transaction is called. CICS ensures, that any COMMAREA is addressable by the program, that receive the data. In our sample CICS business application NACT, all programs except NACT04 use the COMMAREA.

Each COMMAREA must not exceed a limit of 32.763 bytes including an additional header. However to be safe, it should not be exceeded a 24KB limit because of some factors that can reduce the limit from the maximum ([APG03], ch. 3.3.3.1). The length of the COMMAREA is stored in the EIBCALEN-variable (Execute Interface Block COMMAREA Length). The header may have a length of up to 18 bytes, consisting of an identifier of the target system (bytes 1-4), a program identifier for the called program name (bytes 5-12), a decimal return code (bytes 13-16), and the COMMAREA length inclusive of the header (bytes 17-18). For example, such a header applies to the COMMAREA in the XCTL and LINK command. Of course, the header of the COMMAREA used in the RETURN command consists of the transaction identifier (4 bytes) instead of the program name (8 bytes).

When a CICS COBOL program calls a COMMAREA, it sets a pointer to the address of the COMMAREA known to the system, and receives the data which was transferred from another program to it previously. Because the COMMAREA is freed before the next program starts, a copy of the data area is created and a pointer to the copy is passed to the called program.

Passing exchange data to a DPL program using a COMMAREA is the recommended method. In CICS COBOL programs, the command DFHCOMMAREA creates the COMMAREA.

A COMMAREA is used as a Task-Private-Storage, when information is passed between one program and the next in the same task. When information is passed in a pseudo-conversational sequence between one transaction and the next, the COMMAREA is used as Task-Shared-Storage.

In Figure 20 on the next page it is shown, that the program PR1 of the transaction TR1 transfers data to the program PR2 using the COMMAREA1. For example, the command EXEC CICS LINK PROGRAM('PR2') has been executed. After PR2 has received the data, the data is processed and sent to the COMMAREA2 executing the command EXEC CICS RETURN TRANSID('TR2'). PR3 receives the data because this program is the first program of the transaction TR2. When the data is executed, PR3 sends it to PR4 using the EXEC CICS LINK PROGRAM('PR4') command. PR5 also uses the EXEC CICS LINK command to pass data to PR4.
Example: Using the COMMAREA in the CICS business application NACT

As an example, we describe, how a communication area is created and used, when NACT02 is called for a read request from NACT01. Within the COBOL copybook NACWCRUD a logical record storing data has been defined (cf. Listing 17, page 228). From this record, also called working storage, the COMMAREA is built by the called program. The copybook defines a 25 bytes header and a 383 bytes buffer for the record. The name of the working storage – WS-CRUD-COMMAREA – is only set to refer to it from NACT01 (line 21, Listing 1, next page). All variables are only used to refer to them from NACT01. In the working storage there are only defined pure data.

The program name to be called captures the first 8 bytes of the data area. Because a CICS COBOL program can have maximal 8 characters and NACT02 consists only of 6, 2 blank characters have been inserted at the end of the program name (‘NACT02__’; cf. Lines 100-103, Listing 2, next page). This entry is automatically created, when the COBOL command EXEC CICS LINK is executed within the keyword PROGRAM. As next, a few entries have been created that hold some specific information. The first data item WS-CRUD-VER is set to 3 spaces (line 25, Listing 1). This item is filled with the data description entry WS-CRUD-CORRECT-VERSION to ‘1A’, if NACT01 sets this entry (line 26, Listing 1 & line 2127, Listing 2). The field WS-CRUD-FUNCTION is set to one space to store the request type which is sent to NACT02. The request type describes an associated operation (line 36, Listing 1). For example, the user wants to read an account without locking the file, the request type E is set within the variable WS-CRUD-REQ-ENQUIRE (line 41, Listing 1). The variable WS-CRUD-VALID-
The CICS Business Application NACT 63

21  05 WS-CRUD-COMMAREA.
...
25     10 WS-CRUD-VER   PIC XXX VALUE SPACES.
26     88 WS-CRUD-CORRECT-VERSION VALUE 'VIA'.
...
36     10 WS-CRUD-FUNCTION PIC X VALUE SPACES.
...
41     88 WS-CRUD-REQ-ENQUIRE VALUE 'E'.
44     88 WS-CRUD-VALID-REQUEST VALUE 'C' 'R' 'U' 'D'
     'E' 'L' 'F'.
...
56     10 WS-CRUD-RESP PIC 9(4) VALUE ZERO.
...
59     88 WS-CRUD-NO-ERROR VALUE '0000'.
60     88 WS-CRUD-BAD-FORMAT VALUE 'FRMT'.
...
76     10 WS-CRUD-REAS PIC 9(4) VALUE ZERO.
...
81     88 WS-CRUD-REQUEST-ERROR VALUE 'REQE'.
...
90     10 WS-CRUD-CICS-FUNCTION PIC 9(5) VALUE ZERO.
...
97     10 NACTREC-DATA.
98     COPY NACWTREC.

Listing 1: Extract from the COBOL copybook NACWCRUD (cf. Listing 17, page 228)

Note: COBOL uses different level numbers to indicate the name, the functions, and the fields of the
COMMAREA. Such level numbers indicate the hierarchical position of the data items (level numbers 1-
49) or the special properties of a data description entry (level numbers 66, 77, and 88). For example, the
level number “05” sets the name of the data area, “10” indicates a field, and “88” refer to a variables
having a value.

...
100  05 WS-PROGRAM-NAME PIC X(8) VALUE SPACES.
101  05 CRUD-PROGRAM.
102     10 WS-CRUD-PROGRAM-PREFIX PIC X(4) VALUE SPACES,
103     10 FILLER PIC X(4) VALUE '02 '.
...
306    COPY NACWCRUD.
...
2127    SET WS-CRUD-CORRECT-VERSION TO TRUE.
2128*
2129    EXEC CICS LINK
2130    PROGRAM(CRUD-PROGRAM)
2131    COMMAREA(WS-CRUD-COMMAREA)
2132    RESP(RESPONSE)
2133    RESP2(REASON-CODE)
2134    END-EXEC.
...

Listing 2: Extract from the CICS COBOL program NACT01 (cf. Listing 13, page 227)
REQUEST points to the same value as specified previously to validate the request type. For example, it points to the value E that is set for the read-access and allows the reading (line 44&45, Listing 1). Four more bytes are reserved for the response code values, set within the item WS-CRUD-RESP (line 56, Listing 1). There are also reserved four bytes for the reason code within WS-CRUD-REAS (line 76, Listing 1). These codes will set during passing the data to NACT02. For example, the entry WS-CRUD-NO-ERROR is set to '00000 in the field WS-CRUD-RESP, in case the request was successful (line 59, Listing 1), or WS-CRUD-BAD-FORMAT is set to 'FRMT', if there was an interface error (line 60, Listing 1). Furthermore, the reason code variable WS-CRUD-REQUEST-ERROR can be set to 'REQE' in the field WS-CRUD-REAS, if there was a request error (line 81, Listing 1). However, this reason code is only stored, if an interface error happened previously. The last 5 bytes of the header have been reserved for a numeric value containing the character representation of the EIBFN value giving rise to the exception condition (line 90, Listing 1). The EIBFN is of particular interest, because it shows the type of the last EXEC command to be issued by NACT01. In our case, the decimal number 3586 resp. the hex number E02 indicates the last performed command EXEC CICS LINK (EIBFN codes can be found in [CUH00]). At the end, the 5 bytes account number plus an empty 378 bytes buffer is added to the working storage to store the requested account record data. The variables of all these empty record fields have been defined within the copybook NACCTREC (line 97 & 98; for the copybook NACCTREC refer to Listing 18, page 228).

All in all, NACT01 creates the logical record for the COMMAREA, consisting of 408 bytes, when the COBOL copybook NACWCRUD is loaded within the COBOL command COPY (line 306, Listing 2). This record named as WS-CRUD-COMMAREA is passed to NACT02, when the COBOL command EXEC CICS LINK is executed using the keyword COMMAREA plus the name of the record (lines 2129-2134, Listing 2). For example, following record could be sent to NACT02 to built the COMMAREA (underlined blanks indicate the empty bytes):

```
NACT02__V1AE0000000358611100_________________________________________________________
_______________________________________________________________________________________
_______________________________________________________________________________________
_______________________________________________________________________________________
____________________________________________________________
```

where the values mean:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>NACT02</td>
<td>Program name to be called</td>
</tr>
<tr>
<td>V1A</td>
<td>“Eyecatcher” sent in this string to NACT02</td>
</tr>
<tr>
<td>E</td>
<td>Request type sent in this string to NACT02</td>
</tr>
<tr>
<td>0000</td>
<td>Fields for response code sent to NACT02</td>
</tr>
<tr>
<td>0000</td>
<td>Fields for reason code sent to NACT02</td>
</tr>
<tr>
<td>03586</td>
<td>Indicates the last performed command – EXEC CICS LINK</td>
</tr>
<tr>
<td>11100</td>
<td>The requested account number</td>
</tr>
</tbody>
</table>
From this data the COMMAREA is created, when the COBOL command `DFHCOMMAREA` is executed from the LINKAGE SECTION of the called program (line 192, Listing 3). This command only applies to CICS COBOL programs, if not specified, the compiler creates a default communication area consisting of 1 byte. The COMMAREA is built within the masks set in the COBOL copybook `NACCCRUD` and has the same format as it did in the passing program `NACT01` (line 193, Listing 3; for the copybook `NACCCRUD` refer to Listing 19, page 228). The name of the communication area is set to `CA-CRUD-COMMAREA` (line 22, Listing 4). This name is only used to call the data area from `NACT02` as same as calling all other fields defined by the copybook `NACCCRUD`. These variables have been introduced to assign correct names for the COMMAREA-fields to point them from `NACT02`. Before `NACT02` can build the COMMAREA from the record, it must know the address of the working storage. Within the COBOL command `SET` the system provides the address of the record (line 249 referring to line 96, Listing 3).

The first 3 bytes of the record behind the 8 byte program name refer to the first mask set in the copybook `NACCCRUD` – `CA-CRUD-CORRECT-VERSION` in the `CA-CRUD-VER` field (line 26 & 27, Listing 4). The value of `CA-CRUD-CORRECT-VERSION` has not been set by `NACT02` to `V1A`! It is only used to compare the passed value with the one specified in the mask (line 261, Listing 3). If the values are the same, it is checked whether the request type is valid or not (line 262, Listing 3 and lines 45 & 46, Listing 4). If it is valid, for example it points to the value `'E'` to read an account, a function is executed that indicates some fields to lock an account (line 263, Listing 3). After the request type has been transmitted it is evaluated which request type has been

---

```cobol
95      05 DEBUG-COMMAREA-ADDR USAGE IS POINTER.
...
191 LINKAGE SECTION.
192 01 DFHCOMMAREA.
193     COPY NACCCRUD.
...
249      SET DEBUG-COMMAREA-ADDR TO ADDRESS OF DFHCOMMAREA.
...
261      IF CA-CRUD-CORRECT-VERSION
262          IF CA-CRUD-VALID-REQUEST
263              PERFORM A-ANALYZE-REQUEST
264              EVALUATE TRUE
...
273              WHEN CA-CRUD-REQ-ENQUIRE
274                PERFORM C-READ-THE-RECORD
...
279              END-EVALUATE
408         EXEC CICS READ
409            FILE(WS-LITS-FILES-ACCOUNT)
410            RIDFLD(ACCTDO IN NACTREC-DATA)
411            INTO(NACTREC-DATA)
412            RESP(CA-CRUD-RESP)
413            RESP2(CA-CRUD-REAS)
414         END-EXEC
415      MOVE EIBFN    TO WORK-FN-X
416      MOVE WORK-FN TO CA-CRUD-CICS-FUNCTION
...
```

Listing 3: Extract from the CICS COBOL program NACT02 (cf. Listing 14, page 227)
passed into the COMMAREA (line 264-279, Listing 3). Because the request type 'E' has been passed referring to the COMMAREA-field CA-CRUD-REQ-ENQUIRE (line 42, Listing 4), a function called C-READ-THE-RECORD is performed to read the account record (lines 273 & 274, Listing 3).

Before passing the data through the COMMAREA to NACT01 back, the values for the response and reason codes will also be set as same as they was set for the record which was sent to NACT02. For example, in the field CA-CRUD-RESP entry CA-CRUD-NO-ERROR is set to '0000' because the request was successful (line 60, Listing 4). Furthermore, the reason code has also been set to '0000' because no error occurred. The EIBFN value stores the decimal number 1538 resp. the hex number 602 to indicate the last executed EXEC CICS command, in that case EXEC CICS READ.

At the end, the account record data is copied in the masks specified within the copybook NACCTREC (line 98 & 99, Listing 4). All data stored in the COMMAREA are received by NACT01 as a record consisting of the pure data, for example (for the terminal display please refer to Figure 17 on page 51):

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>NACT02__VIAE0000000000153811100BUSSE</td>
<td>TOBIAS Mr 0000245756Hursley Park RITTER</td>
</tr>
<tr>
<td></td>
<td>Hants UK</td>
</tr>
<tr>
<td>RUNKEL</td>
<td>FRU ELSTER</td>
</tr>
<tr>
<td>PITTIPLATSCH</td>
<td>DIE DIGEDAGS</td>
</tr>
<tr>
<td>101120INCIBM123N</td>
<td>1000.00</td>
</tr>
<tr>
<td></td>
<td>0.00000000</td>
</tr>
<tr>
<td></td>
<td>0.00000000</td>
</tr>
<tr>
<td></td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>0.00000000</td>
</tr>
<tr>
<td></td>
<td>0.00</td>
</tr>
</tbody>
</table>
4.3.5.5 Common Work Area, Transaction Work Area, and Terminal User Area – other scratch-pad facilities

The Common Work Area (CWA) is a fixed size data area created at CICS startup that exists only for one CICS address space during a CICS session. “The whole system, the format, and use of this area must be agreed upon by all transactions in all applications that use it.” ([HOR00], page 84) Because changing transaction definitions or program contents in future, affinities could cause major problems. Using the CWA is not regulated by CICS, all programs using the CWA must agree the rules for shared access. Data stored in the CWA is unrecoverable if a transaction or the system fails and it cannot be secured by the CICS resource level security. After creation, the CWA is always allocated; hence, it is not suitable for large data or short-time data. However, it is suitable for small amounts of data, that is read or updated by multiple programs, for example status information. The CWA must be prevented from any overload, that will corrupt all storage areas created in the CICS Task-Shared Storage.

As same as in the example for the COMMAREA, it is shown, that the program PR1 of the transaction TR1 transfers data to program PR2 using the CWA in the same address space (refer to Figure 20, page 62). After PR2 has received the data, it is processed, and is also sent to the CWA. PR4 of the transaction TR2 receives the data and executes it. The CWA uses only the EXEC CICS XCTL PROGRAM and EXEC CICS LINK PROGRAM commands to call the programs.

The Transaction Work Area (TWA) is used to pass data among programs executed in the same transaction not between programs of different transactions. Therefore, only one transaction with its program(s) can use this area. Because the TWA exists for the entire transaction, a large fixed size area (up to 32,767 bytes) has a greater benefit for the conversational than for pseudo-conversational transactions. The TWA uses the EXEC CICS XCTL PROGRAM and EXEC CICS LINK PROGRAM commands to call the programs. In the example shown in Figure 20, data can only be transferred within the one transaction TR1 or TR2. For example, the data can be exchanged between program PR1 and PR2 using the TWA1, or the programs grouped in the transaction TR2 use the TWA2 to exchange the data among each other.

As another scratchpad facility the Terminal User Area (TUA) is allocated at log on to CICS for both, autoinstalled and non-autoinstalled terminals (please refer to Figure 19 on page 59). It is similar to the CWA, but this area is only shared among the transactions using that terminal until log out. It can be used by pseudo-conversational transactions for small amount of data that should only be exchanged during a terminal session.

The TWA, CWA, TUA, the Temporary Storage, and Intrapartition Transient Data have not been used for the CICS application NACT. Contrary, the COMMAREA and the Extrapartition Transient Data have been chosen.
4.4 Storing the data – account file, locking file, and name file

4.4.1 File description

The account file stored on OS/390 saves the customer information into data records. Each customer has an assigned account number. This number has to be entered into the ACCOUNT MENU screen on the CICS terminal by the employee of the bank to get an access to the data. Subsequently, the transaction NACT sends the data to NACT01 which sends this request over NACT02 to the account file TBUSSE.CICSADP.ACCTFILE (cf. Figure 15, page 50). The account file is stored as a Key-Sequenced Data Set (KSDS) organised with the Virtual Storage Access Method (VSAM KSDS). This method was introduced by IBM in the 1970s for systems having virtual storage and to use this storage. A VSAM is created with the VSAM definition language Access Method Services (AMS). The data of a VSAM data set is managed in a VSAM catalogue – the central information point. In this catalogue are stored the physical properties of the data set, for instance, the maximal record length, or the position of the key within the record. This simplifies programs because they can take the information from the catalogue to process the data and not from the application program itself. VSAM data sets are referred to as clusters. A VSAM KSDS cluster consists of two physical parts, a data component, and an index component. Other VSAM data sets are Entry-Sequenced Data Sets (ESDS) and Relative-Record Data Sets (RRDS). They are also referred to as clusters but use other access methods. ESDS and RRDS clusters consist of only of the data component.

Each record in the data component of a KSDS cluster contains a key field, which must be the same length and occurs in the same relative position in each record. Based upon their key field values the records are stored in logical sequence in the data component. The index component of the KSDS cluster contains the list of key values for the records in the cluster with pointers to the corresponding records in the data component. Usually, a KSDS was used as a database by IMS. For more information about VSAM and non-VSAM data sets and how to use AMS see [UDS00], [SPR77], and [MOS03].

The account file is accessed through the Keyed Access Method using the account number as key. Because all CRUD operations can be executed on the account file a separate external data base is not needed. Consisting of 22 fields/entries the account file is distinguished into some fields with read access and a few to be modified. Important data fields to be read and modified are the account number, the surname, and first name, address, and a debit card code. Data fields with only read access are account status and charge limit. Fields for future use, in case the debit card is used, are balance, bill date, bill amount, date, and amount paid. It is not necessary to define these record fields because the format of the entries are provided by the KSDS.

The account file must be prevented from concurrent updating a record field. When a user modifies an account record other users must be detained from a concurrent update, so long as the first user frees the transaction/record ("Isolation" of ACID). To prevent this, an account locking file is created. The file called TBUSSE.CICSADP.ACTINUSE notices the account number of the record that is being updated and the user ID.
with the associated terminal ID. The locking file is also a VSAM KSDS. In contrast to the account file, it only consists of six data fields.

When a user wants to update an account record it is firstly checked whether the file is in use by another process. The NACT02 program searches the locking file for the stored account number. If found, an error message is sent to the user terminal with the information that the modifying is not allowed. At the end, the process is aborted. But it is conceivably that a record is locked for an unlimited time. The ACID criteria implies that transactions may not wait for any user input. They should only be in use for a “short” time. Therefore, an account record has to be updated in a specified period of time. After this time is elapsed, the account record is freed, and other users can modify it. It does not care whether the first user has ended the update process. Another user who is waiting owns now the record and can modify it. The current time and date of the update are also stored into the locking file.

Unfortunately, freeing an account record after the time has expired, dislodge the business application a bit from the main property “Isolation” of the ACID criteria for transactions. The record is only isolated in the specified time. For the application it is decided to rate the property “short” slightly more than the property “Isolation”. Because in an environment, where multiple users modify records, the locking scheme for this problem is a better solution than abiding the ACID criteria absolutely. An example:

When user Barbie tries to update a record, which is updating by user Ken, she gets a message to try again later and her update process is declined. Why? The program scanned the locking file and detected that the record is in use by Ken. He has a short period of time to complete his update process. As soon as he has modified the record it is freed and the account number is deleted from the locking file. After that, Barbie may modify the same record.

Supposed, that Ken is modifying a record and he is going to lunch for an hour in the meantime. He forgets to finish the update. Since he stays too long away, his owning time is elapsed and the update process is abandoned and backed out. Ken loses his possession right of the account record and the account number is erased from the locking file. Thus the record is freed. Meanwhile, Barbie wants to update the same account record which is now locked by her. The account number is again stored into the locking file until the record is freed again. At this point, Ken can only read the account data. But, when Ken comes back he sees the same screen displaying all changes he made as he left. He finishes his update process but the changes are not saved to the account file because he does not own the record at this time. Don’t worry, his changes are not loss, they are stored to the transient data.

All in all, the record updates of Barbie are saved to the account file. Ken’s update does not change the record. In case of recovery, Ken can salvage his changes and can add these to the account file later.

Calling the account file by the CICS program NACT02 means calling the appropriate file entry ACCTFIL in the CICS FCT. The FCT forwards the request to the account file to execute it. This method keeps CICS programs
independent of the access to the data files. If the locking file TBUSSE.CICSADP.ACTINUSE is called in case of an update, delete, or create process, NACT02 calls the FCT entry ACINUSE.

The browse program NACT05 calls entry ACCTNAM in the CICS FCT to make a request on the VSAM file TBUSSE.CICSADP.ACCTNAME — the name file. That file is a path via an alternate index (TBUSSE.CICSADP.ACCTNAIX) to the account file. An alternate index (AIX) is a CICS facility that sorts the account file, for instance, by the customer’s surname. The account number can also be queried through the index. Hence, browsing a file is possible through an AIX without any extra intervention by an application. The AIX creates a new account file, in that case the name file.

4.4.2 Installing the storing data

The JCL script called VSAM, uploaded into the data set TBUSSE.CICSADP.JCLLIB, defines the account, locking, and name file used for the CICS business application (for a complete listing see Listing 20, page 228). In contrast to the CICS COBOL programs and the map set this script has to be compiled on OS/390 during the install procedure.

The script starts with an absolute necessary jobcard, where ACCTJCL1 is the name of the job indicated by the parameter JOB (Listing 5). For instance, this name refers to a log created during execution of the script. The parameter is followed by parameters set for the script, for example, who is to be informed in case of a successful or failed compilation (NOTIFY). If the jobcard is globally set, it can be omitted. The program IDCAMS creates VSAM data sets and stores the data set information automatically into the default VSAM catalogue OS390.MASTER.CATALOG. IDCAMS manages all of the housekeeping needs of VSAM data sets. The print out for the procedure protocol is set in line 19, in that case the it is the user system output, accessible from the System Display and Search Facility (SDSF) program. The SYSIN statement introduces the job control data. Listing 5 shows the commands and attributes used:

```
01 //ACCTJCL1 JOB (),CLASS=A,MSGCLASS=H,MSGLEVEL=(1,1),NOTIFY=&SYSUID,
02 //          TIME=1440
... 18 //SETUP EXEC PGM=IDCAMS,REGION=2M
19 //SYSPRINT DD SYSOUT=* 20 //SYSIN DD *
```

Listing 5: Extract from the member VSAM stored in TBUSSE.CICSADP.JCLLIB (part 1)

The account file TBUSSE.CICSADP.ACCTFILE and the locking file TBUSSE.CICSADP.ACTINUSE are created using the AMS DEFINE command (Listing 6, page 72). If these files are previously defined they are firstly deleted (line 24 and 25) before defining them new. Actually, the account file, which is a KSDS, is organised as a VSAM cluster consisting of a data component and an index component (lines 30 – 41). The AMS command DEFINE CLUSTER creates and the parameter NAME names the VSAM cluster which must be unique in the
VSAM catalogue. KEYS is only used in a KSDS and sets the length (five-digit account number) and position of the key fields in the VSAM catalogue (counting begins at position 0). The length of the key corresponds to the account numbers specified in the input data file TBUSSE.CICSADP.VSAMDATA, which fills up the account file. Since the keys are numeric, only 99,999 customer accounts can be created. In case of future expansion, alphanumeric keys may be used “and with just one letter in the (record) key can grow to 359,964.” ([HOR00], p.42) INDEXED indicates the key-sequenced data, the first number of RECORDSIZE specifies the average length (383) and the second the maximal length (383) of the data records, in that case the length of one customer entry in the input data file:

<table>
<thead>
<tr>
<th>Account number (5 records)</th>
<th>Date issued (3*2 records)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Surname (18)</td>
<td>Reason issued (1)</td>
</tr>
<tr>
<td>First name (12)</td>
<td>Card code (1)</td>
</tr>
<tr>
<td>Middle initial (1)</td>
<td>Approver – initials (3)</td>
</tr>
<tr>
<td>Title (4)</td>
<td>Special codes (3*1)</td>
</tr>
<tr>
<td>Phone number (10)</td>
<td>Account status (2*1)</td>
</tr>
<tr>
<td>Address line (24*3)</td>
<td>Charge limit (8)</td>
</tr>
<tr>
<td>Other charge name (32*4)</td>
<td>Payment history: Balance (8), Bill amount (8), Date paid (6), Amount paid (8)</td>
</tr>
<tr>
<td>Cards issued (1)</td>
<td></td>
</tr>
</tbody>
</table>

Total: 383 records

Indeed, the account numbers and the customer entries are fixed, a variable record length is a better solution and can be simple added in future. The parameter RECORDS (shortcut REC) sets “the amount of space to allocate for the cluster.” ([UDS00], ch. 2.2.2.3) 80 records (1 track) are allocated as primary space, a secondary space is not set. Sharing the VSAM data set is defined with the option SHAREOPTION (shortcut SHR). The first number refers to the Cross-Region Sharing (CRS) and the second refers to the Cross-System Sharing (CSS). Because Record Level Sharing (RLS) is not activated between the CICS region and the OS/390 region on the JEDI-server the first number of the CRS-parameter is set to 2. RLS is used to share VSAM data for many applications between many CICS regions in an MVS parallel sysplex. With this option, any non-RLS user is allowed to access the data set for read processing. Additionally, it can also be accessed by one non-RLS user for write processing. “VSAM ensures write integrity by obtaining exclusive control for a control interval when it is to be updated.” ([UDS00], ch. 2.7.2.1) The CSS-SHAREOPTION is set to number 3 – the data set can be fully shared. This option is only important if the CRS-SHAREOPTION fits number 3 or 4, which means, “each user is responsible for read and write integrity for the data.” ([UDS00], ch. 2.7.2.1) For more information about sharing VSAM data sets refer to “Chapter 13 – Sharing VSAM Data Sets” in [UDS00]. The data set recovery option is set within the parameter LOG to UNDO for backout only. This parameter applies only when the file is accessed in RLS mode, otherwise VSAM ignores this attribute. The VOLUMES parameter allocates space for the file cluster on the volume serial number SMS002. The DATA and INDEX parameters are only specified when subparameters are explicitly required for the data and index component. Within the NAME parameter are defined separately names for the components. Both components are created on the specified volume (SMS002) with the parameter UNIQUE. If UNIQUE is specified, free space must exist on the volume SMS002.
Generation of a Java front end for a standalone CICS application accessed through MQSeries &
Securing CICS with RACF

The locking file locks currently accessed records of the account file to prevent concurrent updates on them. Therefore, the properties in the locking file cluster are copied from the account file cluster (lines 45 – 56). Only the RECORDSIZE parameter is changed to a smaller amount (25) because only six data fields are noticed in the locking file:

- Account number (5 records)
- Owner: user name (8) + terminal identifier (4)
- Date (4)
- Time (4)
- Total (25)

The account file is replenished with the source data set TBUSSE.CICSADP.VSAMDATA using the REPRO command (Listing 6, lines 62 – 64). All records in the data file must be in ascending order by non-duplicated account numbers as keys. INDATASET (shortcut IDS) specifies the source data set, and OUTDATASET (shortcut ODS) the output data set – the account file. With the AMS command DEFINE AIX the alternate index is created (Listing 7, lines 71 – 82). It is named by the NAME parameter as TBUSSE.CICSADP.ACCTNAIX and relates to the account file within the RELATE parameter. A space of 80 records for the AIX is also allocated; SHARE-OPTION and VOLUMES are set as same as in the file cluster definitions. However, the alternate index uses other
The CICS Business Application NACT

The first attribute is set to 18 because the alternate index is built on the customer's surname, which has maximal 18 characters. Since the surname begins on the sixth position in the account file, the second attribute is set to 5 (the five-digit account number starts at position 0). NONUNIQUEKEY (shortcut NUNQK) is set to specify that unique keys are not needed because same surnames could exist in the customer database. With the UPGRADE (shortcut UPG) parameter VSAM updates the alternate index whenever there is a change to the associated account file. The alternate index should not be called by the CICS application itself but by the name file TBUSSE.CICSADP.ACCTNAME, that has still to be created. This file is defined as a path using the AMS command DEFINE PATH. It is filled up with the AIX specified within the PATHENTRY (shortcut PENT) parameter. Hence, when the alternate index is updated, the name file is also updated (UPDATE parameter). At the end, the AMS command BLDINDEX (shortcut BIX) is used to build the internal keys for the alternate index from the account file specified on the IDS parameter (shortcut for IDSDATASET). The keys are sorted within the parameter INTERNALSORT.

After the JCL script is successfully submitted, the file clusters (account & locking file), the AIX and the path (name file) are created on OS/390.

```sql
71    DEFINE AIX(NAME(TBUSSE.CICSADP.ACCTNAIX) -
72                              RELATE(TBUSSE.CICSADP.ACCTFILE) -
73                              VOLUMES(SMS002) -
74                              KEYS(18 5) -
75                              NONUNIQUEKEY -
76                              UPGRADE -
77                              REC(80) -
78                              SHR(2 3)) -
79                              DATA(NAME(TBUSSE.CICSADP.ACCTNAIX.DATA) -
80                              UNIQUE) -
81                              INDEX(NAME(TBUSSE.CICSADP.ACCTNAIX.INDEX) -
82                              UNIQUE)
...  
86    DEFINE PATH(NAME(TBUSSE.CICSADP.ACCTNAME) -
87                              PATHENTRY(TBUSSE.CICSADP.ACCTNAIX) -
88                              UPDATE)
...  
95    BLDINDEX IDS(TBUSSE.CICSADP.ACCTFILE) -
96    ODS(TBUSSE.CICSADP.ACCTNAIX) -
97    INTERNALSORT
```

Listing 7: Extract from the member VSAM stored in TBUSSE.CICSADP.JCLLIB (part 3)
4.5 The CICS resource definitions

4.5.1 Overview

Since the employees of the bank should access the business application through a CICS terminal there has to be defined some CICS resources. “(CICS resource) Definitions are stored on the CICS system definition (CSD) file, and are installed into an active CICS system from the CSD file.” ([RDG03], ch. 1.1) The CSD can be created on-line using the CEDA/B/C transaction or off-line within the CSD update utility DFHCSDUP. Besides, there exist three more methods to define resource definitions – within automatic installation, within system programming using EXEC CICS CREATE commands, and within macro definition. The resource definitions for the sample CICS application are created off-line with the help of a JCL surrounded script calling the DFHCSDUP program. For details on how to create resource definitions using these five methods see [RDG03].

During the upload process two CSD update scripts – CICS0ADP and CICSIJADP – are transferred into the data set TBUSSE.CICSADP.CSDEFS (refer to Table 3, page 42). The script CICS0ADP is renamed to CICSCSD. The other script CICSIJADP is not required; it defines CICS resources for the JAVA interface.

4.5.2 Setting up the CICS resources

There are defined 11 CICS resource definitions – three file objects, one map set object, five program objects, and two transaction objects. All these definitions have been installed into the CICS group CICS0ADP. The script begins with a definition of the file objects (cf. Listing 21, page 228). For example, the account file TBUSSE.CICSADP.ACCTFILE stored on OS/390 is accessed by the following CICS resource definition:

```
07 DEFINE FILE(ACCTFIL) GROUP(CICS0ADP)
08 DESCRIPTION(MAIN ACCOUNT FILE)
09 DSNNAME(TBUSSE.CICSADP.ACCTFILE) RLSACCESS(NO) LSRPOOLID(1)
10 READINT(UNCOMMITTED) DSNSHARING(ALLREQS) STRINGS(1)
11 RECORDSIZE(383) KEYLENGTH(5) STATUS(ENABLED) OPENTIME(FIRSTREF)
12 DISPOSITION(SHARE) DATABUFFERS(2) INDEXBUFFERS(1) TABLE(NO)
13 MAXNUMRECS(NOLIMIT) UPDTEMODEL(LOCKING) LOAD(NO)
14 RECORDFORMAT(F) ADD(YES) BROWSE(NO) DELETE(YES) READ(YES)
15 UPDATE(YES) JOURNAL(NO) JNLREAD(NONE) JNLSYNCREAD(NO)
16 JNLUPDATE(NO) JNLADD(NONE) JNLSYNCWRITE(YES) RECOVERY(NONE)
17 FWDRECOVLOG(NO) BACKUPTYPE(STATIC)
```

When the CICS account file object ACCTFIL is created using the DEFINE FILE command the CICS group CICS0ADP specified on the parameter GROUP is immediately created. In this group the CICS file object is stored. Some important parameters are described in detail, for a full description of all parameters used see Part 5 of [RDG03].

---

7 CEDA is the abbreviation of CICS Execute-level Dynamic Add; CEDB and CEDC consist of a few transactions CEDA contains. They got the endings B and C solely CEDA ends with an A.
The parameter `DSNAME` refers to the associated account file on OS/390. Since RLS is not supported on the JEDI OS/390-server, the option `RLSACCESS` has to be set to `NO`. Since the VSAM file is not accessed through RLS it has to be accessed using a local shared pool (LSR) or not. An LSR having number 1 is set within the `LSRPOOLID` parameter. On `RECORDSIZE` has to be specified the same length as on the eponymous AMS parameter specified for the VSAM KSDS account file cluster (383). `KEYLENGTH(5)` sets the length of the characters used for the account number. The amount of concurrent updates allowed on the account file is set with the `STRINGS` parameter, in that case one user can update the account file; concurrent updates are permitted. Buffers to be used for the data component are set within the `DATABUFFERS` parameter. The minimum value that must be specified is one more than the number specified in `STRINGS`. Additionally, buffers also has to be set for the index component of the account file within the `INDEXBUFFERS` parameter. The minimum number to be specified for this parameter is the same as set in `STRINGS`. The `STRINGS`, `DATABUFFERS`, and `INDEXBUFFERS` parameters are required because RLS mode access is not activated. These parameters are set globally, but usually, it is recommended to define these parameters in an LSR resource definition that corresponds to the `LSRPOOLID`. The attribute `ENABLED` on the `STATUS` parameter specifies that the CICS file object may be used. Because the records in the account file have a fixed length the `RECORDFORMAT` parameter is set to `F`.

The CICS locking file object ACINUSE has few parameters different from the parameters defined in the CICS account file object. The format of the records is set to variable length within the `RECORDFORMAT` parameter. Hence, only the required characters up to 25 are to be stored. The records may not be longer as specified in the VSAM file definition for the locking file `TBUSSE.CICSADP.ACTINUSE`.

For the CICS name file object ACCTNAM are specified the same parameters as for the CICS account file object except the missing parameters `RECORDSIZE` and `KEYLENGTH`. The CICS name file object accesses the VSAM name file for browsing on the surnames but this file is only a path to the alternate index file which gets the records from the account file. Therefore, record size and key length are not needed for the CICS name file object.

Five important parameters are not mentioned yet – `ADD`, `BROWSE`, `DELETE`, `READ`, and `UPDATE`. The CICS file objects declare the restrictions on how the VSAM file on OS/390 can be accessed by the CICS COBOL programs as shown in Table 6.

<table>
<thead>
<tr>
<th>CICS file object</th>
<th>CICS Resource definition parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ADD</td>
</tr>
<tr>
<td>ACCTFIL</td>
<td>YES</td>
</tr>
<tr>
<td>ACCTNAM</td>
<td>NO</td>
</tr>
<tr>
<td>ACINUSE</td>
<td>YES</td>
</tr>
</tbody>
</table>

*Table 6: Allowed operations on the CICS file objects*
The map set \textit{NACTSET} stored on OS/390 has also to be linked to CICS to display the screens of the application. This CICS map set object for the CSD, also named as \textit{NACTSET}, is created and stored in the CICS group CICS0ADP using the \texttt{DEFINE MAPSET} command (Listing 21):

\begin{verbatim}
40 DEFINE MAPSET(NACTSET) GROUP(CICS0ADP)
41        DESCRIPTION(DISPLAYS THE MENUS)
42        RESIDENT(NO) USAGE(NORMAL) USELPACOPY(NO) STATUS(ENABLED)
\end{verbatim}

For a detailed description of the parameters see Part 5 of [RDG03].

The five CICS program objects are created as the example for \textit{NACT01} shows:

\begin{verbatim}
43 DEFINE PROGRAM(NACT01) GROUP(CICS0ADP)
44        DESCRIPTION(PRESENTATION LOGIC FOR THE TERMINAL)
45        LANGUAGE(COBOL) RELOAD(NO) RESIDENT(NO) USAGE(NORMAL)
46        USELPACOPY(NO) STATUS(ENABLED) CEDF(YES) DATALOCATION(BELOW)
47        EXECKEY(USER) CONCURRENCY(QUASI-RENT) DYNAMIC(NO)
48        EXECUTIONSET(FULLAPI) JVM(NO)
\end{verbatim}

The same parameters are set for the other four CICS program objects, of course, the name of the program has to be changed for each program and \texttt{DESCRIPTION} can also be changed to a proper description of the program.

The important parameter of this resource definition is \texttt{LANGUAGE}. This parameter designates the programming language in which the program is written, in that case \texttt{COBOL}. Other languages that could be specified are \texttt{ASSEMBLER}, \texttt{LE370}, \texttt{C}, and \texttt{PL/I} resp. \texttt{PL/1}. A detailed description of all other parameters specified could also be found in Part 5 of [RDG03].

As last objects, the script defines the CICS transaction objects \textit{NACT} and \textit{NACP}. \textit{NACT} is the transaction ID that the user needs to run the application on the CICS terminal, \textit{NACP} is the transaction ID to print the display. \textit{NACP} is defined but not used yet. The \textit{NACT} transaction is defined by following parameters:

\begin{verbatim}
82 DEFINE TRANSACTION(NACT) GROUP(CICS0ADP)
83        DESCRIPTION(NACT TRANSACTION)
84        PROGRAM(NACT01) TWASIZE(0) PROFILE(AAACICST) STATUS(ENABLED)
85        TASKDATALOC(BELOW) TASKDATAKEY(USER) STORAGECLEAR(NO)
86        RUNAWAY(SYSTEM) SHUTDOWN(DISABLED) ISOLATE(YES) DYNAMIC(NO)
87        ROUTABLE(NO) PRIORITY(1) TRANCLASS(DFHTCL00) DTIMOUT(NO)
88        RESTART(NO) SPURGE(NO) TPURGE(NO) DUMP(YES) TRACE(YES)
89        CONFDATA(NO) ACTION(BACKOUT) WAIT(YES) WAITTIME(0,0,0)
90        RESSEC(NO) CMDSEC(NO)
\end{verbatim}

The transaction \textit{NACT} is defined within the command \texttt{DEFINE TRANSACTION}. Some important parameters are described, for all other parameter see Part 5 of [RDG03]. The parameter \texttt{PROGRAM} specifies the program which is executed when the transaction \textit{NACT} is issued on the terminal. \texttt{PROFILE} sets the name of the profile definition (\texttt{AAACICST}) that specifies the processing options used in conjunction with the terminal that initiated the transaction. This profile has been used to convert all entered letters into capitals. How to define such a profile is described in chapter 6.6 “Adjust the LOGIN terminal to pass capital letters to RACF” on page 207. The param-
er **SHUTDOWN** specifies whether a transaction can be used during a CICS shutdown, furthermore. The transaction itself has no own security options defined (**RESSEC(NO)** and **CMDSEC(NO)**) because the application should be accessible for all users in any way.

These CICS resource definitions are installed to the CSD file by using the **DFHCSDU** job. There has to be inserted some JCL code around the definitions before the script can be executed:

```plaintext
01 //CSDADP   EXEC PGM=DFHCSDU,REGION=0M,
02 //         PARM='CSD(READWRITE),PAGESIZE(60),NOCOMPAT'
03 //STEPLIB   DD DSN=CICSTS13.CICS.SDFHLOAD,DISP=SHR
04 //DFHCSD    DD UNIT=SYSDA,DISP=SHR,DSN=CICS.COMMON.DFHCSD
05 //SYSPRINT  DD SYSOUT=A
06 //SYSPRINT  DD *
```

In the first line the item **PGM** refers to the program **DFHCSDU** using the parameters specified in the next line. To update the CSD file the parameter CSD is set to **READWRITE**. The program is stored in the data set **CICSTS13.CICS.SDFHLOAD** specified by the item **STEPLIB**. The item **DFHCSD** links to the data set where the CSDs are stored. When submitting this script, all resource definitions are installed into the CICS group CICS0ADP. Attention, the CSD file must be closed before updating it because the off-line update utility **DFHSCDUP** is used. This utility only performs its operations on a closed CSD! The CSD file is closed either no user is logged on to CICS, or CICS has been shut downed. If it is open, an error is indicated (level 12) and the log notes following message:

```
DFH5128 S PROCESSING TERMINATED. PRIMARY CSD ACCESSED BY ANOTHER USER AND COULD NOT BE SHARED. DDNAME: DFHCSD
DFH5103 I ERROR(S) OCCURRED WHILE PROCESSING DEFINE COMMAND.
DFH5104 W SUBSEQUENT COMMANDS ARE NOT EXECUTED BECAUSE OF ERROR(S) ABOVE.
```

If the script is successfully executed (level 0), the log notes success messages. For example, the DEFINE FILE command that creates the CICS object file ACCTFIL is successfully executed:

```
DFH5120 I PRIMARY CSD OPENED; DDNAME: DFHCSD
DFH5143 I GROUP CICS0ADP CREATED.
DFH5159 I FILE ACCTFIL DEFINED IN GROUP CICS0ADP
DFH5101 I DEFINE COMMAND EXECUTED SUCCESSFULLY.
```

At the end of the log, a brief information is listed:

```
DFH5107 I COMMANDS EXECUTED SUCCESSFULLY: 11 COMMANDS GIVING WARNING(S): 0
DFH5108 I COMMANDS NOT EXECUTED AFTER ERROR(S): 0
DFH5109 I END OF DFHCSDU UTILITY JOB. HIGHEST RETURN CODE WAS: 0
```

As next step, the new CICS group CICS0ADP has to be added to one of the scripts for the CICS System Initialisation Table (SIT) to load the group during the CICS region startup. In the SIT are stored CICS system ini-
tialisation parameters that specify CICS system attributes. For more information about the SIT definition scripts refer to chapter 6.3.1 “The CICS region's SIT” on page 169.

_C001_ is one of those three scripts that sets the CICS transaction server definition. It is stored in the data set CICS.COMMON.SYSIN (cf. Listing 22, page 228). The SIT-parameter _GRPLIST_ set in the script _C001_ contains 4 lists which are loaded during CICS starts up – _DFHLIST_, _C001LIST_, _PPLIST_, and _DEMOLIST_. To add the group CICS0ADP to one of these lists it is chosen to create a new list named DEMOADP. Besides the group CICS0ADP, this new list should also contain all the groups DEMOLIST previously contained. The entry _DEMOLIST_ is then replaced by _DEMOADP_ on the _GRPLIST_ SIT-parameter (cf. Listing 22, page 228):

| 23 GRPLIST=(DFHLIST,C001LIST,PPLIST,DEMOADP), RDO GROUP LISTS |

After that, CICS has to be started to copy the groups of the list DEMOLIST into the new list DEMOADP. The CEDA transaction lists all groups DEMOLIST contains (see Figure 21 on next page):

CEDA DISP LI(DEMOLIST)
The groups of the list DEMOLIST have been copied into the new list DEMOADP with this command:

CEDA APPend LIst(DEMOLIST) To (DEMOADP)
The CICS group CICS0ADP that stores the resources for the CICS business application is also added to the new list:

CEDA ADd Group(CICS0ADP) List(DEMOADP)
At the end, all objects of the group CICS0ADP has to be installed dynamically on CICS to make the resource definitions available to an active CICS system.

CEDA INSTALL GROUP(CICS0ADP)
Before executing the transaction NACT, the CICS startup JCL script _CICSC001_ located in the data set _SYS1.PROCLIB_ must be completed by an entry in the DFHRPL concatenation (cf. Listing 23, page 229):

| 30 // DD DISP=SHR,DSN=TBUSSE.CICSADP.LOADLIB ADP~SAMPLE |

The library _TBUSSE.CICSADP.LOADLIB_ contains the compiled CICS COBOL programs to which CICS connects the CICS program objects ACCTFIL, ACINUSE, and ACCTNAM during startup. The DFHRPL concatenation works like a load path. After these settings are made CICS needs to be restarted to accept the changes of the DFHRPL concatenation (refer to chapter C.1 “Restarting the CICS region”, page 239).

After CICS is restarted the transaction ID NACT can be issued and the CICS business application NACT will start (cf. Figure 16, page 50).
<table>
<thead>
<tr>
<th>NAME</th>
<th>TYPE</th>
<th>LIST</th>
<th>DATE</th>
<th>TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSQSAMP</td>
<td>GROUP</td>
<td>DEMOLIST</td>
<td>01.346</td>
<td>18.04.50</td>
</tr>
<tr>
<td>EBUS</td>
<td>GROUP</td>
<td>DEMOLIST</td>
<td>01.346</td>
<td>18.05.05</td>
</tr>
<tr>
<td>ARTT</td>
<td>GROUP</td>
<td>DEMOLIST</td>
<td>01.346</td>
<td>18.06.11</td>
</tr>
</tbody>
</table>

**Figure 21: CEDA DISP LI(DEMOLIST)**
5 THE MQSERIES CICS BUSINESS APPLICATION

MQNACT

Another Bank Customer Account Program for the JVM

5.1 Introduction

The CICS business application NACT runs on the CICS TS. It consists of a set of programs to gather information about a customer, changes customer details, or deletes customer accounts. For instance, details of a customer account can be displayed on the screen using the CICS transaction NACT on a CICS terminal. But there are many reasons a company could say: 'Is there another method to use the and display the data?'. For one reason, the company decides, that the customer should not use a terminal program to display the details because the connection to the terminal is probably not available through all the time or the GUI is not attractive enough for him. For another reason, the customer wants a billing receipt after an order is made and wants to print this receipt on the client printer at home. Or the printing of the billing receipt requires an additional check on a non-CICS application. For all cases, the company wants to provide an assured and unattended access to the data stored on the OS/390-server using the existing CICS business logic with a high degree of network independence. Furthermore, the bank wants to provide a high availability of the GUI application over a range of operating systems and hardware platforms wherever and whenever the customer wants to access the data. What possibilities the company has?

The company decides to use a middleware system for connecting and transporting data to and from the existing CICS business logic component on the OS/390-server. Either getting the information from programs and transactions running within CICS on one or more OS/390-servers, or getting the information from transaction monitors running on non-mainframe platforms, for example an AIX machine, MQSeries (new: WebSphere MQSeries) – the commercial middleware product introduced by IBM 1992, can be used to solve this problem. MQSeries is part of a generic group called Message Oriented Middleware (MOM). IBM presents it as a “family of products for cross-network-communication” ([AMQ95], S.7), consisting of the MQSeries Server and the MQSeries Client, the MQSeries Integrator, and the MQSeries Workflow. Across a network of unlike components (CPU, operating systems, communication protocols), the user/company gets with MQSeries a simple and consistent GUI-application-to-program/transaction communication. As a result, it exists an indirect program-to-program communication that handles data using messaging and queuing. For example, queries can be queued from one branch to another or within a branch of the bank company.
This thesis uses the existing CICS business logic programs NACT02 and NACT04 of the CICS business application NACT to create a new MQSeries CICS application called MQNACT. On the OS/390-server the IBM MQSeries Server v2.1, as one part of the message queuing product, has been installed. On the Windows2000 client the IBM MQSeries Server v5.2.1, including the MQSeries support pack MA88, has been installed. This support pack provides the JAVA classes for the Message Queue Interface (MQI) which establishes a connection between the JVM and the MQSeries server. Furthermore, a connection called MQSeries CICS Bridge has to be created and started on the OS/390-server that runs the CICS TS. This bridge links the CICS TS with the OS/390 MQSeries Server. The presentation logic and the MQSeries communication logic for the WINDOWS2000 client is programmed in JAVA (MQClient.java and MQCommunicator.java). JAVA makes it possible to display the customer accounts on every system on which a JVM is installed. A programmer of the JAVA application requires no knowledge of the business logic running on the CICS region. There is no need to restructure the CICS business logic programs on the OS/390-server, and other client applications can also use these programs, e.g. the CICS business application NACT. The JAVA Runtime Environment (JRE) version 1.3.1_02 including the SWING package has to be installed on the WINDOWS2000 computer system to run the JAVA application.

8 For downloading this package, please point to: http://www-306.ibm.com/software/integration/support/supportpacs/individual/ma88.html or point to the CD: additions\Windows2000\MQSeries_v5.2.1\MA88
5.2 Messaging and Queuing

5.2.1 Overview

Messaging and Queuing, or Message Queuing (MQ), is one of four program-to-program communication models defined in the IBM Open Blueprint [OBP96] published in 1996. Other alternatives are the Application Programming Interfaces (API) for conversation, remote procedure calls (RPC), and HyperText Transfer Protocols (HTTP). MQ can be used by any application or application-service to obtain appropriate communication support, which may be TCP/IP-oriented, OSI-oriented, or SNA-oriented.

The following definition for MQ is given in [AMQ95], chapter PREFACE:

**Messaging:** “Programs communicate by sending each other data in messages rather than by calling each other directly.”

**Queuing:** “The messages are placed on queues in storage, so that programs can run independently of each other, at different speeds and times, in different locations, and without having a logical connection between them.”

In principle, MQ is not more then putting messages on message queues and taking messages from them. [AMQ95] mentions three main characteristics of MQ:

1. “Communicating programs can run at different times.”

The program-to-program communication is handled by MQSeries. Programs do not communicate directly with each other, they use message queues. When a sender program sends a request to a receiver program, MQSeries creates a message and forwards it to the receiver program. If this program is currently unavailable, MQSeries stores the message until the program is activated. The receiver program executes the request and sends the response back to MQSeries. This data is again stored in a message to be sent to the sender program when it is available. Both programs continue their work without waiting for a response of each other. MQSeries ensures that messages are sent and received without requiring programs to run concurrently. This method is known as asynchronous program-to-program communication. Synchronous communication implies, an application waits for a response to come back before starting the next request.

2. “There are no constraints on application structure.”

Messages can be transferred between two or more applications, also known as one-to-one, many-to-one, or one-to-many connections. Programs may be “on the same processor in a single environment, on the same processor in different environments, or on different processors in different environments.” ([AMQ95], ch. 2.0)
3. “Programs are insulated from network complexities.”

Programs do not communicate directly through a network, they use a middleware that distribute the messages to the right destination. These applications are freed from understanding how the network works and how it is maintained. A network connection is only controlled by MQSeries and not by the programs themselves. Furthermore, program execution is not impacted by a network error or restart.

These three key principles implies four important characteristics:

• time-independent communication
• connectionless communication
• conversational communication
• parallel processing.

MQ distinguishes two different communication models – *Point-to-Point* and *Publish-and-Subscribe*. When applications communicate through the Point-to-Point method, messages are delivered from exactly one sender to one receiver. Other applications have no access to the queue or message. When the Publish-and-Subscribe method is used, a few applications (producer) can produce messages to store them into a message queue. Other applications (consumer) can be registered on specific topics and when a message for such a topic is stored onto the message queue it is immediately transferred to the consumer. Such a message is also called as an *event*; the communication is event-driven.

MQ provides assured “only once” delivery of the message onto the queue. Optionally, it can start the target application to handle the message (called triggering). When no more requests are available for the application it is deactivated until a new trigger message arrives. This optimises resource utilisation.

One characteristic is important for a company. When using MQ, existing business code needs not to be updated or thrown away. The company can use its existing programs for requests and a new one for another request – together without changing the existing one. The new communication code is easy to use and to implement because programs using MQ “have clearly defined inputs and outputs (messages) and a standard interface to other programs that does not vary as the programs themselves are changed and moved.” ([AMQ95], ch. 3.0)

### 5.2.2 Excursion: MQSeries is the UK Post Office

This example describes how MQSeries is represented by the UK Post Office. All Post Office items are tried to paraphrase by MQSeries items.

One of the Post Office main jobs is to deliver letters, cards, and parcels. The method is in MQSeries' point of view *Message Queuing* and the objects are *messages*. The Post Office consists of Post Offices branches that manage the delivery of all postings. Those Post Offices branches are called *queue managers*. 
Imaginary, an invitation letter (request message) is written by a customer (sender program) and should be delivered to his/her friend (receiver program). Such a letter expects an answer. An addressee written on the envelope (message descriptor) and the invitation card (request data) built the invitation letter. An addressee is usually stored in the brain of a human, in MQSeries it is stored in the remote queue. After putting the letter into a postbox (transmission queue) it is transferred by a postman (channel sender) of the home Post Office branch (client queue manager (QM)) to another Post Office branch (server QM). Even if the postbox is outside of a Post Office branch, it is also managed by the branch as same as the postman is an employee of the branch! The postman of the home Post Office branch knows to which branch the letter has to be shipped to. However, from the addressee inscription he gets the name, street, town and postcode. The postcode refers to a unique Post Office branch the letter should be delivered to. After the letter arrived the other Post Office branch, another postman (channel receiver) delivers the letter to the addressee. The letter is put by the postman into the mailbox of the addressee (request queue). The addressee opens the letter and replies to the invitation (receiver program).

The addressee (sender program) writes a confirmation back (reply data) and becomes now the sender. The sender knows the address where to send the letter to (reply message/response message) because it stands on the invitation letter's back. When the letter is complete, consisting of the answer (reply data) and the new addressee (message descriptor), it is again put into a postbox (transmission queue). The postman (channel sender) of the Post Office branch (server QM) delivers the letter to the Post Office branch of the new addressee (client QM). There, the letter is shipped by another postman (channel receiver) to the mailbox of the originator (reply-to queue). At the end, either the request is confirmed by the invited person or not, the whole process is successfully closed. In MQSeries this method is known as the two-way communication.

In the next sections all mentioned MQ catchwords are described in MQ's view.

5.2.3 Messages

MQSeries consists of a number of components to handle message queuing. The most important object is the message itself. It stores application data to be transferred from one MQ-system to another. The message itself is stored in a specific data structure – the message queue. Additionally, this data structure contains a message descriptor (also called message header). It stores control information required for the message, for example the message type (MsgType). Message type datagram is specified, when the sender does not need a reply from the receiver. Request is specified when the sender requires a reply from the receiver. Reply or report can also be specified. Each message can have its own 24-byte string called a message identifier (MsgId) which identifies uniquely a particular message. A message may also contain a 24-byte string called a correlation identifier (CorrelId) that usually matches exactly the message identifier of another message that relates to the current message. For example, it stores the message identifier of the request message to refer to the correct message. Furthermore, it can also set attributes for sending messages in First in, First out (FIFO) or Priority order, for expiring messages after a specified time (expiry), or for assuring the application data in the messages by using the persistence attrib-
ute. If a message is sent using the assured delivery attribute it is stored in a secondary dump until it is successfully delivered to the receiver program. Additional information may be specified in the message descriptor; please refer to [MQI94], ch. 2.2.

Messages can be transferred using a one-way (unidirectional) or a two-way (bidirectional) communication. Unidirectional communication means, sending a message from one program to another using one queue manager without waiting for a response. When using a bidirectional communication, as described in chapter 5.2.2 “Excursion: MQSeries is the UK Post Office” on page 84, two message queues needs to be defined per QM because message queues work always unidirectional. Figure 22 shows the two-way communication that is used for our MQSeries CICS application. When Program A (sender program) want to request a program B it firstly transfers the request data to its QM. The request data is stored in a request message on the specified message queue Queue1 from which it is sent to the program B (receiver program). Program B picks up the request data, executes and answers it, and creates the reply data. The reply data is also stored in a message – the reply message (response message). This is filed on another message queue called Queue2 from which the starting program A receives the reply data. Queue1 and Queue2 are actually generic terms for the transmission queue and the reply-to/request queue.

Figure 22: The two-way communication model for the MQNACT application (image taken from [AMQ95], ch. 2.0)

The main component of the message is the application data. There are no constraints on the content of the data because MQSeries does not interpret this part of the message. Messages can store data up to 100 MB (MQSeries for WINDOWS2000 v5). In case the data is bigger, MQSeries can split the data into a few messages organised by a message group. A message can also be stored on a message queue for a specified time in case many programs want the message data. Hence, each program gets a message copy.

For more information about messages and message descriptors see [AMQ95], [MQI94] and [HKS04].
5.2.4 Queue manager

“A queue manager is the system service that provides the message queuing facilities used by applications.” ([MQI94], ch. 1.3) The queue manager manages all MQ objects, for example message queues, channels, name lists, and process definitions. Other queue manager objects are only required on specialised platforms, such as buffer pools, storage classes, or client connections. As a minimum, one QM with a unique name has to exist on a processor. Applications using MQSeries as middleware communicate with each other only through QMs. For a simple setup, only one QM and one message queue are required (one-way communication). This is realised, when a MQSeries Server is installed on the server computer system, whereas on the client computer system the MQSeries Client as part of the MQSeries family is installed. However, the MQSeries CICS application MQNACT uses the two-way communication which requires at least two QMs. One QM resides on the OS/390-server (server queue manager) and the other on the WINDOWS2000 client (client queue manager) set up within the MQSeries Server product installed on both sides. Using two QMs improves the robustness of the sample business application MQNACT. In case, the OS/390-server is unavailable for any reason, the messages to be sent are kept until the OS/390-server is available again. Referring to as assured delivery, it ensures that the message is delivered once and only once when the broken OS/390-server is available. The QMs in our sample application are named MQA1 for the OS/390-server queue manager and TBUSSE.NACT for the WINDOWS2000 client queue manager. In this scenario, messages are created on a local QM and are transferred to another QM (remote QM) using different message queues. Additionally, there are required transport channels to transfer the messages from one QM to the another.

Queue managers manage the network, look for remote queue managers, and communicate with them. The communication between QMs is handled by TCP/IP. Each QM insures that changes of MQ objects are logged. Messages are built inside a queue manager; an application sends only pure data to it. If messages are too long they are segmented and grouped by the QM. Messages can be send as copies to different QMs. The QM also insures to deliver the messages. For improved administration, QMs can be grouped into QM Clusters (QMC). For more information about QM objects and QMCs see [HKS04] and [QMC00].

5.2.5 Queue manager objects

5.2.5.1 Local and remote queues

Queues are MQ objects that store messages until they are picked up, for instance by applications. They are classified into local and remote queues. All queues are administered by a queue manager. Each queue has an own name unique to each QM. Messages are added onto a queue at the end and removed from the front. This FIFO-method is set per default. In case of priority storing, the messages are stored on their priority level. Each queue is specified with its own attributes. The following queue types are used in the MQSeries CICS application MQNACT:
a) Reply-to queue

This queue is a local queue stored on the local queue manager and is connected with a local program. The reply-to queue stores messages sent by a remote program and provides the received data for the local program. On the OS/390-server this queue is called as request queue, or in connection with its task, it is also called as the MQSeries CICS Bridge queue. This queue receives the request messages. On the WINDOWS2000-client the queue is named as the reply-to queue receiving response messages.

b) Remote queue definition (remote queue)

The remote queue definition is usually simple called remote queue. However, this “queue” works not like a normal queue. It stores only control information for a message to be sent and not the message itself. Therefore, the complete name “remote queue definition” is used in this thesis, however, “definition” is written in italic style. The message consists of this copied information (message descriptor) and the application data. The remote queue definition stores the name of the local queue at the remote location (reply-to resp. request queue), the name of the remote queue manager, and the name of the local transmission queue.

c) Transmission Queue

Before sending messages to a remote queue manager they are transmitted onto a special kind of a local queue called a transmission queue. For our sample-application MQNACT, one transmission queue exists for each QM. Transmission queues are only required when two QMs communicate with each other. Messages are sent to the remote queue manager by a channel sender which is connected with the transmission queue. One default transmission queue can be predefined for the QM. Usually, an own transmission queue is to be created. Furthermore, the transmission queue is connected with the remote queue definition.

The OS/390 MQSeries v2.1 product contains a particular limitation. The OS/390-server queue manager always assigns automatically the transmission queue with the same as the remote queue manager. In other words, the transmission queue on the OS/390-server queue manager must have the same name as the WINDOWS2000-client queue manager. If the transmission queue on the OS/390-server queue manager has another name as the WINDOWS2000-client queue manager, the OS/390-server queue manager assigns the default transmission queue as the transmission queue instead of the specified one. When a message is created, it gets control information from the remote queue definition of the OS/390-server queue manager and is normally put onto the transmission queue specified in the remote queue definition. But the OS/390-server queue manager wants to put the message on a queue named as the WINDOWS2000-client queue manager. If this queue is not found by the OS/390-server queue manager, the message is put onto the default transmission queue. However, the default transmission queue cannot forward the message to the WINDOWS2000-client queue manager because the default transmission queue is linked to another channel sender and not to the channel sender originally specified in the specification of the correct trans-
mission queue. If a default transmission queue is not specified the message is stored on the dead-letter queue.

d) Dead-Letter Queue

A dead-letter queue stores messages, that cannot be delivered. For example, the destination queue is not defined on the remote QM, or the destination queue cannot receive messages. The dead-letter queue must be a local queue. Each QM should have a dead-letter queue. In the Post Office, this dead letter queue is represented by a Dead Letter Office.

There are some more queue types that can be specified, for example an alias queue or model queue. For information see [MQI94] and [HKS04].

5.2.5.2 Message Channels

The QM transfer transfer messages to another QM via channels. There are two channel types – one to connect QMs (message channel) and the other to connect a QM with a MQSeries application (MQI-channel). They are unidirectional and consist always of two parts – the channel sender and channel receiver. IBM calls them also as sender channel resp. receiver channel. However, this notation can be confusing. Therefore, only the notations channel sender and channel receiver are used in this thesis. The channel sender is always located on the local queue manager, the channel receiver always on the remote queue manager. Both channel parts must have the same name9 and thus build together a single channel. The MQSeries CICS application requires a two-way communication between the two queue managers. Hence, two message channels are created, having one channel sender and one channel receiver per message channel (Figure 23, next page).

Both channel parts communicate with each other using a communication protocol such as TCP/IP or LU6.2. Each channel part has a Message Channel Agent (MCA), for the channel sender a sender MCA and for the channel receiver a receiver MCA (Figure 23). This software controls the sending and receiving of messages. A message is taken from the transmission queue by a sender MCA and is put to the communication link. The receiver MCA gets the message and delivers it to the remote QM resp. to the reply-to/request queue. In another view, an MCA that initiates the communication is called a caller MCA, otherwise, it is a responder MCA.

The sender MCA needs to be initiated by a channel initiator program. The receiver MCA is initiated by a channel listener program. This program detects incoming network requests and starts the associated channel receiver. Usually, both programs are started automatically during the startup of the QM. In some situations they can be started manually.

For more information about message channels and Distributed Queue Management see [ICM00].

9 Channel names must not have more than 20 characters!
5.2.5.3 MQI Channels

MQI channels connect MQSeries Clients with a queue manager on an MQSeries server. This one channel sets up a two-way link (bidirectional) to transfer MQI calls and responses. One part of this MQI channel is called the server-connection channel, the other one is called the client-connection channel. The notation “channel” is omitted because both parts build together the channel. A server-connection has been created to connect the JAVA application with the WINDOWS2000-client queue manager. The client-connection is created automatically when a server-connection is used. This feature is a new function added to MQSeries Server v5.2.1. The MQI channel does not transfer messages between the WINDOWS2000-client queue manager and the MQNACT application but pure request resp. response data. Such an MQI channel is only required on the WINDOWS2000-client. On the OS/390-server queue manager it is used another method to transfer the request/response data to the CICS COBOL application. For more details on MQI channels see [CLI00].

5.2.6 Message Queuing Interface

Queue managers communicate with each other using the MQI. It consists of only a few function calls, is easy to use and consistent in every environment. Hence, it is not required to write complex communication code; the programmer can concentrate on programming presentation and business logic.

There are only two important basic calls – MQPUT and MQGET. One for putting messages on a message queue and the other for getting messages from the message queue. This is the “e-mail for applications” part. All in all, 11 calls are provided by the MQI used for MQSeries v2.1, but approximately 90 percent of all calls used in MQSeries applications are MQPUT and MQGET. The MQCONN call, for example, is used to enable a connection between the application and MQSeries, MQDISQ terminates it. MQOPEN opens the message queue to get ready for an MQPUT call. After getting messages for a message queue using MQGET, the MQCLOSE call closes the message queue. These calls are also known as major calls. Minor calls are used for special purpose.
For instance, MQINQ provides information of message queues to manage them. Some queue attributes can be modified by application programs using the MQSET call. If an application encounters problems or errors during executing MQPUT or MQGET calls, it can back out all operations performed on the message queue using the MQBACK call. Operations performed on a queue can be committed by the MQCMIT call. The last call—MQPUT1—simplifies putting messages on a queue. It represents a sequence of MQOPEN, MQPUT, and MQCLOSE calls. In MQSeries v.5 are introduced 2 new API-calls—MQBEGIN and MQCONNX. More information about these 2 calls and the MQI in general provides [MQI94] and [HKS04].

Programmers who does not want to use the functions provided by the MQI can use another simple interface to connect applications with MQSeries—the MQSeries Application Messaging Interface (AMI). This interface “is more abstract in nature than MQI, but provides less flexibility because not all features available through MQI are available in AMI.” ([YOU01], page 170) Still another interface is the JAVA Message Service (JMS). This interface is based on an object-oriented method rather than a procedural interface as the MQI. For more information about the AMI see [AMI00] and about the JMS see [MUJ00].
5.3 The architecture of the MQSeries CICS application

The MQSeries CICS application consists of two program parts – the JAVA presentation and MQSeries communication logic on the WINDOWS2000-client and the CICS business logic on the OS/390-server. The MQSeries communication logic of the WINDOWS2000-client is only used to create a connection to the MQSeries queue manager on the WINDOWS2000-client. The actual business logic is the OS/390 CICS COBOL program NACT02. The JAVA program MQClient.java requests information from a customer account and access the CICS COBOL program NACT02. Both programs communicate through MQSeries; they put messages on and receive messages from message queues. Using MQ is a simple and elegant way to send requests to and to get responses from the CRUD program. The functionality of such a procedure is shown in Figure 24 on page 94 in general and in Figure 25 on page 95 where each object is named. The numbers of the following descriptions refer to both – Figure 24 and Figure 25.

Firstly, the JAVA program connects itself to the WINDOWS2000-client queue manager TBUSSE.NACT using the MQI channel called the server connection channel TBUSSE.NACT.CLIENT (No.1). Within this channel the request data is transported to the remote queue definition TBUSSE.NACT.REMOTEQ (No.2). The remote queue definition provides necessary information to create the request message. This message is immediately put on the transmission queue TBUSSE.NACT.XMITQ (No.3). From this queue the message is transmitted to the OS/390 server queue manager MQA1 using the channel sender TBUSSE.NACT.WIN.OS (No.4a). The channel sender is connected with the eponymous channel receiver specified on the OS/390-server queue manager and builds the channel called request channel (No.4b). When the channel receiver gets the request message, it reads the message descriptor that contains the request queue name SYSTEM.COMMAND.CICS.BRIDGE among other object names, and puts the request message onto this queue (No.5).

<table>
<thead>
<tr>
<th>Queue manager objects</th>
<th>OS/390-Server Queue Manager</th>
<th>WINDOWS2000-Client Queue Manager</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reply-to/Request Queues</td>
<td>SYSTEM.COMMAND.BRIDGE.QUEUE</td>
<td>TBUSSE.NACT.REPLYQ</td>
</tr>
<tr>
<td>Transmission Queues</td>
<td>TBUSSE.NACT</td>
<td>TBUSSE.NACT.XMITQ</td>
</tr>
<tr>
<td>Remote Queue Definitions</td>
<td>TBUSSE.NACT.REPLYQ</td>
<td>TBUSSE.NACT.REMOTEQ</td>
</tr>
<tr>
<td>Channel Senders</td>
<td>TBUSSE.NACT.OS.WIN</td>
<td>TBUSSE.NACT.WIN.OS</td>
</tr>
<tr>
<td>Channel Receivers</td>
<td>TBUSSE.NACT.WIN.OS</td>
<td>TBUSSE.NACT.OS.WIN</td>
</tr>
<tr>
<td>Server Connection</td>
<td>---</td>
<td>TBUSSE.NACT.CLIENT</td>
</tr>
</tbody>
</table>

Table 7: Named queue manager objects used for the MQSeries CICS application MQNACT

The CICS Bridge Monitor Task consecutively checks the request queue if a “start UoW” message has arrived (No.6). After some relevant authorisation checks, the CICS DPL Bridge Task is started (No.7). This task removes the message from the request queue and builds a COMMAREA from the data delivered within the message (No.8). Afterwards, the NACT02 program is called (No.9). The program executes the request and sends the
response back to the COMMAREA (No.10). The response data is carried to the remote queue definition TBUSSE.NACT.REPLYQ stored on the OS/390-server queue manager (No.11). The control information of the remote queue definition is added to the response data and the response message is built and put onto the transmission queue TBUSSE.NACT (No.12). The channel sender TBUSSE.NACT.OS.WIN connected with the transmission queue transfers the response message to the WINDOWS2000-client queue manager (No.13a). The WINDOWS2000 channel receiver (No.13b) builds together with the eponymous OS/390 channel sender the response channel. The channel receiver reads the message descriptor containing the reply-to queue name TBUSSE.NACT.REPLYQ among other object names and puts the response message onto the queue (No.14). After the message is stored on this queue the MQI channel TBUSSE.NACT.CLIENT wakes up and transmits the response data to the JAVA program where the information is displayed (No.1).
Figure 24: The architecture of the MQSeries CICS application MQNACT
Figure 25: The architecture of the MQSeries CICS application MQNACT with named MQSeries objects
5.4 Setting up MQSeries on the OS/390-server

5.4.1 The OS/390-server queue manager MQA1

On the MQSeries server on the JEDI OS/390-server runs only one queue manager called MQA1. It is started when following command is entered in the command line in any panel of the SDSF program:

```
!/MQA1 START QMGR
```

The SDSF program can be reached from the CUSTOMPAC MASTER APPLICATION MENU entering SD in the command line. Each queue manager on OS/390 may consist of maximal four characters because its name is also the name of the MQSeries subsystem which may be only four characters long. Such a MQSeries subsystem is a formal defined OS/390 subsystem and must be defined to the subsystem name table of OS/390. A queue manager can only be started if the assigned MQSeries subsystem is created and loaded previously. For instance, an MQSeries subsystem can be manually created with the SETSSI command:

```
/SETSSI ADD,S=ssid,I=CSQ3INI,P='CSQ3EPX,cpf,scope'
```

where `ssid` is the name of the subsystem ("MQA1"). `cpf` is the command prefix that must preceding an MQSeries command to identify the OS/390 subsystem for which the commands are intended for, for instance "/MQA1". The last parameter `scope` is only important if the subsystem is created in an OS/390 sysplex, if not, 'M' has to be specified. However, the command that creates the MQSeries subsystem MQA1 reads as follows:

```
/SETSSI ADD,S=MQA1,I=CSQ3INI,P='CSQ3EPX,!MQA1,M'
```

It is suggested to include an entry into the IEFSSNxx member of SYS1.PARMLIB. Hence, it is not required to input this command after every IPL. On the JEDI OS/390-server there exit two of such members, IEFSSN00 and IEFSSN02; executed one after another. Following entry is entered in IEFSSN00 (cf. Listing 24, page 234):

```
10 SUBSYS SUBNAME(MQA1) INITRTN(CSQ3INI) INITPARM('CSQ3EPX,!MQA1,M')
```

There are created four more MQSeries subsystems on the JEDI OS/390-server – MQA2, MQA3, MQA4, and MQA5. Of course, the associated entries are also written into the IEFSSN00 member. These subsystems should be used for further practise. Additional information on how to create an MQSeries subsystem and an MQSeries queue manager is given in [SMQ99].

Using the START QMGR command the MQSeries executes the startup script MQA1MSTR stored in the data set SYS1.PROCLIB (Listing 25, page 234). This script is also called the MQSeries started task procedure. The name of the script always consists of the queue manager name, here MQA1, plus the string MSTR. After a subsystem has been created, such a script has to be defined to load and start the queue manager. During the startup of

---

10 Any command entered from the command line of SDSF must be prefixed by a slash "/". If the slash is entered alone the System Command Extension panel appears. The advantage is that more command strings as on the command line can be entered.
the queue manager MQA1 a few queue manager objects have been created using the *CSQINP1* and *CSQINP2* steps (line 42-47). Within these both steps following scripts are loaded:

```plaintext
42 //CSQINP1   DD DSN=MQM.MQA1.SCSQPROC(CSQ4INP1),DISP=SHR
43 //CSQINP2   DD DSN=MQM.MQA1.SCSQPROC(CSQ4INSG),DISP=SHR
44 //          DD DSN=MQM.MQA1.SCSQPROC(CSQ4INSX),DISP=SHR
45 //          DD DSN=MQM.MQA1.SCSQPROC(CSQ4INYG),DISP=SHR
46 //          DD DSN=MQM.MQA1.SCSQPROC(CSQ4CKBM),DISP=SHR
47 //          DD DSN=MQM.MQA1.SCSQPROC(CSQ4STRT),DISP=SHR
```

The script *CSQ4INP1* loaded within the *CSQINP1* step creates non-recoverable objects for the queue manager (Listing 26, page 234). Within the script *CSQ4INSG* some system objects are defined. They must be created the first time the queue manager MQA1 is started (Listing 27, page 234). For the non-CICS distributed queuing and clustering facility some more objects are defined within the script *CSQ4INSX* (Listing 28, page 234). These three scripts are required for each OS/390-server queue manager and may not be changed.

Additional general objects for the queue manager MQA1 have been defined using the script *CSQ4INYG* provided within the MQSeries installation. For instance, this script defines the default transmission queue and the default dead-letter queue. Enabling the communication between CICS and MQSeries requires creating the initiation queue *CICS01.INITQ*. This queue is used by the MQSeries CICS adapter and enables the CICS applications to use the MQI to communicate between MQSeries and CICS. The script *CSQ4CKBM* defines the request queue resp. MQSeries CICS Bridge queue and a trigger for this queue. But, triggering is not used for our sample application MQNACT, however, the trigger definition is created for future use.

The last loaded script *CSQ4STRT* starts the channel initiator and the channel listener programs for the OS/390-server queue manager (cf. Listing 29, page 234). Within the command *START CHINIT* the channel initiator is started. On the *PARM* keyword is specified the data set in which the definitions for the channel initiator are stored (line 11). The command *START LISTENER* starts the channel listener program on the specified port *PORT(1414)*. The TCP/IP method has been chosen as transport type *TRPTYPE(TCP)*, however, port and the transport type can be omitted because these are the default parameters (line 15):

```plaintext
11 START CHINIT PARM(CSQXPARM)
   ...
15 START LISTENER PORT(1414) TRPTYPE(TCP)
```

Both programs can also be started manually using the command line of the SDSF application or the MQSeries panels. For starting both services from the command line execute the following commands, subsequently:

```plaintext
//!MQA1 START CHINIT
//!MQA1 START LISTENER
```
When starting these programs from the MQSeries panels, open the “Main Menu” panel and enter in the field 
*Action* the number 6 as a shortcut for *Start*. Furthermore, fill in the field *Object type* the entry *SYSTEM* and press Enter.

An entry in the field *Name* is not required (Figure 26). Pressing the ENTER key opens the “Start a System Function” panel. Firstly, choose the number 1 to start the channel initiator (Figure 27). When the channel initiator has been started successfully, the listener can be started choosing number 3 in the “Start a System Function” panel. Both services can be stopped in the same way as they were started. The commands to stop them from the command line of the SDSF application read as follows:

```bash
!/MQA1 STOP CHINIT
!/MQA1 STOP LISTENER
```

In case, the channel initiator and the channel listener need to be stopped, it is only required to stop the channel initiator. This stops the listener automatically. When both services should be stopped from the MQSeries panels, enter number 7, the shortcut for *Stop*, instead number 6 in the field *Action* on the MQSeries “Main Menu” panel. This opens the “Stop a System Function” panel (Figure 28). Choose the numbers to stop the service and press Enter.

![IBM MQSeries for OS/390 - Main Menu Panel](image)

*Figure 26: MQSeries for OS/390 on CICS – Main Menu panel*
Start a System Function

Select function type, complete fields, then press Enter to start system function.

Function type . . . . . . . 1  
1. Channel initiator
2. Channel listener for LU6.2
3. Channel listener for TCP/IP

Channel initiator
Parameter module name . . ________
JCL substitution .... ____________________________

Listener for LU6.2
LU name ............ ______________

Listener for TCP/IP
Port number ........ 1414

Command ===> __________________________________________________________________
F1=Help      F2=Split     F3=Exit      F9=Swap     F10=Messages F12=Cancel

Figure 27: MQSeries for OS/390 on CICS – Start a System Function panel

Stop a System Function

Select function type, then press Enter to stop system function.

Function type . . . . . . . 1  
1. Channel initiator
2. Channel listener for LU6.2
3. Channel listener for TCP/IP

Command ===> __________________________________________________________________
F1=Help      F2=Split     F3=Exit      F9=Swap     F10=Messages F12=Cancel

Figure 28: MQSeries for OS/390 on CICS – Stop a System Function panel
If it is needed to stop the queue manager MQA1, following command has to be executed from the command line of the SDSF application:

```
!/MQA1 STOP QMGR MODE(option)
```

The attributes for `option` in the optional keyword `MODE` can be one of the following: `QUIESCE`, `FORCE`, and `RESTART`. `QUIESCE` allows programs currently being executed to finish processing. This is the default option; if the keyword `MODE` with its parameter is omitted, this option is always used to shut down the queue manager.

The `FORCE` option terminates the queue manager without ending the programs in their normal way. Both options deregister MQSeries from the MVS automatic restart manager. The option `RESTART` allows an automatic restart. All three options exclude starting of new programs during terminating the queue manager. If the queue manager is terminated it can be restarted because the OS/390 subsystem MQA1 is not terminated. It waits for an reactivation call of the queue manager. Following message is written to the system log to indicate the complete termination:

```
\HASP395 MQA1MSTR ENDED
CSQ3104I !MQA1 CSQ3EC0X - TERMINATION COMPLETE
CSQ3100I !MQA1 CSQ3EC0X - SUBSYSTEM MQA1 READY FOR START COMMAND
```

### 5.4.2 The MQSeries CICS Bridge

#### 5.4.2.1 Overview

The Java-GUI front end on the WINDOWS2000-client wants to access the CICS COBOL program `NACT02` with the help of MQSeries. `NACT02` should only be called by the MQSeries CICS application and may not be modified because an application programmer should not need to know about the mainframe programming. The programmer has only to know the name of programs and of the MQ objects which process the request. Therefore, it is required to use a method that connects CICS with MQSeries to work in the background. For this situation, IBM offers to use the MQSeries CICS Bridge. It solves the problem of transferring data between the interfaces of MQSeries and CICS using the MQI. Hence, any application outside of CICS can run a single program or a set of programs (UoW) on CICS. Additionally, the MQSeries CICS Bridge queue supports the synchronous and the asynchronous processing of data.

`NACT02` is executed by the MQSeries CICS application when a request within a message is sent to the CICS Bridge Monitor Task of the MQSeries CICS Bridge. Firstly the message has to be put on the request queue – the MQSeries CICS Bridge queue – which is constantly browsed by the CICS Bridge Monitor Task. When the task recognises that a message has arrived, it checks the authentication, and it starts the CICS DPL Bridge Task executing the `EXEC CICS START` command. It collects the request data from the MQSeries CICS Bridge queue and deletes the message from the queue. From the data is build the COMMAREA and an `EXEC CICS LINK` command is issued to call `NACT02`. The CICS COBOL program processes the data and returns the response back
into the COMMAREA. From this area, the CICS DPL Bridge Task reads the response data using the `EXEC CICS RETURN` command and transmits it to the remote queue definition of the OS/390-server.

On the one hand, the MQSeries CICS Bridge is an application consisting of the CICS Bridge Monitor Task started by the CICS transaction CKBR and consisting of the CICS DPL Bridge Task started by the transaction CKBP. But on the other hand, it is defined as a local queue on the OS/390-server queue manager MQA1. Until these both tasks have been installed to the CICS TS and the local queue is created to MQSeries, the bridge can be successfully started, and data can be transferred between the OS/390-server queue manager and the CICS region. Every CICS region having an installed CICS Bridge Monitor Task and a CICS DPL Bridge Task can run a MQSeries CICS Bridge, if an assigned local queue exists on a OS/390-server queue manager – same or different queue managers. However, using the MQSeries CICS Bridge requires, that both, the OS/390-server queue manager and the CICS region, run in the same OS/390 region.

5.4.2.2 Configuring CICS to use the MQSeries CICS Bridge

The MQSeries CICS adapter must be started at first before the MQSeries CICS Bridge tasks can run on CICS. The adapter provides control functions to manage the connection between CICS and MQSeries, for example, starting and stopping the connection to the OS/390-server queue manager. As second functionality, the adapter implements the MQI support used by CICS programs. For example, it “can handle up to eight MQI calls concurrently.” ([SMQ99], ch. 3.1.1.2)

The MQSeries CICS adapter is started by the CKTI transaction during CICS startup. This transaction and its assigned programs have been defined within the script `CSQ4B100` stored in the library `CICS.COMMON.CSDDEFS`. This script also defines all other objects required by the MQSeries CICS adapter to the CICS region A06C001 (cf. Listing 30, page 235). These objects have been installed into an own CICS group called CSQ (line 232). A complete specification of `CSQ4B100` is omitted because the script is provided within the CICS TS installation and needs only to be executed. However, at the end of the script one command has been inserted. It adds the CICS group CSQ to one of the four CICS startup lists. These lists works like a search path to find the necessary objects to start the MQSeries CICS adapter during CICS startup. The CICS group list PPLIST has been chosen (line 242):

```
232 DEFINE TRANSACTION(CKTI) GROUP(CSQ)
...  
242 ADD GROUP(CSQ) LIST(PPLIST)
```
The objects specified in `CSQ4B100` have been added to the CICS region using the Update Resource Definition Utility `DFHCSDUP`. The script `DFHCSD01` stored in `CICS.COMMON.JCL` executes the utility and loads as input the script `CSQ4B100` (line 8, and 14, cf. Listing 31, page 235):

```
08 //CSDUP   EXEC PGM=DFHCSDUP,REGION=4M
14 //SYSIN   DD DISP=SHR,DSN=CICS.COMMON.CSDDEFS(CSQ4B100)
```

In order to use the MQSeries CICS adapter, an initiation queue has to be created on the OS/390-server queue manager. This queue is automatically created during each queue manager startup executing the script `CSQ4INYG` (cf. Listing 32, page 235). The initiation queue is named as `CICS01.INITQ`:

```
223 DEFINE QLOCAL( 'CICS01.INITQ' ) REPLACE +
```

Additionally, it is required to add the parameter `INITPARAM` to the CICS SIT definition script `C001` to declare the initiation queue to the CICS region (line 5, Listing 22):

```
05 INITPARAM=(CSQCPARM='SN=MQA1,TN=001,IQ=CICS01.INITQ')
```

where the parameters mean:

- **SN** – The MQSeries queue manager name to which the MQSeries CICS adapter should be connected to,
- **TN** – The trace number to identify the MQSeries CICS adapter in the CICS trace entries,
- **IQ** – The name of the initiation queue on the queue manager provided for the MQSeries CICS adapter.

**Note:** Attributes of `INITPARAM` must not be more than 60 characters long; counted from `'SN=...'` until `'...INITQ'`.

Furthermore, there are added some required MQSeries libraries to the `STEPLIB` and to the `DFHRPL` concatenation in the CICS startup procedure `CICSC001` (lines 13-38, Listing 23):

```
13 //STEPLIB DD DISP=SHR,DSN=&CICSHLQ..SDFHAUTH        CICS
      DD DISP=SHR,DSN=MQM.SCSQANLE                     MQSERIES
      DD DISP=SHR,DSN=MQM.SCSQAUTH                      MQSERIES
      DD DISP=SHR,DSN=MQM.SCSQLOAD                      MQSERIES
```

```
17 // DD DISP=SHR,DSN=CICS.COMMON.TABLIB               CICS TABLES
```

```
18 // DD DISP=SHR,DSN=MQM.SCSQANLE                     MQSERIES
```

```
25 //DFHRPL DD DISP=SHR,DSN=CICS.COMMON.TABLIB         CICS TABLES
```

```
35 // DD DISP=SHR,DSN=MQM.SCSQANLE                     MQSERIES
```

```
36 // DD DISP=SHR,DSN=MQM.SCSQCICS                     MQSERIES
```

``` 
37 // DD DISP=SHR,DSN=MQM.SCSQAUTH                     MQSERIES
```

```
38 // DD DISP=SHR,DSN=MQM.SCSQLOAD                     MQSERIES
```
At last, the SIT-parameter `MQCONN` in the CICS SIT definition script `C001` has to be activated to auto-connect MQSeries and CICS (line 5, cf. Listing 33, page 235):

```
05 MQCONN=YES AUTO-CONNECT TO MQ
```

Hence, the MQSeries CICS adapter is started and the connection between the CICS region and the OS/390-server queue manager is established. That can be checked when the CICS transaction CKQC is started on the CICS terminal. After the initial panel appears, point the cursor under “Connection”, press the enter key, and choose option number 4 to display the connection status (Figure 29). The “Display Connection” panel shows the name of the CICS region, the name of the queue manager connected to, and that a connection persists between both (Figure 30, next page).

After the successful installation and running of the MQSeries CICS adapter it is possible to set up the MQSeries CICS Bridge on the CICS region. The MQSeries CICS Bridge usually consists of three tasks. However, running CICS DPL programs requires only two of these – the CICS Bridge Monitor Task and the CICS DPL Bridge Task. The CICS Bridge Monitor Task consists of the transaction CKBR calling the program `CSQCBR00` whereas the CICS DPL Bridge Task consists of the transaction CKBP calling the program `CSQCBP00` and the abend handler program `CSQCBP10`. The error messages program `CSQBTX` relates to both tasks. The third task – the MQSeries CICS Bridge Exit Task – consists of the 3270 CICS Bridge Exit and of the Data Conversion Exit. The 3270 CICS Bridge Exit program `CSQCBE00` is only used if 3270 transactions should be run within MQSeries. `CSQCBDCI`, the program for the Data Conversion Exit, drives conversion of 3270 map data.

![Figure 29: MQSeries for OS/390 on CICS – Display Connection panel](image-url)
CKQCM2 Display Connection panel

Read connection information. Then press F12 to cancel.

CICS Applid = A06C001 Connection Status = Connected QMgr name= MQA1
Trace No. = 001 Tracing = On API Exit = Off
Initiation Queue Name = CICS01.INITQ

---------------------------- S T A T I S T I C S -----------------------------
Number of in-flight tasks = 2 Total No. of API calls = 516505
Number of running CKTI = 1

APIs and flows analysis

<table>
<thead>
<tr>
<th>Run OK</th>
<th>MQINQ</th>
<th>3</th>
<th>Tasks</th>
<th>181</th>
<th>Indoubt</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQOPEN</td>
<td>184</td>
<td>Flows</td>
<td>517047</td>
<td>S-Phase</td>
<td>180</td>
<td>Commit</td>
</tr>
<tr>
<td>MQCLOSE</td>
<td>0</td>
<td>Calls</td>
<td>515958</td>
<td>2-Phase</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>MQGET</td>
<td>516138</td>
<td>SyncComp</td>
<td>1268</td>
<td>Commit</td>
<td>180</td>
<td>Backout</td>
</tr>
<tr>
<td>MQPUT</td>
<td>180</td>
<td>Msgwait</td>
<td>517779</td>
<td>InitTCBs</td>
<td>8</td>
<td>StrtTCBs</td>
</tr>
<tr>
<td>MQPUT1</td>
<td>180</td>
<td>Switched</td>
<td>517046</td>
<td>BusyTCBs</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

F1=Help F12=Cancel Enter=Refresh

Figure 30: MQSeries for OS/390 on CICS – Display Connection panel

All these three tasks have been defined to the CSD using the script CSQ4CKBC stored in the library CICS.COMMON.CSDDEFS (cf. Listing 35, page 235). When the off-line update resource definition utility DFHCSDUP is executed, all MQSeries CICS Bridge objects are added to the CICS region. As known, this utility program is loaded within the script DFHCSD01 (cf. Listing 31, page 235). But now, its SYSIN step refers to CSQ4CKBC (line 14, Listing 31):

14 //SYSIN DD DISP=SHR,DSN=CICS.COMMON.CSDDEFS(CSQ4CKBC)

All MQSeries CICS Bridge objects have been installed into the CICS group CSQCKB as same as the transaction CKBP (line 24, Listing 35):

24 DEFINE TRANSACTION(CKBP) GROUP(CSQCKB)

After all objects have been defined, the CICS group CSQCKB has also to be added to any of the four CICS group startup lists; it is again chosen to use PPLIST (line 84, Listing 35):

84 ADD GROUP(CSQCKB) LIST(PPLIST)
5.4.2.3 Configuring MQSeries to use the MQSeries CICS Bridge

The MQSeries CICS Bridge queue must be defined as a local queue on the OS/390-server queue manager MQA1. Additionally, this queue must be connected to the CICS Bridge tasks to enable the MQSeries CICS Bridge. Provided within the dataset MQM.MQA1.SCSQPROC the sample script CSQ4CKBM is used to create the MQSeries CICS Bridge queue (cf. Listing 34, page 235). Within the MQSeries command (MQSC) **DEFINE QLOCAL** the queue with its name has been defined – ‘SYSTEM.CICS.BRIDGE.QUEUE’ (line 32). The attributes set in line 36 to 39 are common queue attributes for a local queue. The queue has the lowest priority (**DEFPRTY(0)**, line 36) and can process persistent messages (**DEFPSIST(YES)**, line 37). To recover messages in case of errors, they are set persistently on the queue. Furthermore, to put messages sent from the CICS DPL Bridge Task onto the queue the keyword **PUT** is set to ‘**ENABLED**’ (line 39). Because the CICS Bridge Monitor Task gets messages from the MQSeries CICS Bridge queue, **GET** has also to be enabled within the keyword **GET** (line 49). These options are set in the local queue attributes section that spread from line 43 until line 56. Instead of priority sequence message delivery there is chosen to use per default the FIFO method. This allows an efficient answering to the messages in the sequence as they are catered to the bridge; the oldest request message is the first to process (**MSGDLVSQ(FIFO)**, line 54). The keyword **HARDENBO** is set to maintain an accurate back out for this queue (line 55). This ensures that messages are not reprocessed erroneously after an emergency restart. As well as, both, the CICS Bridge Monitor Task and the CICS DPL Bridge Task, read messages stored on the MQSeries CICS Bridge queue, shared access onto the queue has to be permitted (**SHARE**, line 56):

```sql
32  DEFINE QLOCAL('SYSTEM.CICS.BRIDGE.QUEUE') REPLACE + 36          DEFPRTY(0) + 37          DEFPSIST(YES) + 38          DESCR('MQSERIES CICS BRIDGE QUEUE') + 39          PUT(ENABLED) + ... 43 *        BOQNAME(' ') + 44          BOTHRESH(0) + 45          CLUSNL(' '), + 46          CLUSTER(' '), + 47          DEFBIND(OPEN)+ 48          DEFSOPT(SHARED) + 49          GET(ENABLED) + 50          INDXTYPE(NONE) + 51          INITQ('CICS01.INITQ') + 52 *        MAXDEPTH(5000) + 53 *        MAXMSGL(4 194 304) + 54          MSGDLVSQ(FIFO) + 55          HARDENBO + 56          SHARE +
```

For future extensions, a trigger and a trigger process have also been created. Since the script CSQ4CKBM should be worked off automatically during every OS/390-server queue manager start up, it has to be pointed within the queue manager startup script MQA1MSTR as described in 5.4.1 “The OS/390-server queue manager MQA1” on page 96 (line 46, Listing 25). After each restart of the OS/390-server queue manager, this queue is always created new.
Tu sum up, on the CICS region there are installed and activated the required MQSeries CICS adapter, the CICS Bridge Monitor Task, and the CICS DPL Bridge Task. On the OS/390-server queue manager has been created the MQSeries CICS Bridge queue also known as request queue. As next, both – the CICS region and the MQSeries queue manager – are connected to activate the MQSeries CICS Bridge.

5.4.2.4 Running the MQSeries CICS Bridge

The CICS transaction CKBR entered on a CICS terminal establishes the connection between the CICS Bridge Monitor Task/CICS DPL Bridge Task and the MQSeries CICS Bridge queue. The complete transaction command is specified as:

\[ \text{CKBR } Q = \langle \text{queue name} \rangle, \text{ AUTH = } \langle \text{option} \rangle, \text{ WAIT = } nnn \]

For instance, the command could be read as:

\[ \text{CKBR } Q = \text{SYSTEM.CICS.BRIDGE.QUEUE}, \text{ AUTH = } \text{LOCAL}, \text{ WAIT = } 30 \]

The parameter \( Q \) specifies the name of the MQSeries CICS Bridge queue. The security parameter \( \text{AUTH} \) is set to the option \( \text{LOCAL} \); security is not used. The last parameter \( \text{WAIT} \) is set to 30 seconds. The following message is displayed within the CICS log MSGUSR, part of the CICS started procedure \( \text{CICSC001} \), when the MQSeries CICS Bridge is successfully started (for reaching this log refer to B.1, page 231):

\[ \text{CSQC700I CKBR 0000043 IBM MQSeries for OS/390 V2.1 - CICS bridge. Copyright(c) 1997,1999 IBM, All rights reserved} \]

\[ \text{CSQC702I CKBR 0000043 Monitor initialization complete} \]

\[ \text{CSQC703I CKBR 0000043 Auth=LOCAL, WaitInterval=30000, Q=SYSTEM.CICS.BRIDGE.QUEUE} \]

Setting the \( \text{WAIT} \) parameter to 30 seconds implies that the CICS Bridge Monitor Task waits this time until a second subsequent request message arrives. When no more requests are outstanding on the queue, the MQSeries CICS Bridge terminates. For each request on the CICS programs \( \text{NACT02} \) and \( \text{NACT03} \) the MQSeries CICS Bridge can be started manually, but this is not recommended. As another possibility, omitting the \( \text{WAIT} \) parameter implies that the bridge is ever running. The log displays \( \text{waitInterval = -1} \) instead \( \text{waitInterval} = 30000 \). However, if the bridge is started as ever running task from a CICS terminal, the terminal is not freed until the CICS Bridge Monitor Task ends. No more entries could be written on the terminal (Figure 31, next page). The CICS terminal can only be freed using three different procedures. First method: changing the queue parameter \( \text{GET} \) of the MQSeries CICS Bridge queue and sets \( \text{GET(DISABLED)} \). Following command is entered on SDSF’s command line:

\[ /!MQA1 ALTER QL(SYSTEM.CICS.BRIDGE.QUEUE) GET(DISABLED) \]

Changing this parameter in a sub panel of the OS/390 MQSeries program is another possibility. The MQSeries MAIN MENU is started from the CMAM when an ‘M’ is entered on the command line. In this menu the parameters, that alter the queue, are set as shown in Figure 32 on the next page. In the sub panel “Alter a Local Queue” the parameter \( \text{GET} \) has to be set to \( \text{NO} \) (Figure 33, next page).
Of course, a shutdown of the MQSeries queue manager MQA1 or a CICS shutdown also terminate the MQSeries CICS Bridge. Following message is written, after a successful shutdown of the MQSeries CICS Bridge, in the CICS log part MSGUSR of CICSC001:

```
CSQC712I CKBR 0000027 Bridge quiescing
CSQC713I CKBR 0000027 Bridge terminated normally
```

Figure 31: After starting the CKBR transaction the monitor is locked, system is in wait status
IBM MQSeries for OS/390 - Main Menu

Complete fields. Then press Enter.

Action ...... 3
1. Display 5. Perform
2. Define 6. Start
3. Alter 7. Stop
4. Delete

Object type ...... QLOCAL +
Name ............. SYSTEM.CICS.BRIDGE.QUEUE
Like .............

Connect to queue manager . . . . . . . : MQA1
Target queue manager : MQA1
Response wait time : 30 seconds

(C) Copyright IBM Corporation 1993,1999. All rights reserved.

Command ===> __________________________________________________________________
F1=Help      F2=Split     F3=Exit      F4=Prompt    F6=QueueMgr  F9=Swap
F10=Messages F12=Cancel

Figure 32: Shutting down the MQSeries CICS Bridge manually – 01

Alter a Local Queue

Complete fields, then press F8 for further fields, or Enter to alter queue.

Queue name ............. : SYSTEM.CICS.BRIDGE.QUEUE
Description ............. : MQSERIES CICS BRIDGE QUEUE

Put enabled ............. Y  Y=Yes,N=No
Get enabled ............. N  Y=Yes,N=No
Usage ............. N=Normal,X=XmitQ
Storage class ............. DEFAULT
Creation method ............. PREDEFINED
Output use count ............. 0
Input use count ............. 1
Current queue depth ............. 0

CSQ9022I !MQA1 CSQMMSGP ' ALTER QLOCAL' NORMAL COMPLETION

Command ===> __________________________________________________________________
F1=Help      F2=Split     F3=Exit      F6=Clusinfo F7=Bkwd      F8=Fwd
F9=Swap      F10=Messages F12=Cancel

Figure 33: Shutting down the MQSeries CICS Bridge manually – 02
5.4.2.5 An automatic start job for the MQSeries CICS Bridge

In contrast to start and stop the MQSeries CICS Bridge manually, a permanent process of the bridge, running in the background, is a better and pleasant way to get the MQSeries CICS Bridge operated. Of course, when the transaction CKBR is entered on a terminal, the bridge never ends, but the terminal is locked as long as the bridge is shut down. The following procedure shows, how to implement the automatic start and the “background” run of the MQSeries CICS Bridge at CICS startup without using a terminal.

The new created COBOL program STRTCKBR starts the bridge during CICS startup (Listing 36, page 236). The script consists of COBOL statements surrounded by JCL code. Such a script is also called a COBOL stub. It is stored in the data set CICS.COMMON.CICSSRC. One part of the COBOL stub – the JCL-statements – can be found in the lines 1-15 and the others in the lines 54-56 of the script, whereas in line 19 the COBOL parameters are introduced by the parameter TRN.SYSIN. From this point the data, that will start the transaction CKBR, is passed to another script called DFHYITVL (Listing 37, page 236). This procedure is stored in the data set CICSTS13.CICS.SDFHPROC. CICS provides it for the COBOL translate, compile, and link-edit jobs.

Firstly, DFHYITVL passes the data to the COBOL translator DFHECP1£ stored in CICSTS13.CICS.SDFHLOAD (cf. Listing 37, page 236, lines 16-24). As next, the data is sent to the COBOL compiler IGYCRCTL stored in the data set IGY.V2R1M0.SIGYCOMP to compile the script (cf. Listing 37, page 236, lines 29-46). All these three script and load modules are provided within the installation of the CICS Transaction Server (DFHYITVL, DFHECP1£) resp. IBM COBOL for OS/390 (IGYCRCTL). Compiling the STRTCKBR script creates the program STRTCKBR that is placed into one of the CICS program load libraries, for example CICS.COMMON.CICSLAD. Such a load library has to be specified in the CICS region startup script CICSC001 (cf. Listing 23, page 229, line 34). A load library works like a path to the programs stored in it. When the program STRTCKBR is called during CICS startup all load libraries are searched to find this program. Hence, the transaction CKBR resp. the MQSeries CICS Bridge is started during the CICS region startup.

The STRTCKBR script starts in line 1 with the name of the compile job AUTCKBR1 and with options set for it:

```
01 //AUTCKBR1 JOB (0),RACF,MSGCLASS=X,REGION=3M,TIME=1439,
02 // CLASS=A,NOTIFY=&SYSUID
09 //JCILIB JCLLIB ORDER=CICSTS13.CICS.SDFHPROC
10 //COMPILE EXEC DFHYITVL, CICS
11 // INDEX='CICSTS13.CICS', TARGET FOR LMOD
12 // PROGLIB='CICS.COMMON.CICLOAD', COBOL COMPILER
13 // AD370HLQ='IGY.V2R1M0',
19 //TRN.SYSIN DD *
```

In line 9 of the STRTCKBR script is written the library where to find the script DFHYITVL called in line 10. Line 12 points to the program load library where the program STRTCKBR is placed to after the compilation. The lib-
rary of the CICS COBOL compiler is set in line 13. Line 19 refers to the JCL stepname TRN in the procedure DFHYITVL to execute the COBOL translator and compiler.

The CICS COBOL program code is introduced by the language statement CBL (or COBOL) in line 20 and ends in line 52 with the COBOL statement GOBACK. Behind the language statement there are listed two translator options within the XOPT option. CICS indicates that the COBOL translator is to process EXEC CICS commands and EDF starts the Execution Diagnostic Facility during compilation. The CICS COBOL program code is subdivided as follows:

```
IDENTIFICATION DIVISION
DATA DIVISION
    WORKING-STORAGE SECTION
PROCEDURE DIVISION
```

Within the IDENTIFICATION DIVISION (line 22) is defined the COBOL program name STRTCKBR (line 23) and a few program data. The DATA DIVISION comprises the data to use in the program (line 28-31). The WORKING STORAGE SECTION of this division defines the parameter CKBRPARM (line 30):

```
20       CBL  XOPTS(CICS,EDF)
21         ***
22         IDENTIFICATION DIVISION.
23         PROGRAM-ID.    STRTCKBR.
...
28         DATA DIVISION.
29         WORKING-STORAGE SECTION.
30       01  CKBRPARM                    PIC X(37) VALUE
31           'Q=SYSTEM.CICS.BRIDGE.QUEUE,AUTH=LOCAL'.
```

The parameter declares the name of the MQSeries CICS Bridge queue 'SYSTEM.CICS.BRIDGE.QUEUE.' The bridge is accessed by CKBR with the authority provided by the default CICS user ID. That is the default specified by AUTH=LOCAL (see ch. 6.4.4.6 “Securing the MQSeries CICS transactions used for the NACT application” on page 195 for more information). The WAIT attribute has been omitted to start an ever running MQSeries CICS Bridge. In the PROCEDURE DIVISION the parameter CKBRPARM is used to execute the CICS transaction CKBR starting the CICS command EXEC CICS START TRANSID (line 48). The COBOL statement GOBACK specifies the logical end of the COBOL program (line 57):

```
48       EXEC CICS START TRANSID('CKBR')
49               FROM(CKBRPARM)
50               LENGTH(37)
51               END-EXEC.
52       GOBACK.
```
The last JCL statement _LKED.SYSIN_ links to the step _LKED_ in the script _DFHYITVL_ to place the compiled CICS COBOL program _STRTCKBR_ into the right program load library, in this case _CICS.COMMON.CICSLOAD_ using the parameter _PROGLIB_ (cf. Listing 37, line 12):

```plaintext
54 //LKED.SYSIN DD *
55 NAME STRTCKBR(R)
```

After the _STRTCKBR_ script is written, it is compiled, when _sub_ is entered on the command line in the editor. If the status report appears including the message part 'MAXCC=0' the job is well done. In case, a problem occurs during the compilation, the SDSF program needs to be started to view the log of the compile job. In the SDSF program the status of the finished job (by typing 'ST') gives an error message back. The printout of the compile job _AUTCKBR1_ (spelled in line 1 of _STRTCKBR_) can be opened by entering an 'S' before the job name. Looking for the word pattern 'STMT NO. MESSAGE' shows an error report with a detailed description of the failure and lists the line number where the failure appeared. It should be no problem to solve this, for example, there is written a wrong parameter, a COBOL statement is not set or written wrong, or the library entries links to the wrong one.

In the last step, the CICS Program List Table (PLT) that holds CICS programs to be started during the CICS region initial start has to be adapted. But beware, there are two different PLTs – one is loaded during the CICS region initial start and one is executed during the termination of this CICS region. Within the CICS SIT definition script _C001_ (stored in the data set _CICS.COMMON.SYSIN_) both tables are loaded using the SIT-parameters PLTPI (PLT Program Initialisation) resp. PLTSD (PLT ShutDown):

```plaintext
16 PLTPI=IT,                    PGMS AT CICS INIT (USE TOR PLT)
17 PLTSD=ST,                    PGMS AT CICS TERM (USE TOR PLT)
```

_IT_ and _ST_ are suffixes for the PLT scripts. Completely, they are named as _DFHPLTIT_ for the initial table and _DFHPLTST_ for the system termination table. Because the string DFH is automatically added during compilation in front of the script name, their real script names consists of the string PLT plus the suffixes IT resp. ST – PLTIT resp. PLTST. Theses scripts are stored in the dataset _CICS.COMMON.TABSRC_. The compiled tables are placed into the dataset _CICS.COMMON.TABLIT_ from where they are loaded.

A new entry in the table PLTIT is required to start the COBOL program script _STRTCKBR_ to initiate the MQSeries CICS Bridge queue during the initial start of the CICS region (cf. also Listing 38, page 236):

```plaintext
09 //S1   EXEC DFHAUPLE
10 //ASSEM.SYSUT1 DD *
18       DFHPLT TYPE=ENTRY, PROGRAM=STRTCKBR   ENABLE CICS BRIDGE
```
PLTIT is compiled using the procedure \textit{DFHAUPLE} (Listing 39, page 236) that is also stored in the dataset 
\textit{CICS.COMMON.TABSRC}. Therefore, a concatenation is not required to refer to another library. \textit{DFHAUPLE} is 
called in line 9. The JCL step \textit{ASSEM.SYSUT1} in line 10 links to the step \textit{ASSEM} in the \textit{DFHAUPLE} script 
(Listing 39, line 31):

\begin{verbatim}
31 //ASSEM EXEC PGM=IEBGENER
\end{verbatim}

After the successful compiling of the table PLTIT, the MQSeries CICS Bridge is automatically started during 
every initial start of the CICS region A06C001. However, to load the initial PLT and to start the MQSeries CICS 
Bridge, the CICS region has to be restarted firstly. It should be considered, that, before starting the CICS region, 
the OS/390-server queue manager MQA1 has also to be terminated. Afterwards, the queue manager must be 
started at first because the CICS region may only be started after the queue manager is up. Whilst the queue manager 
is started, it executes the script \textit{CSQ4CKBM} to create the MQSeries CICS Bridge queue \texttt{SYSTEM.CICS.BRIDGE.QUEUE} to MQA1. When the CICS region A06C001 is started afterwards, the compiled CICS COBOL program \texttt{STRTCKBR} is executed to connect the CICS region to the MQSeries CICS Bridge queue. The 
successful connection is reported in the CICS \texttt{MSGUSR} log as follows:

\begin{verbatim}
DFHPG0209 03/01/04 02:04:29 A06C001 STCCICS CPLT PPT entry for STRTCKBR has been autoinstalled using model DFHPGAPG.
CSQC700I CKBR 0000027 IBM MQSeries for OS/390 V2.1 - CICS bridge. Copyright(c) 1997,1999 IBM, All rights reserved
CSQC702I CKBR 0000027 Monitor initialization complete
CSQC703I CKBR 0000027 Auth=LOCAL, WaitInterval=-1, Q=SYSTEM.CICS.BRIDGE.QUEUE
\end{verbatim}

\textbf{Note:} An ever running MQSeries CICS Bridge queue can decrease the performance of the 
system. Therefore, it is suggested to enable triggering the MQSeries CICS Bridge in future.

\subsection*{5.4.3 The required queue manager objects}

\subsubsection*{5.4.3.1 The transmission queue \texttt{TBUSSE.NACT}}

Besides the MQSeries CICS Bridge queue, the request queue, four more objects has to be defined to the 
OS/390-server queue manager: the transmission queue, the remote queue \textit{definition}, the channel sender of the re-
response channel, and the channel receiver of the request channel. All objects are created using the panels of the 
MQSeries product on OS/390. The first panel, the MQSeries “Main Menu”, can be reached from the CMAM 
when an \texttt{M} is entered on the command line of the first screen appearing after log on to the OS/390-server.

Firstly, the transmission queue \texttt{TBUSSE.NACT} for the OS/390-server queue manager MQA1 is to specify 
(Figure 34, page 114). Into the field \textit{object type} of the MQSeries “Main Menu” panel is entered the type of 
the queue which must always be \texttt{QLOCAL}, because only a local queue can be transformed into a transmission 
queue. Pressing the PF4 key opens a selection of all object types. When number 2 is chosen, shortcut for \texttt{QLOC-}
AL, and the enter key is pressed, this object type is filled into the field **Object type.** In the field **Name** has to be entered the name of the queue. It is again alluded, that the name of this transmission queue must have the same name as the WINDOWS2000-client queue manager.

The transmission queue is created, when the field **Action** is set to 2 as synonym for 'Define. As next screen the sub panel “Define a Local Queue” appears. When the parameter **Usage** is set to X the transmission queue is created. Both parameters, **Put enabled** and **Get enabled,** have to be set to Y. With it, the transmission queue can get response messages from the MQSeries CICS Bridge queue and can transfer these messages onto the reply-to queue on the WINDOWS2000-client queue manager. Furthermore, a description for the transmission queue is added (Figure 35, page 114). All next panels for defining additional queue parameters can be reached when pressing the PF8 key. On the second panel, the parameter **Message delivery sequence** is changed from **Priority** to **FIFO** to retrieve messages in order of “First in, First out” method. As a future aspect, shared access to the transmission queue is permitted and more than one application could retrieve messages; the parameter **Permit shared access** is set to Y. If shared access is forbidden, only one specified application can communicate with the queue. Coherently with the parameter **Permit shared access,** the parameter **Default share option** has to be set to S. In case of an illicit access, the **Exclusive** option (E) has to be set (Figure 36, page 115).

On the next panel a trigger process can be defined (press PF8), however, this has been omitted. The panel for event control has been also skipped and on the last panel for backout reporting the parameter **Hardenbo** has been set to Y to maintain an accurate back out for this queue (Figure 37, page 115).

When pressing the enter key, the queue is created. If the queue is successfully created the first sub panel screen appears displaying following message in the lower area:

```
CSQ9022I !MQA1 CSQMSGP ' DEFINE QLOCAL' NORMAL COMPLETION
```
Generation of a Java front end for a standalone CICS application accessed through MQSeries &
Securing CICS with RACF

IBM MQSeries for OS/390 - Main Menu

Complete fields. Then press Enter.

Action . . . . . . . . 2  1. Display  5. Perform
            2. Define  6. Start
            3. Alter   7. Stop
            4. Delete

Object type . . . . QLOCAL +
Name . . . . . . . . TBUSSE.NACT
Like . . . . . . . .

Connect to queue
manager . . . . : MQA1
Target queue manager : MQA1
Response wait time : 30 seconds

(C) Copyright IBM Corporation 1993,1999. All rights reserved.

Command ===> __________________________________________________________________
F1=Help      F2=Split     F3=Exit      F4=Prompt    F6=QueueMgr  F9=Swap
F10=Messages F12=Cancel

Figure 34: Defining the transmission queue TBUSSE.NACT – 01

Define a Local Queue

Complete fields, then press F8 for further fields, or Enter to define queue.

More: +

Queue name . . . . . . . . TBUSSE.NACT
Description . . . . . . . XMITQ THAT STORES RESPONSE______
                     MESSAGES TO BE SEND__________

Put enabled . . . . . . . Y  Y=Yes,N=No
Get enabled . . . . . . . Y  Y=Yes,N=No
Usage . . . . . . . . . . X  N=Normal,X=XmitQ
Storage class . . . . . . DEFAULT

Command ===> __________________________________________________________________
F1=Help      F2=Split     F3=Exit      F7=Bkwd      F8=Fwd       F9=Swap
F10=Messages F12=Cancel

Figure 35: Defining the transmission queue TBUSSE.NACT – 02
### Define a Local Queue

Press F7 or F8 to see other fields, or Enter to define queue.

More: - +

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default persistence</td>
<td>N (Yes), N (No)</td>
</tr>
<tr>
<td>Default priority</td>
<td>0 (0 - 9)</td>
</tr>
<tr>
<td>Message delivery sequence</td>
<td>F (Priority), F (FIFO)</td>
</tr>
<tr>
<td>Permit shared access</td>
<td>Y (Yes), N (No)</td>
</tr>
<tr>
<td>Default share option</td>
<td>S (Exclusive), S (Shared)</td>
</tr>
<tr>
<td>Index type</td>
<td>N (None), M (MsgId), C (CorrelId), T (MsgToken)</td>
</tr>
<tr>
<td>Maximum queue depth</td>
<td>9999999999 (0 - 999999999)</td>
</tr>
<tr>
<td>Retention interval</td>
<td>9999999999 (0 - 999999999) hours</td>
</tr>
<tr>
<td>Cluster name</td>
<td>____________________________</td>
</tr>
<tr>
<td>Cluster namelist name</td>
<td>____________________________</td>
</tr>
<tr>
<td>Default bind</td>
<td>O (Open), N (Notfixed)</td>
</tr>
</tbody>
</table>

Command ====> ____________________________

F1=Help          F2=Split          F3=Exit          F4=Bkwd          F5=Fwd          F6=Swap
F7=Bkwd          F8=Fwd            F9=Swap

Figure 36: Defining the transmission queue TBUSSE.NACT – 03

### Define a Local Queue

Press F7 to see previous fields, or Enter to define queue.

More: -

### Backout Reporting

Backout threshold . . . 0 (0=No backout reporting)

Harden backout counter . . Y (Yes), N (No)

Backout requeue name . . ____________________________

Command ====> ____________________________

F1=Help          F2=Split          F3=Exit          F4=Bkwd          F5=Fwd          F6=Swap
F7=Bkwd          F8=Fwd            F9=Swap

Figure 37: Defining the transmission queue TBUSSE.NACT – 04
5.4.3.2 The remote queue definition TBUSSE.NACT.REPLYQ

For the reply-to queue TBUSSE.NACT.REPLYQ created on the WINDOWS2000-client queue manager has to be defined a remote queue definition on the OS/390-server queue manager (Figure 38). It is chosen to give it the same name as the reply-to queue on the remote queue manager. In the field Object type is entered the object type QREMOTE; if the selection panel is used (press PF4) the number 3 is to be chosen. A description for this object is also added. On the next panel “Define a Remote Queue” (Figure 39, next page) has to be specified the name of reply-to queue – TBUSSE.NACT.REPLYQ – on the field Remote name. The WINDOWS2000-client queue manager name TBUSSE.NACT has to be entered into the field Remote queue manager. As last definition, the name of the transmission queue for MQA1 – TBUSSE.NACT – has to be entered into the field Transmission Queue. Pressing PF8 leads to the next panel on which nothing needs to be altered. When hitting the enter key, the remote queue definition is created and if successfully, following message is displayed:

```
CSQ9022I MQA1 CSQMMSGP ' DEFINE QREMOTE' NORMAL COMPLETION
```

![Figure 38: Defining the remote queue definition TBUSSE.NACT.REPLYQ – 01](image)
Define a Remote Queue

Complete fields, then press F8 for further fields, or Enter to define queue.

More: +

Queue name . . . . . . . . . TBUSSE.NACT.REPLYQ
Description . . . . . . . . . REMOTE QUEUE DEFINITION FOR THE_
REPly-TO QUEUE ON TBusse.NACT___

Put enabled . . . . . . . . . Y  Y=Yes,N=No
Default persistence . . . . N  Y=Yes,N=No
Default priority . . . . . . 0  0 - 9
Remote name . . . . . . . . . TBUSSE.NACT.REPLYQ
Remote queue manager . . . TBusse.NACT_____________________________
Transmission queue . . . . . TBusse.NACT_________________________________

Command ===> __________________________________________________________________
F1=Help      F2=Split     F3=Exit      F7=Bkwd      F8=Fwd       F9=Swap
F10=Messages F12=Cancel

Figure 39: Defining the remote queue definition TBusse.NACT.REPLYQ – 02

5.4.3.3 The dead-letter queue MQA1.DEAD.QUEUE

The dead-letter queue has been defined on the OS/390-server queue manager MQA1 using the script CSQ4INYG provided within the MQSeries installation (cf. Listing 32, page 235). Therefore, a manually creation using the MQSeries panels is not required. The dead-letter queue is also a local queue as same as the MQSeries CICS Bridge queue. The MQSC \texttt{DEFINE QLOCAL} plus the name of the queue \texttt{MQA1.DEAD.QUEUE}, and setting the keyword \texttt{USAGE} to \texttt{NORMAL} create the queue (line 143, 164). Wrong-sent/failure messages are stored onto the queue (line 153) until they are picked up to check them (line 147). Activating \texttt{DEFPSIST} sets all dead-letter messages persistent on the queue (line 149). The shared access to the queue has also to be enabled because all programs should be able to store wrong-sent/failure messages onto the dead-letter queue (lines 154, 155). Messages to be stored onto the queue uses the FIFO method (line 156). A backout for the dead-letter messages is not required, therefore \texttt{NOHARDENBO} is specified (line 160). For the dead-letter queue a trigger is also not required; \texttt{NOTRIGGER} has been set (line 177):

\begin{verbatim}
143 DEFINE QLOCAL( 'MQA1.DEAD.QUEUE' ) REPLACE +
... 146       DESC( 'MQA1 dead-letter queue' ) +
147      PUT( ENABLED ) +
148      DEFPRTY( 0 ) +
149      DEFPSIST( YES ) +
... 153      GET( ENABLED ) +
154      SHARE +
155      DEFSOPT( SHARED ) +
156      MSGDLVSQ( FIFO ) +
... 
\end{verbatim}
Further, the same script also defines some needed MQSeries objects, an initiation queue required for the MQSeries CICS adapter and a default transmission unused by the MQSeries CICS application.

### 5.4.3.4 The channel sender TBUSSE.NACT.OS.WIN

A channel sender needs to be created to send the response messages from the transmission queue to the WINDOWS2000-client queue manager. On the MQSeries “Main Menu” panel is entered the string `CHLSENDER` into the field `Object Type` (Figure 40, next page). If the selection panel is used (press PF4), number 13 has to be chosen. The name of the channel sender is set to `TBUSSE.NACT.OS.WIN`. The character sequence `OS.WIN` indicates the direction from where the channel sender transmits the response messages to which target – from OS/390 to WINDOWS2000. Some more parameters has to be specified on the next panel “Define a Sender Channel” (Figure 41, next page). The response messages are sent to the remote queue manager using the TCP/IP protocol with the port number 1414 as default. Parameter `T` is specified in the field `Transport type` to use the TCP/IP protocol. In the field `Connection name`, the IP address of the remote WINDOWS2000 computer is filled in, in this case `80.135.230.111`. The host address is followed by the port number `1414` in parentheses. The closing parenthesis is optional. Into the field `Transmission queue` is entered the name of the local transmission queue. Fields on the next three panels need not to be set up. After confirming the creation of the channel sender, this message appears:

```
CSQ9022I !MQA1 CSQMMGP ' DEFINE CHANNEL' NORMAL COMPLETION
```
IBM MQSeries for OS/390 - Main Menu

Complete fields. Then press Enter.

Action ............ 2   1. Display   5. Perform
                     2. Define   6. Start
                     3. Alter    7. Stop
                     4. Delete

Object type ......... CHLSENDER +
Name ............... TBUSSE.NACT.OS.WIN
Like ..............

Connect to queue
manager ........... MQA1
Target queue manager : MQA1
Response wait time : 30 seconds

(C) Copyright IBM Corporation 1993,1999. All rights reserved.

Command ===> ____________________________
F1=Help      F2=Split     F3=Exit      F4=Prompt    F6=QueueMgr  F9=Swap
F10=Messages F12=Cancel

Figure 40: Defining the channel sender for the transmission queue – 01

Define a Sender Channel

Complete fields, then press F8 for further fields, or Enter to define channel.

More:    +

Channel name ............ TBUSSE.NACT.OS.WIN
Description .............. CHANNEL FOR THE XMITQ TO SEND THE RESPONSE MESSAGES

Transport type .......... T L=LU6.2,T=TCP/IP
Connection name .......... 80.135.230.111(1414)
LU6.2 mode name ........... ________________
LU6.2 TP name ............ ________________
Transmission queue ...... TBUSSE.NACT_____________________________________

Command ===> ____________________________
F1=Help      F2=Split     F3=Exit      F7=Bkwd     F8=Fwd     F9=Swap
F10=Messages F12=Cancel

Figure 41: Defining the channel sender for the transmission queue – 02
5.4.3.5 The channel receiver TBUSSE.NACT.WIN.OS

Request messages sent by the channel sender of the WINDOWS2000-client queue manager to the OS/390-server queue manager are received by the channel receiver TBUSSE.NACT.WIN.OS. The channel receiver must have the same name as the remote channel sender to create a message channel. After the message is received it is forwarded to the request queue – the MQSeries CICS Bridge queue.

Such a channel receiver is created when CHLRECEIVER is entered into the field Object type; number 15 can also be chosen from the selection panel that is reached by pressing PF4 (Figure 42). The name of the channel receiver has to be entered in the associate field. On the next both panels is no modifying required, except adding a description if wanted (Figure 43, next page). Note: The field sequence number wrap must have the same number as specified on the remote channel sender, usually it is set to 999999999. After confirming the creation of the channel receiver, the same message as for the channel sender is displayed:

```
CSQ9022I !MQA1 CSQMMSGP ' DEFINE CHANNEL' NORMAL COMPLETION
```

All required MQSeries objects for the OS/390-server queue manager MQA1 are now defined. As next, the MQSeries objects for the WINDOWS2000-client queue manager TBUSSE.NACT have to be created.

```
<table>
<thead>
<tr>
<th>IBM MQSeries for OS/390 - Main Menu</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete fields. Then press Enter.</td>
</tr>
<tr>
<td>Action . . . . . . . . . . . . . 2</td>
</tr>
<tr>
<td>1. Display   5. Perform</td>
</tr>
<tr>
<td>2. Define    6. Start</td>
</tr>
<tr>
<td>3. Alter     7. Stop</td>
</tr>
<tr>
<td>4. Delete</td>
</tr>
<tr>
<td>Object type . . . . . . . . . . . . CHLRECEIVER +</td>
</tr>
<tr>
<td>Name . . . . . . . . . . . . . . . TBUSSE.NACT.WIN.OS</td>
</tr>
<tr>
<td>Like . . . . . . . . . . . . . . . ________________________________________________</td>
</tr>
<tr>
<td>Connect to queue manager . . . . . . : MQA1</td>
</tr>
<tr>
<td>Target queue manager : MQA1</td>
</tr>
<tr>
<td>Response wait time . . . . . . . . 30  seconds</td>
</tr>
<tr>
<td>(C) Copyright IBM Corporation 1993,1999. All rights reserved.</td>
</tr>
</tbody>
</table>
```

Figure 42: Defining the channel receiver for the CICS Bridge queue – 01
Define a Receiver Channel

Complete fields, then press F8 for further fields, or Enter to define channel.

More: +

Channel name ............ TBUSSE.NACT.WIN.OS
Description ............ GETS RESPONSE MESSAGES, FORWARDS THEM TO THE MQ CICS BRIDGE QUEUE

Put authority .......... D D=Default,C=Context,O=OnlyMCA,A=AltMCA

Figure 43: Defining the channel receiver for the CICS Bridge queue – 02
5.5 Setting up MQSeries on the Windows2000 client

5.5.1 The WINDOWS2000-client queue manager TBUSSE.NACT

For the management of the MQSeries system on WINDOWS2000, two Snap-Ins for the Microsoft Management Console (MMC) have been installed – the MQSeries Services and the MQSeries Explorer. Using both programs help to create, start, stop, and delete queue managers and its objects on the WINDOWS2000 client. MQSeries objects are managed by the MQSeries Explorer, as well as the status of the queue managers can be checked with this Snap-In. In contrast, the MQSeries Services do all that what a user understands in services. It starts and stops the MQSeries command server, the channel initiator program, and the channel listener program. For instance, these services can be configured to start them automatically, to use error checking, or to use transmission protocols. Furthermore, the alert monitor can be started in case of an abnormal error, and a trace facility can trace the whole processes.

After the MQSeries application is installed, the WINDOWS2000-client queue manager is to be created. It manages message queuing between the JAVA presentation logic and the CICS business logic. The WINDOWS2000-client queue manager is created using the MQSeries assistant. In the MQSeries Services window, a right mouse click on Queue Managers as shown in Figure 44 on the next page pops up a menu on which New and then Queue Manager is clicked to open the panel “Create Queue Manager (Step 1)”. On this first panel the name of the queue manager TBUSSE.NACT, the name of the default transmission queue TBUSSE.NACT.XMITQ, and the name of the dead letter queue TBUSSE.NACT.DEAD.LETTER.QUEUE are keyed in (Figure 45, a), next page). However, these queues exist not yet, they are defined and created later as described in chapter “The required queue manager objects” on page 125. When clicking on Next, the panel “Create Queue Manager (Step 2)” appears (Figure 45, b)). On this panel can be defined where to file the error log. The default parameters are chosen and a click on Next leads to the panel “Create Queue Manager (Step 3)” to mark the box Start Queue Manager (Figure 45, c)). On the last panel “Create Queue Manager (Step 4)” the box Create listener configured for TCP/IP is marked and the port number 1414 is entered in the box Listen on port number (Figure 45, d)). This creates a listener, automatically. When clicking on the Finish button the queue manager is created and already started with all services it needs. It is also not required to create manually a command server and a channel initiator. These services are always loaded automatically when the queue manager starts. The green upturned arrow in front of the queue manager's name confirms that the queue manager is started. If the arrow is red coloured and is oriented down the queue managers is stopped. All created and started services are shown in the right window of the MQSeries Services (Figure 46, page 124). After switching to the MQSeries Explorer window, all created default MQSeries system objects can be viewed, e.g. the created system queues (Figure 47, page 124). The option Show/Hide System Objects must be enabled to see the default objects. It is recommended to switch off this option to not losing the overview, if own objects need to be defined.
Figure 44: Defining the queue manager TBUSSE.NACT – 01

Figure 45: Defining the queue manager TBUSSE.NACT – 02
Generation of a Java front end for a standalone CICS application accessed through MQSeries & Securing CICS with RACF

Figure 46: The MQSeries Services, the green arrow indicates that the queue manager is up

Figure 47: System objects for the queue manager TBUSSE.NACT
5.5.2 The required queue manager objects

5.5.2.1 A definition script for the queue manager objects

As on the OS/390-server queue manager adjacent queue manager objects has to be created on the WINDOWS2000-client queue manager. A transmission queue to send the request messages to OS/390-server queue manager, a reply-to queue that receives the response messages from the OS/390-server queue manager, a remote queue definition for the request queue on OS/390, and a dead-letter queue on which all messages are stored that are unable to send have been created. As channel objects, a channel sender and a channel receiver have been created to build the request channel. Additionally, an MQI channel has to be created to connect the WINDOWS2000-client queue manager with the JAVA application.

All these definitions can be defined using the MQSeries assistant in the MQSeries Explorer. For example, the reply-to queue can be defined as shown in Figure 48. Clicking on Local queues opens the panel “Create Local Queue” on which all specifications can be made. Instead using this manual method as used for the OS/390-server queue manager objects, a batch script and an MQSC script have been created. In an MQSC script all parameters can also be set for every queue manager object. Such a script can be either created for the OS/390-server queue manager (for example Listing 30, page 235; refer also to chapter 5.4.2.2 “Configuring CICS to use the MQSeries CICS Bridge”, page 101) or for the WINDOWS2000-client queue manager (cf. Listing 40, page 236).

![Figure 48: NACT objects for the queue manager TBUSSE.NACT](image)
The MQSC script is executed within the batch file `def.cmd` (cf. Listing 41, page 236). The first batch command `strtmqm` checks whether the queue manager has been started previously (line 06). If not, the local queue manager `TBUSSE.NACT` is started. Usually, the second batch command `runmqsc`, executed from a MS-DOS-console, starts an MQSeries console on which MQSCs can be entered. However, the command `runmqsc` can also be used to read input data from an external MQSC script and to write a report to another file (line 11). Behind both batch commands it is required to write the name of the queue manager referring to:

```
06 strtmqm TBUSSE.NACT
11 runmqsc TBUSSE.NACT < MQadmvs.tst > MQadmvs.out
```

All MQSCs are put together in the MQSC file `mqadmvs.tst` (Listing 40). The report is sent to the output file `mqadmvs.out`. If following message stands at the end of the report file, all MQSCs were successfully executed:

```
8 MQSC commands read.
No commands have a syntax error.
All valid MQSC commands were processed.
```

In the next sections are described the required MQSCs with their keywords and attributes for each MQSeries object that needs to be created. However, default parameters are adopted for the most objects. For a complete specification of all MQSCs, their keywords, and attributes please refer to [MCR00].

### 5.5.2.2 The transmission queue TBUSSE.NACT.XMITQ

The transmission queue `TBUSSE.NACT.XMITQ` is defined with the MQSC `DEFINE QLOCAL`. In parentheses the name of the queue is written (cf. Listing 40):

```
01 DEFINE QLOCAL('TBUSSE.NACT.XMITQ') REPLACE +
02 DESC('XMITQ that stores request messages to be send') +
03 TRIGGER TRIGTYPE(FIRST) +
04 TRIGDATA(TBUSSE.NACT.WIN.OS) +
05 PROCESS('MQCLIENT') +
06 INITQ(SYSTEM.CHANNEL.INITQ) +
05 USAGE(XMITQ)
```

The keyword `REPLACE` defines the transmission queue always new when the command file is executed. Such an MQSC can often be spread over a few lines. The plus sign at the end of a line continues the command on the next line. Of course, on the last line of the MQSC this character has to be omitted. When specifying `DESCR`, a description of the queue can be added, as shown in line 2. An important note aside – the name of the queue and the description must be always surrounded by single quotes. In case trigger messages should be written to the initiation queue to trigger an application (named by the `PROCESS` keyword), `TRIGGER` has to be specified. Such a trigger is defined but not used for the MQSeries CICS business application (line 3). This is a reference for future use. A trigger message is send to the initiation queue when the first request message arrives on the transmission
The MQSeries CICS Business Application MQNACT

queue. TRIGDATA names the data that is additionally inserted in the trigger message, in that case the name of the channel sender (line 4). When an error occurs, a message is displayed on the screen to check whether the channel sender has been started as the message was sent or not. On the keyword PROCESS is specified the triggered business application (line 5). The trigger messages are transmitted to the initiation queue SYSTEM.CHANNEL.INITQ specified on the keyword INITQ (line 6). When the keyword USAGE is set to XMITQ, the local queue becomes a transmission queue (line 7).

As referenced by the MQSC that creates the transmission queue, a process object needs to be defined to use the trigger. This object is created within the MQSeries command DEFINE PROCESS and the assigned name MQCLIENT (cf. Listing 40):

```
09 DEFINE PROCESS('MQCLIENT') REPLACE +
10        DESCR('MQSeries CICS Application') +
11        APPLICID('java MQClient') +
12        APPLTYPE(WINDOWSNT) +
13        USERDATA('80.135.230.111 TBUSSE.NACT.CLIENT TBUSSE.NACT +
14                  TBUSSE.NACT.REMOTEQ TBUSSE.NACT.REPLYQ')
```

The keyword APPLICID specifies the application to be started and APPLTYPE specifies the application type. Because the application is running on a WINDOWS2000 operating system, the type has to be WINDOWSNT. On the parameter USERDATA is listed a string consisting of user-specific information for the application to be started. This specification can be used by a trigger monitor. The trigger monitor sends the user data as part of the parameter list to the started application. The trigger monitor has been previously created and started on the WINDOWS2000-client queue manager to use the trigger.

5.5.2.3 The reply-to queue TBUSSE.NACT.REPLYQ

All response messages are sent from the OS/390-server queue manager onto the reply-to queue. This queue is a local queue defined also with the MQSC DEFINE QLOCAL (line 16, Listing 40). When the keyword USAGE is omitted only a local queue is created. Setting the keyword DEFPERSIST to YES make messages persistent to survive a queue manager restart (line 18). If NO is set, all messages are lost during a restart. As defined for the request queue on OS/390, the share option is also set using the keyword SHARE (line 19). Hence, shared access to the reply-to queue is permitted. For example, two independent screens of the application have been started; both can use the same reply-to queue to get the response messages:

```
16 DEFINE QLOCAL('TBUSSE.NACT.REPLYQ') REPLACE +
17        DESCR('Reply-to queue to store the response messages') +
18        DEFPERSIST(YES) +
19        SHARE
```
5.5.2.4 The remote queue definition TBUSSE.NACT.REMOTEQ

On the WINDOWS2000-client queue manager a remote queue definition for the request queue/MQSeries CICS Bridge queue on OS/390 has to be created. The name of this remote queue definition is not the same as the name of the request queue on the OS/390-server queue manager (SYSTEM.CICS.BRIDGE.QUEUE). Because objects having the leading characters SYSTEM are system objects for MQSeries. When the option “Show/Hide System Objects” in the MQSeries Explorer is disabled, system objects are not displayed and hence, the remote queue definition would also not be displayed. Therefore, the remote queue definition is named as TBUSSE.NACT.REMOTEQ defined with the MQSC DEFINE QREMOTE (line 21, cf. Listing 40, page 236):

```
21 DEFINE QREMOTE('TBUSSE.NACT.REMOTEQ') REPLACE +
22        DESCR('Remote queue definition for the MQSeries CICS Bridge') +
23        RNAME('SYSTEM.CICS.BRIDGE.QUEUE') +
24        RQMNAME('MQA1') +
25        XMITQ('TBUSSE.NACT.XMITQ')
```

The request queue name is specified on the keyword RNAME and the remote queue manager is specified within RQMNAME. Onto which transmission queue the request message is transferred to, specifies the keyword XMITQ. This information is grabbed into a header data which is added to the request data. Both the header and the request data builds the request message.

5.5.2.5 The dead letter queue TBUSSE.NACT.DEAD.LETTER.QUEUE

As specified within the definition of the WINDOWS2000-client queue manager the default dead letter queue does not exist yet. Such a queue is also a local queue and is created using the same MQSC as for the definition for the reply-to queue as defined within the chapter 5.5.2.3 “The reply-to queue TBUSSE.NACT.REPLYQ” on page 127 (line 27, Listing 40). It is only required to set the SHARE option to permit shared access to the queue to store all wrong-sent messages on it (line 28). Other parameters need not to be defined; they are set automatically to default parameters:

```
27 DEFINE QLOCAL('TBUSSE.NACT.DEAD.LETTER.QUEUE') REPLACE +
28        DESCR('Dead letter queue on TBUSSE.NACT') +
29        SHARE
```

5.5.2.6 The server connection TBUSSE.NACT.CLIENT

For the communication between the WINDOWS2000-client queue manager TBUSSE.NACT and the JAVA application an MQI channel is used. It consists of the server connection TBUSSE.NACT.CLIENT and an automatically created client connection. The server connection is part of the channel objects, so it is created with the same MQSC DEFINE CHANNEL. Within the keyword CHLTYPE the type of the channel is set, in that case
SVRCONN, to specify the server connection. This parameter must be written immediately behind the MQSC and before all other keywords defining the other parameters. The MQI channel uses the TCP/IP transport type, which is set with the keyword TRPTYPE. MCAUSER sets up the user identifier for the MCA. However, a user identifier is not used because there is no need for it (cf. Listing 40):

```
31 DEFINE CHANNEL('TBUSSE.NACT.CLIENT') CHLTYPE(SVRCONN) REPLACE +
32      DESCR('MQI channel to connect the application with the QMGR') +
33      TRPTYPE(TCP) +
34      MCAUSER(' ')
```

5.5.2.7 The channel sender TBUSSE.NACT.WIN.OS

As another part of the channel family it is required to create the channel sender for the request channel on the WINDOWS2000-client (cf. Listing 40). Behind the MQSC DEFINE CHANNEL and the channel's name, it must be specified the channel type SDR within the keyword CHLTYPE to create the channel sender (line 36). Message to be transmitted to the OS/390-server queue manager use TCP/IP specified with the keyword TRPTYPE (line 38). The keyword XMITQ specifies the name of the transmission queue (line 39). This queue uses the channel sender to transfer the messages to the OS/390-server queue manager. The keyword CONNAME sets the IP address of the OS/390-server (line 40). MCAUSER leaves blank (line 41):

```
36 DEFINE CHANNEL('TBUSSE.NACT.WIN.OS') CHLTYPE(SDR) REPLACE +
37      DESCR('Channel for the XMITQ to send the request messages') +
38      TRPTYPE(TCP) +
39      XMITQ('TBUSSE.NACT.XMITQ') +
40      CONNAME('139.18.4.97(1414)') +
41      MCAUSER(' ')
```

5.5.2.8 The channel receiver TBUSSE.NACT.OS.WIN

The channel receiver of the response channel has been created using same MQSC as used for the channel sender. However, within the keyword CHLTYPE the attribute "RCVR" is specified. RCVR is a shortcut for the type channel receiver (line 43). Response message to be received use the TCP/IP protocol (line 45). The keyword SEQWRAP sets where the sequence numbers wrap of messages starts. The next number wrap always begins at 1. This number must match the same number as specified for the channel sender of the request channel. A user identifier for the channel receiver MCA is also not used; the default one is taken (cf. Listing 40):

```
43 DEFINE CHANNEL('TBUSSE.NACT.OS.WIN') CHLTYPE(RCVR) REPLACE +
44      DESCR('Gets response messages & forwards them to the local REPLYQ') +
45      TRPTYPE(TCP) +
46      SEQWRAP(999999999) +
47      MCAUSER(' ')
```
Generation of a Java front end for a standalone CICS application accessed through MQSeries & Securing CICS with RACF

Figure 49 shows all defined queues as well as Figure 50 on the next page shows all created channels of the WINDOWS2000-client queue manager TBUSSE.NACT.

Hence, all definitions for the WINDOWS2000-client queue manager TBUSSE.NACT and the OS/390-server queue manager MQA1 have been made. However, before connecting both queue managers, the JAVA presentation and MQSeries communication logic have to be created.

![Figure 49: All defined queues for TBUSSE.NACT](image-url)
Figure 50: All created channels for TBUSSE.NACT
5.6 Building the JAVA application

5.6.1 Coding the MQSeries communication logic

5.6.1.1 Creating a connection to the WINDOWS2000-client queue manager

The JAVA-file MQCommunicator.java consists of the MQSeries communication logic, that uses the MQI to establish a connection to the MQSeries WINDOWS2000-client queue manager, to send the request messages to the OS/390 business logic, and to receive the response messages from the OS/390 business logic (Listing 42, page 237). With this JAVA-application an account record is read and displayed on the WINDOWS200-client. For using the MQI within JAVA-applications, IBM offers the product extension MA8811. Within the JAVA-package com.ibm.mq.jar all required classes for the MQI-calls are provided.

For using the MQI to send and receive messages, an instance of MQCommunicator is constructed:

```java
09 public class MQCommunicator
...
41 public MQCommunicator(String hostname, String channel, String qManager,     +
    String aRequestQueue, String aReplyQueue)
```

The parameters used for the MQCommunicator constructor method read as follows:

- `hostname` - IP-address of the WINDOWS2000 computer
- `channel` - Name of the server connection of the MQI-channel
- `qManager` - Name of WINDOWS2000-client queue manager
- `aRequestQueue` - Name of the Remote Queue Definition (= Request Queue on OS/390)
- `aReplyQueue` - Name of the Reply-To-Queue

Firstly, a connection to the WINDOWS2000-client queue manager has to be established using the constructor method MQQueueManager of the MQSeries supplied JAVA-class MQQueueManager. Before constructing an MQQueueManager instance for use in client mode, some static member variables must be set in the MQSeries supplied JAVA-class MQEnvironment because they take effect when the constructor is called – that are hostname, channel, and port (lines 44-46). The application start command passes the IP-address to hostname and the name of the MQI-channel to channel. The port on which the MQSeries Server listens for incoming connection requests is hard coded within the attribute port to 1414 in the JAVA source program. The method enableTracing is commented, when enabled, the MQSeries Client JAVA trace facility is started (line 47).

After setting these parameters, the connection to the WINDOWS2000-client queue manager `qManager` can be established, when an instance of MQQueueManager is created (line 51). The creation of such an object is in-

---

11 For the MA88-extension downloading instructions refer to 5.2.1 on page 83.
intercepted by an exception, in case the queue manager cannot be connected. If the connection fails, CICS sends
back a completion code to get to the bottom of the error (lines 95-104, please refer to Listing 42).

```java
44 MQEnvironment.hostname = hostname;
45 MQEnvironment.channel = channel;
46 MQEnvironment.port = 1414;
47 // MQEnvironment.enableTracing(1, System.out);
51 qMgr = new MQQueueManager(qManager);
```

### 5.6.1.2 Opening the MQSeries queues for message transport

Before both queues are opened to perform PUTs and GETs on them (input=get, output=put), the name of
reply-to queue is stored to the new variable `replyQueueName` to stores the response message on the right queue.

```java
66 replyQueueName = aReplyQueue;
```

As next operation, the queues have to be opened. For placing messages on the request queue, the open option
is set to `MQC.MQOO_OUTPUT` (line 73). This option is used for the method `accessQueue` of the MQSeries sup-
plied class `MQQueueManager` (line 77). The first parameter `aRequestQueue` assigns the name of the request
queue and the second parameter `openOptions` assigns the open options for the request queue. The last three para-
meters have been set to `null`, they are assigned to the default values: `null`, if it is the queue manager to which
`MQQueueManager` is actually connected to; `null`, if a model queue is not used; and `null`, if the default user ID is
used.

```java
73 openOptions = MQC.MQOO_OUTPUT;
... 77 requestQueue = qMgr.accessQueue(aRequestQueue, openOptions, null, null, null);
```

For placing the response messages onto the reply-to queue, the open option for the queue is set to `MQC.M-
QOO_INPUT_AS_Q_DEF` (line 87). As same as for the request queue, the method `accessQueue` assigns the
name of the reply-to queue, the open options, and the default values for the queue manager name, the model
queue, and the user ID (line 88).

```java
87 openOptions = MQC.MQOO_INPUT_AS_Q_DEF;
88 replyQueue = qMgr.accessQueue(aReplyQueue, openOptions, null, null, null);
```
In case, the reply-to queue and the request queue have the same name (that is possible), the open options have been set as follows.

```java
70 openOptions = MQC.MQOO_INPUT_AS_Q_DEF | MQC.MQOO_OUTPUT;
```

### 5.6.1.3 Creating the request message and send it

The request message object is created within the `send`-method of the JAVA-class `MQCommunicator` (line 109). The request message object `sendMessage` is initialised (line 113) before it is created as an instance of the MQSeries supplied JAVA-class `MQMessage` (line 117). The format of the message (`sendMessage.format`) is set to the queue manager built-in format `MQC.MQFMT_STRING` to indicate the nature of the data in the message (line 126). In line 134 is specified the type of the message (`sendMessage.messageType`); the value `MQC.MQMT_REQUEST` sets the message to a request message. Both – the message format and the message type – build the message descriptor. The previously stored name of the reply-to queue `replyQueueName` specifies that queue on which the response messages are stored (line 137):

```java
109 public MQMessage send(String customerNumber) {
    ...
    113    MQMessage sendMessage = null;
    ...
    117    sendMessage = new MQMessage();
    ...
    126    sendMessage.format = MQC.MQFMT_STRING;
    ...
    134    sendMessage.messageType = MQC.MQMT_REQUEST;
    ...
    137    sendMessage.replyToQueueName = replyQueueName;
```

As next, a string is defined consisting of the contents of the request COMMAREA. It is specified by the strings `bufferFront`, the `customerNumber`, and the `bufferEnd` (line 140). This data string is always 408 characters long as same as the request COMMAREA measures. It consists of a 25 bytes `bufferFront`, a 5 digit `customerNumber`, and at least, a 378 bytes `bufferEnd`. The strings `bufferFront` and `bufferEnd` are defined in the MQSeries communication logic class `MQCommunicator`. The `customerNumber` is passed as a string from the external JAVA-class `MQClient` defined within the file `MQClient.java`.

Within the string `bufferFront` there are listed three fixed values and some blank characters – that is the header of the COMMAREA. The first 8 characters specifies the CICS program which is to be called, in that case NACT02 (line 24). Additionally, 2 blank characters have to be inserted behind the name to get an 8 bytes program name (“NACT02__”). The characters `VIA` refer to the version and is the “eyecatcher” to create the correct COMMAREA (line 25). Because the account is to be read, the request type `E` has also to be added to the `bufferFront` (line 26). The next 13 characters leave free to contain response and reason codes and an EIBFN code (lines 27-29). The string `customerNumber` (position 26-30 of the record) contains the account number to request the
customer account. Position 31 through 408 of the record is filled with empty strings to store the account record data on response (lines 30-37). With the method writeString of the MQMessage-instance sendMessage, the buffer/data is added to the message (line 141):

```java
24 bufferFront = "NACT02" + 
25 "V1A" +   
26 "E" +    
27 " " +    
28 " " +    
29 " " +    
30 bufferEnd = " " +      
31 " " +    
32 " " +    
33 " " +    
34 " " +    
35 " " +    
36 " " +    
37 " ";
140 String buffer = new String(bufferFront + customerNumber + bufferEnd);
141 sendMessage.writeString(buffer);
```

For a complete description on how the buffer is used to transmit data, please refer to the chapter 4.3.5.4 “COMMAREA – a scratchpad facility” on page 60.

After the message has been created, consisting of the message descriptor and the request data, it is put onto the request queue using the put-method of the MQSeries supplied JAVA-class MQQueue using default options set within an instance of the MQSeries object MQPutMessageOptions (line 144 & 148). When the message is placed onto the queue, a note is displayed (line 154). To retrieve the correct response message after requesting NACT02, a new MQSeries message object named storedMessage is created in which the message identifier is stored as a correlation identifier (line 159, 162 & 168). This object is used as a reference for the response message. If an error occurs on creating or sending the response message, some exception routines are specified (lines 170-182, please refer to Listing 42).

```java
144 MQPutMessageOptions pmo = new MQPutMessageOptions();
148 requestQueue.put(sendMessage, pmo);
154 print("Message placed on queue");
159 storedMessage = new MQMessage();
162 storedMessage.correlationId = sendMessage.messageId;
168 return storedMessage;
```
5.6.1.4 Receiving the response message

When NACT02 has answered on the request and a message is sent back from the MQSeries server on OS/390 the JAVA application has to receive the response message. For this response message, the MQSeries message object is already supplied as the parameter replyMessage in the instance storedMessage (line 187):

```
187 public String receive(MQMessage replyMessage) {
    MQGetMessageOptions gmo = new MQGetMessageOptions();
    gmo.options = MQC.MQGMO_WAIT | MQC.MQGMO_CONVERT;
    gmo.waitInterval = 60000;
    replyQueue.get(replyMessage, gmo);
    int msglen = replyMessage.getMessageLength();
    String msgText = replyMessage.readString(msglen);
    return msgText;
}
```

The message is received with the get-method of the MQSeries supplied JAVA-class MQQueue using options set within an instance of the MQSeries object MQGetMessageOptions (line 190). The default options are not used. If the response message is not yet transmitted onto the reply-to queue, the application waits for it using the option MQC.MQGMO_WAIT. The received chars are converted to the right char set using the option MQC.MQGMO_CONVERT (line 195). The time period to wait for the response message is set to 60 seconds using the integer waitInterval (line 198). After the options are set, the get-method is used to receive the account record data (line 211). Defining the integer msglen provides the length of the transmitted message (line 217). This length is used to read the message data using the method readString (line 220 & 225). The data is stored within the string msgText. Receiving the response message is also intercepted by exceptions; however, for this code fragment refer to Listing 42.

5.6.1.5 Finalising the connection to the WINDOWS2000-client queue manager

The finalise-method closes the request and the reply-to queue after the response message is received. Closing the queues is done with the close-method of the MQSeries supplied JAVA-class MQQueue (line 246 & 249). At the end, the connection to the queue manager is terminated with the disconnect-method of the MQSeries supplied JAVA-class MQQueueManager (line 252):

```
243 public void finalise() {
    requestQueue.close();
    if (requestQueue != replyQueue) replyQueue.close();
    qMgr.disconnect();
}
```
5.6.2 Coding the presentation logic

The JAVA-file MQClient.java contains the presentation logic. It requests an customer account record by passing the typed-in account number to the MQSeries communication logic class. When the request is successful, the account record data is displayed within the JAVA-GUI on the WINDOWS2000 screen. The JAVA GUI uses the JAVA SWING classes.

The JAVA application is started when following command, including the required parameters, is executed on the command line of a MS-DOS console:

```
java MQClient <<WINDOWS-IP-ADDRESS>> <<server connection>> +
<<queue manager>> <<remote queue definition>> <<reply-to queue>>
```

This opens an application screen as shown in Figure 51. With the command parameters and the constructor method of the JAVA class MQClient, an instance of the JAVA class MQCommunicator named MQComms is created (line 141). For a description of and using the parameters refer to chapter 5.6.1 “Coding the MQSeries communication logic ”, page 132.

The input screen is build with the JFrame structure consisting of a menu bar created with the JAVA class JMenuBar using the createClientMenuBar-method (lines 149 & 366-403, Figure 51, No.1) and of two panels

![Figure 51: Input screen of the MQSeries JAVA-application](image-url)
Generation of a Java front end for a standalone CICS application accessed through MQSeries & Securing CICS with RACF

created as an instance of the JAVA-class JPanel. One panel, that holds the field to input the account number and a button to fetch it, is created using the method createFetchPanel (lines 154 & 440-469, Figure 51, No.2). The other panel is created as a scroll panel using the method createRecordPanel and displays the account record data (lines 150, 155 & 472-681, Figure 51, No.3). Both panels are adjusted using the standard layout manager BorderLayout (lines 153-155):

```java
MQComms = new MQCommunicator(hostname, channel, qManager, requestQueue, + replyQueue);
...
setJMenuBar(createClientMenuBar());
JScrollPane scrollpane = new JScrollPane(createRecordPanel());
...
getContentPane().setLayout(new BorderLayout());
getContentPane().add("North", createFetchPanel());
getContentPane().add("Center", scrollpane);
... protected JMenuBar createClientMenuBar()
...
protected JPanel createFetchPanel()
...
protected JPanel createRecordPanel()
```

The account number field is a combination box used for input and for choosing an account number from a list. It is built as an instance of JComboBox (lines 447-449, Figure 51, No.2a). The entry can be fetched by a clickable button created as an instance of JButton (lines 452-461, Figure 51, No.2b). It can also be fetched by a menu bar item created as an instance of JMenuItem within the method createClientMenuBar (lines 366 & 378-382). Both fetch methods have to be activated to get a response; this is handled by the method createEventHandlers (lines 175 & 339-363). With the method enableDisableListItems the fetch button and the fetch menu bar item are set enabled (lines 406, 408 & 410). Getting the response using the fetch button resp. the fetch menu bar item is done with the get-methods in line 415 resp. 420:

```java
createEventHandlers();
...
protected void createEventHandlers()
...
protected JMenuBar createClientMenuBar()
...
menuFetch = new JMenuItem("Fetch Record");
...
public void enableDisableListItems(boolean option)
...
getMenuItemFetch().setEnabled(option);
...
getButtonFetch().setEnabled(option);
...
protected JButton getButtonFetch()
...
protected JMenuItem getMenuItemFetch()
...
accountNumberField = new JComboBox();
...
buttonFetch = new JButton(new ImageIcon("images/up.gif"));
```
Whether, if one of both fetch methods is used, the method `clickedFetch` is always activated to retrieve the account number from the account number field when clicked on the button or used the menu bar item (line 190-282). This method gets the account number as a string using the method `getAccountNumber` (lines 684-715). Using the method `getAccountNumberField` retrieves the number from the input field whereas `getSelectedltem` retrieves the number from the pull down list of the combination box (lines 435 & 687). The number is then passed to `MQComms` using its `send`-method (lines 207 referring to line 192). The `clickedFetch`-method also receives the requested data from `MQComms` using `receive-method` of `MQComms` (line 221) and validates whether the enquiry has been successful or not (lines 238-277). If the enquiry has been successful (lines 238-244), the methods `displayRecord` and `setAccountHistory` are executed to display the account record data (lines 718-742 & 745-780). Both methods calls the two external application classes `AccountRecord` and `AccountHistory` to extract the received account record data into the correct strings. If the response is returned with an error code, some information messages are displayed and the fields are cleared using the `clearDisplay`-method, that is called for example from line 202 (lines 783-811):

```java
190 public void clickedFetch()
192    String requested = getAccountNumber();
202    clearDisplay();
207    currentRequest = MQComms.send(requested);
221    messageReturned = MQComms.receive(currentRequest);
238    if(CICSresponseCode.equals("0000")){
240       customerDetails = messageReturned.substring(25,408);
241       AccountRecord retrieved=new AccountRecord(customerDetails);
242       displayRecord(retrieved);
243       return;
244    }
435 protected JComboBox getAccountNumberField()
684 public String getAccountNumber()
687    String accountNumber = (String)getAccountNumberField().getSelectedItem();
718 public void displayRecord(AccountRecord record)
741    setAccountHistory(record);
745 private void setAccountHistory(AccountRecord record)
783 public void clearDisplay()
```

The panel created within the `createRecordPanel`-method consists of 4 more panels arranged top left, top right, bottom left, and bottom right using the `GridLayout` with 2 rows and 2 columns (lines 478-494 & 669-677). The top left panel displays personal information about the customer (Figure 51, No.3a). The fields are arranged using the `GridLayout` again. There are used 9 rows and 2 columns (lines 587-610). In the first row is written the header for the panel using `JLabel` (line 551 & 552). The next 8 rows contains the output fields with an associated
label. In the first column are displayed the names using instances of \texttt{JLabel}, too (lines 553-560). The non-editable output fields besides the names are set in the second column using instances of the external application class \texttt{DisplayField} (497-504 & 93-100). All fields are added to the top left panel using the \texttt{add}-method (lines 593-610). The bottom left panel uses also the \texttt{GridLayout} consisting of 9 rows and 2 columns (Figure 51, No.3b). This panel is used to display the credit card details. The objects are defined as same as for the top left panel (lines 105-114, 507-516 & 562-571). The top right panel displays a scrollable table of the account history created as an instance of \texttt{JTable} and \texttt{JScrollPane} that calls another external application class \texttt{AccountHistoryTableModel} (lines 117-119, 534-548 & 651, Figure 51, No.3c). Furthermore, a title is also added with a \texttt{JLabel} instance (lines 573 & 650). The last panel, the bottom right panel displays user who are authorised to use the customer's credit cards, too (Figure 51, No.3d). This panel is once more arranged by a \texttt{GridLayout} consisting of 9 rows and only 1 column (lines 656-659). The labels and output fields are defined as same as the fields for the personal information (lines 101-104, 524-527, 575-576 & 661-666):

478 topleft = new JPanel(), topright = new JPanel(), // Defining the Panels
479 bottomleft = new JPanel(), bottomright = new JPanel();
...
497 titleField = new DisplayField(50, background);
...
534 dataModel = new AccountHistoryTableModel();
536 JTable tableView = new JTable(dataModel);
...
547 JScrollPane scrollpane = new JScrollPane(tableView);
548 scrollpane.setPreferredSize(new Dimension(tableWidth, tableHeight));
...
551 JLabel personalLabel = new JLabel("PERSONAL ", JLabel.RIGHT),
552 personal1Label = new JLabel("INFORMATION"),
553 titleLabel = new JLabel("Title: ", JLabel.RIGHT),
...
562 creditCardLabel = new JLabel("CREDIT ", JLabel.RIGHT),
563 creditCard1Label = new JLabel("CARD DETAILS"),
...
573 acctHistoryLabel = new JLabel("ACCOUNT HISTORY"),
...
575 emptyLabel = new JLabel(""),
576 othersLabel = new JLabel("Additional Authorised Card User:",JLabel.LEFT);
...
587GridLayout gleft = new GridLayout(9, 2); // Layout topleft Panel
...
In case, errors occur, they are intercepted by exceptions and the user are informed by popping up a message window. Some messages are displayed on the MS-DOS console such as MQSeries status messages or CICS response and reason codes.

At the end of the application class `MQClient`, it is instantiated by the `main`-method (lines 814-828). The class ends in line 828.

```java
814 public static void main (String[] args) throws Exception 
  ... 
822    new MQClient(args);
```
5.7 Connecting both queue managers

5.7.1 Checking the status of the queue managers and activate services

Before connecting both queue managers, a query has to be started to check whether all required services have been started successfully – the subsystem/command server, the queue manager, the channel initiator, and the TCP/IP listener. On the OS/390-server, an entry in the DA-panel shows if the MQSeries subsystem and the queue manager have been started. The jobid entry MQAIMSTR confirms that the MQSeries subsystem is up and the queue manager runs (Figure 52). Reaching the DA-panel is described in Appendix B.1 on page 231. When the jobid MQA1CHIN is displayed in the same panel, the channel initiator is activated, too (Figure 52). On the SDSF command line following command can be typed-in to show whether the channel initiator and the TCP/IP listener are started:

```bash
!/MQA1 DISPLAY DQM
```

The output displays that both have been started (Figure 53, next page). This message can also be displayed using the MQSeries panels. On the main screen “MAIN MENU” choose DISPLAY (shortcut 2) and SYSTEM in the field Object type. Characters are not required to input in the field Name. On the next panel “Display a System Function” choose the option “1” for Distributed queuing and press the enter key. A similar output as on the DA-panel is displayed (Figure 54, next page).

Figure 52: Displaying the active users on OS/390

12 Suggestion: Use the command line of the syslog panel.
On the WINDOWS2000-client queue manager it is only required to start the MMC Snap-In “MQSeries Services” and all services will start automatically. When choosing the queue manager TBUSSE.NACT the services with their status will displayed (Figure 46, page 124).

Now, both queue managers can be connected.

Figure 53: Displaying the started MQSeries Services – Channel initiator and TCP/IP listener – 01

Figure 54: Displaying the started MQSeries Services – Channel initiator and TCP/IP listener – 02
5.7.2 Connecting the WINDOWS2000-client queue manager with the OS/390-server queue manager

After the “MQSeries Services” MMC has been confirmed that all the required services for the WINDOWS2000-client queue manager TBUSSE.NACT are activated it is switched to the MQSeries Explorer. A right mouse click on the channel sender TBUSSE.NACT.WIN.OS pops up a menu where the option Start has been chosen to start the channel sender (Figure 55). Be sure, that the IP-address has been set correctly (JEDI’s one is always 139.18.4.97). After the Start-button is clicked, a message appears and informs that The request to start the channel was accepted (see Figure 56, next page). That does not imply that the channel sender is successfully connected to the OS/390-server queue manager. Only the green turned up arrow in front of the channel sender name shows whether the start command has been successfully executed or not (Figure 57, next page). If not, a red turned down arrow appears and the error has to be found, for example looking at the log for the queue manager.

The channel receiver is set active automatically when the WINDOWS2000-client queue manager is connected to the OS/390-server.

Figure 55: Starting the channel sender TBUSSE.NACT.WIN.OS
Figure 56: Message that the request to start the channel was accepted

Figure 57: Channel was successfully connected to the server queue manager
5.7.3 Connecting the OS/390-server queue manager with the WINDOWS2000-client queue manager

After the channel sender of the WINDOWS2000 queue manager is started, the channel sender TBUSSE.NACT.OS.WIN of the OS/390-server queue manager has to be started and connected with WINDOWS2000. The channel sender is started from the MQSeries panel “List Channels”. This panel can be reached from the MQSeries MAIN MENU. On the field action Display (shortcut 2), on the field Object type the string CHANNEL, and on the filed Name the string T* have been entered. This panel is usually used to check the status of all channel objects of MQA1. As first, it can be seen, that the channel receiver TBUSSE.NACT.OS.WIN has been activated; in the column Status stands RUN. To start the channel sender the number 6 (shortcut for Start) is entered in front of the channel name (Figure 58, next page). If the IP-address of the WINDOWS2000-client has been changed during this set up, it has to be changed into the right one. If number 3 (shortcut for Alter) is keyed in, the channel can be altered. After modifying is done it is turned back to the panel “List Channels” (pressing PF12 instead of PF3) and the channel sender can be started. The next panel “Start a Channel” is only confirmed and immediately a message is shown that the channel sender is started:

```
CSQ9022I !MQA1 CSQXCRPS 'START CHANNEL' NORMAL COMPLETION
```

Pressing PF12 leads back to the panel “List Channels” that displays the status of the channel objects. But, the status of the channel sender has not been changed into RUN. When pressing PF5, the screen is refreshed and the status of the channel changes into “RUN” (Figure 59, next page). The WINDOWS2000-client queue manager also displays that the response channel is activated. On WINDOWS2000 the Refresh Key (F5) has to be used to switch the status of the channel receiver. The green turned up arrow indicates that the receiver has been started (Figure 60, page 148).

After both queue managers are connected, the MQSeries CICS application MQNACT, as next and last step, are started.
### Figure 58: Connecting the channel sender with the client queue manager

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBUSSE.NACT.OS.WIN</td>
<td>CHLSENDER</td>
<td>STOP</td>
</tr>
<tr>
<td>TBUSSE.NACT.WIN.OS</td>
<td>CHLRECEIVER</td>
<td>RUN</td>
</tr>
</tbody>
</table>

### Figure 59: Channel status after pressed the refresh key

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBUSSE.NACT.OS.WIN</td>
<td>CHLSENDER</td>
<td>RUN</td>
</tr>
<tr>
<td>TBUSSE.NACT.WIN.OS</td>
<td>CHLRECEIVER</td>
<td>RUN</td>
</tr>
</tbody>
</table>

| Command ==> __________________________________________________________________ |
| F1=Help                      | F2=Split   | F3=Exit | F5=Refresh | F7=Bkwd | F8=Fwd |
| F9=Swap                      | F10=Messages| F11=Status| F12=Cancel |        |       |
Figure 60: Channel receiver was activated after successful call from the server queue manager
5.8 Starting the JAVA application

After both – the request and response channels – have been created, which implies the connection of both queue managers, the MQSeries CICS application needs only to be started using this command on the DOS-console:

```
java MQClient <<WINDOWS-IP-ADDRESS>> <<server connection>> + <<queue manager>> <<remote queue definition>> <<reply-to queue>>
```

Example: java MQClient 80.135.228.155 TBUSSE.NACT.CLIENT + TBUSSE.NACT TBUSSE.NACT.REMOTEQ TBUSSE.NACT.REPLYQ

This starts the JAVA interpreter to execute the code specified in the main-method of MQClient.java. When all parameters are transmitted, the connection to the WINDOWS2000-client queue manager is established and the screen “The Royal Bank of KanDoIT – Account Enquiry Client” appears. When a request on the account number 11100 is done, an application window like the Figure 61 appears. Synchronously, MQSeries status messages appear on the MS-DOS console (Figure 62, next page).

![Figure 61: The Royal Bank of KanDoIT – Account Enquiry Client](image-url)
Figure 62: The output messages on the DOS-console
5.9 Terminating the connection between the MQSeries servers

In case the connection between the both queue managers is terminated after the MQSeries CICS application MQNACT was closed, both channel senders need to be stopped. For stopping the channel sender of the OS/390-server queue manager switch, the MQSeries “List Channels” panel is to be opened on OS/390. Choosing option number 7 and filling it into the input field before the name of the channel sender (Figure 63) and pressing Enter opens the “Stop a Channel” panel. In this panel choose one of the two stop modes, press Enter and an information message confirms the termination. Returning to the “List Channels” panel and pressing PF5 refreshes the panel and the status of the channel sender switches to “STOP” (Figure 64, next page).

On the WINDOWS2000 client queue manager the channel sender is stopped when a right mouse click on this object is performed and the option “STOP” is chosen (Figure 65, next page). As next, a “Stop a Channel” panels opens and clicking on “Yes” stops the channel. In case, a force interrupt is wanted, choose the associated option and click on “Yes” (Figure 66, page 153). An information message appears and the green turned up arrow of the channel sender switches into a red one. The connection between both queue managers is now terminated.

If wanted, both queue managers can also be stopped, but consider, that both queue managers run on a MQSeries server. How to stop the queue manager on OS/390 is described in chapter 5.4.1 “The OS/390-server queue manager MQA1” on page 96. The WINDOWS2000-client queue manager is stopped with a right mouse click on the queue manager name and choosing “STOP” (Figure 67, page 153). The “End Queue Manager” panel opens to choose a controlled or immediate shutdown (Figure 68, page 154). Again, the green turned up arrow of the WINDOWS2000-client queue manager switches into a red one.

---

**List Channels**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBUSSE.NACT.OS.WIN</td>
<td>CHLSENDER</td>
<td>RUN</td>
</tr>
<tr>
<td>TBUSSE.NACT.WIN.OS</td>
<td>CHLRECEIVER</td>
<td>RUN</td>
</tr>
</tbody>
</table>

---

Figure 63: List Channels panel – Stopping a channel sender of the OS/390-server queue manager
Generation of a Java front end for a standalone CICS application accessed through MQSeries & Securing CICS with RACF

List Channels

Type action codes. Then press Enter.
1=Display 2=Define like 3=Alter 4=Delete 5=Perform
6=Start 7=Stop

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBUSSE.NACT.OS.WIN</td>
<td>CHLSENDER</td>
<td>STOP</td>
</tr>
<tr>
<td>TBUSSE.NACT.WIN.OS</td>
<td>CHLRECEIVER</td>
<td>RUN</td>
</tr>
</tbody>
</table>

******** End of list ********

Command ===> __________________________________________________________________
F1=Help      F2=Split     F3=Exit      F5=Refresh   F7=Bkwd      F8=Fwd
F9=Swap     F10=Messages F11=Status   F12=Cancel

Figure 64: List Channels panel – Channel sender is stopped

Figure 65: Stopping the channel sender on the WINDOWS2000 client queue manager
Figure 66: Stopping the channel sender on the WINDOWS2000 client queue manager

Figure 67: Stopping the channel sender on the WINDOWS2000 client queue manager
Figure 68: Stopping the channel sender on the WINDOWS2000 client queue manager
5.10 Common MQSeries problems indicated due to this thesis

5.10.1 Ghost channel connections on the OS/390-server queue manager

When a network outage occurs the channel receiver is often unaware of this and remains running even though the channel sender is retrying. When connected both queue managers again, the channel sender attempts to start a new receiver instance to OS/390 and sets its status, for example, to "2 RUN" because it does not detected the communication failure (Figure 69). That means, the more active connections are lost and reactivated the more channel receivers are activated when connected again. MQSeries believes that there has been an invalid attempt to start multiple instances of the same channel receiver, from the same channel sender and location, and accordingly treats this as an error, and fails the request. However, the communication behaves in a strange manner because sometimes the request is well done whereas in other situations the request fails.

```
<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TBUSSE.NACT.OS.WIN</td>
<td>CHLSENDER</td>
</tr>
<tr>
<td>2</td>
<td>TBUSSE.NACT.WIN.OS</td>
<td>CHLRECEIVER</td>
</tr>
</tbody>
</table>
```

Figure 69: List Channels panel – Ghost connection

This misfeature is known by IBM and is remedied by a Program Temporary Fix (ptf). Every time a customer reports a problem, IBM creates a report called Authorized Program Analysis Report (APAR). The APAR PQ34355 documents a requirement for the channel receivers of MQSeries in OS/390 to behave differently after a network outage. Within this document it is suggested to download the PTF UQ40939 to install two new functions to MQSeries – AdoptMCA and AdoptCH. Within AdoptMCA an orphaned instance of a channel connection can be automatically stopped. This PTF is also included in the IBM Corrective Service Package 3907547614.

---

13 Information about the APAR PQ34355 to download the PTF UQ40939 can be found at: http://www-306.ibm.com/software/integration/mqfamily/support/apars/mvs210cl.html
or browse the CD (includes the PTF). additions/OS390/MQSeries_v2.1/par/PQ34355
14 The IBM Corrective Service Package COER390754765 can be found on the CD at:
a full repository of all required updates until January 2003. However, this package is no more available at IBM, but included on the CD. Unfortunately, this update has not been installed on OS/390 because this must be done with the SMP/E product. However, this was not part of this master thesis. Managing SMP/E should be handled in future.

When such a failure occurs, simply terminate the channel initiator and restart it again using the MQSeries panels or the command line of the SD-panels as described in chapter 5.4.1 “The OS/390-server queue manager MQA1” on page 96.

5.10.2 Resetting channel in-doubt status – The message sequence error

Sometimes, the channel sender cannot communicate with the channel receiver due to a message sequence failure. In that case, MQSeries fails to deliver the message and sets the transmission queue to GET disabled. The OS/390 syslog and the MQSeries log on OS/390 notes following message due to this error:

```
+CSQX526E !MQA1 CSQXRCTL Message sequence error for TBUSSE.NACT.OS.WIN, sent=1 expected=904
+CSQX506E !MQA1 CSQXRCTL Message receipt confirmation not received for TBUSSE.NACT.OS.WIN
```

On the WINDOWS2000-client such MQSeries messages are recorded in the log AMQERR01.LOG usually stored in the folder “\MQSeries_home\Qmgrs\TBUSSE!NACT\errors\”, where MQSeries_home is the high-level qualifier of the folder where MQSeries has been installed, and where TBUSSE!NACT is the name of the queue manager. The message written in the log due to the message sequence failure reads as:

```
03/18/2004 11:25:33
AMQ9526: Message sequence number error for channel 'TBUSSE.NACT.OS.WIN'.
```

This misfeature can be remedied when the sequence number of the channels sender is reset. On OS/390 open the MQSeries “List Channels” panel and choose number 5 to execute the Perform operation (Figure 70, next page). After the Enter was pressed, the “Perform a Channel Function” appears. Number 1 has to be chosen to reset the message sequence number (Figure 71, next page).

The channel sender on the WINDOWS2000-client queue manager is reset as it is shown in (Figure 72, page 158).
### List Channels

Type action codes. Then press Enter.

1 = Display  2 = Define like  3 = Alter  4 = Delete  5 = Perform
6 = Start  7 = Stop

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBUSSE.NACT.OS.WIN</td>
<td>CHLSENDER</td>
<td>RUN</td>
</tr>
<tr>
<td>TBUSSE.NACT.WIN.OS</td>
<td>CHLRECEIVER</td>
<td>RUN</td>
</tr>
</tbody>
</table>

******** End of list ********

Figure 70: List Channels panel – List Channels panel – Perform a function

### Perform a Channel Function

Select function type, complete fields, then press Enter.

Function type . . . . . . . . . 1
1. Reset sequence number
2. Ping
3. Resolve with commit
4. Resolve with backout

Channel name . . . . . . . : TBUSSE.NACT.OS.WIN
Channel type . . . . . . . : CHLSENDER
Description . . . . . . . : CHANNEL FOR THE XMITQ TO SEND THE RESPONSE MESSAGES

Reset
Sequence number . . . . . 1  1 - 999999999

Ping
Data length . . . . . . . 16  16 - 32768

Command ===> __________________________________________________________________

Figure 71: List Channels panel – Perform a Channel Function panel – Reset message sequence number
Generation of a Java front end for a standalone CICS application accessed through MQSeries & Securing CICS with RACF

Figure 72: MQSeries for WINDOWS2000 – Reset the message sequence number
6 SECURING CICS WITH RACF

6.1 Introduction

During the installation of some CICS objects used for the two business applications NACT and MQNACT it is necessary to access the CICS region. Unfortunately, the CICS subsystem on the JEDI OS/390-server has not been protected by a security manager at this time. However, with the rise of more and more users who have to use this transaction monitor on the JEDI OS/390-server, securing the access to it is a really important feature. Without securing, each user could access CICS without logging on to the system, shut down the subsystem, change CICS-specific options, and change, discard, or delete all resources defined to the subsystem. All things considered, the user would have the full control of CICS. Since, this is intolerable access restriction to any CICS region is a must have. This part of the master thesis describes how to establish a standard CICS security management on the JEDI OS/390-server after CICS has been installed and configured. When the CICS security is installed, any user may log on to CICS and may only execute allowed CICS transactions and CICS commands on behalf of the associated security profile. Any access to unauthorised data is forbidden and recorded to a log journal, too.

CICS itself has no own security mechanism. Controlling, restricting, revoking, and resuming the access to any subsystem on an OS/390-server is done by the OS/390 security mechanism System Authorisation Facility (SAF). All incoming authorisation requests are sent from SAF to an External Security Manager (ESM), such as IBM’s product RACF. This is the IBM security management product for OS/390 and VM. In 1976, the first release of RACF for MVS and VM has been shipped as a stand-alone version. Since march 1996, RACF is included as part of the OS/390 Security Server, later known as SecureWay Security Server for OS/390. This platform includes RACF, the DCE Security Server, OS/390 Firewall Technologies, and the LDAP Server. The JEDI OS/390-server runs RACF Version 2 Release 615.

RACF is running like a CICS subsystem or an MQSeries queue manager in its own address space resp. region in the OS/390 kernel. Any security decision is routed from the resource manager (e.g. CICS) with the help of the system service SAF to the RACF router to invoke the correct RACF function calling the RACF database (Figure 73, next page). Hence, SAF acts primarily as a router facility (also called the OS/390 router). The resource manager gets a message back from RACF and due to this message it makes the decision. This security processing is controlled by the RACROUTE macro (Figure 73). The OS/390 router is present on all OS/390-systems even if RACF is not installed. In that case, an installation-supplied security-processing exit (equivalent to the OS/390

15 Hint: There is recorded the identifier “HBF226” in the OS/390-syslog during an IPL. This refers to the RACF Version 2 Release 6. For name conventions of these identifiers belonging to RACF versions refer to [HEN03].
router) can call another ESM (Figure 73). Such well known alternatives to RACF are Computer Associates CA-ACF2 for OS/390 (now: eTrust ACF2 Security for z/OS and OS/390), and CA-Top Secret for OS/390 (now: eTrust CA-Top Secret Security for z/OS and OS/390). In the latest operating system z/OS, IBM has excluded the RACROUTE macro to replace it by providing callable services written in the most common programming languages. Further descriptions of the security concept of the Operating System/390 can be found in [GUS01], [HK-S04], and [RSG03].

The next sections gives a short explanation of the RACF principles and how to invoke its services to show that securing CICS relies on a number of facilities provided by RACF. Securing a resource manager like a CICS region by RACF can be done in phases to have a up-to-date secured system at the required level. Firstly, the **CICS Region User ID (CRU)** is set up to allow the CICS subsystem an access to other subsystems on the OS/390-server. The **Default CICS User ID (DCU)** is defined to give, for example, all users a simple access to the CICS region as same as giving applications access to the CICS region. For programs that are executed during the CICS system initialisation an own user ID called the **PLTPI User ID (PLTPIU)** has been created. Further and as an important part, it is described how to define security profiles for the resources of the CICS region to give each CICS user an appropriate access to them. Last but not least, it is described how the CICS login terminal accepts not only the upper case characters.

![Figure 73: System Authorisation Facility (SAF)](image)
6.2 RACF Topics

6.2.1 RACF mechanisms

The approach of RACF to data security is to provide an effective user verification, resource authorisation, and logging capabilities. Each component of the operating system and users like system administrators can invoke the services of RACF.

The concept of the user accountability is one of the prime security objectives RACF supports. IBM uses the word “user” in context to human users who work with the system through a terminal and in context to components which access RACF-protected resources in the same way as a user. Such a component could be a subsystem running on OS/390 like CICS or IMS, a machine like a printer which executes an output, or an application which reads, changes, or deletes data. Both the human user and the system component have an assigned user identification (user ID) and usually a password to access the called resources. Each user ID may be created to RACF only once. With it the user can access different subsystems, for example TSO/E or CICS, or a subsystem can access another subsystem. RACF stores the access information about the permission a user has to a demanded subsystem.

To simplify the maintenance of the system access, users are organised very flexible in groups. Each user must be connected to minimal one group and can be connected to any number of groups. The authority assigned to the group can be used to give one or more permissions for all users defined in the group and to supervise the user activity. Such a RACF group could be pointed for example to a department, project, number of applications, or to administrators, data controllers, trainees, or secretaries, and so on. There is one superior or owning group to every group – the system supplied SYS1 group. Each user or group defined to RACF is linked with the group SYS1 in any way. This relationship can be sketched very good in a tree diagram.

For each user and group RACF builds profiles and stores them in its own database. These profiles consist of one or more segments. The first or basic segment is always the RACF segment to both the user and group profile. It contains RACF specific user resp. group information. The RACF segment is subdivided into a few divisions. This could be in a user profile the user identification, the owner of the user profile, user attributes, the groups associated with the user, or additional installed security classifications. The RACF segment of the group profile stores information about the owner of the group profile, the superior group, subgroups if exist, and connected users.

User attributes are one of the importing RACF segment information in a user profile. It is differentiated with respect to user attributes and to group-related user attributes. A user attribute applies all of the time and is specified at the system level. In contrast to that, a group-related user attribute applies to a specified group or groups and is specified at the group level. User attributes can be used to administrate RACF centralised or decentralised. Centralised administration refers to the resources and functions controlled by users system-wide. Such a user has for example the user attribute SPECIAL. With it the user can modify all entries in the RACF database and may
perform all RACF functions, except auditing-related commands, on the scope of the system. That is why it is also called the system-SPECIAL user attribute. Contrary, the decentralised administration refers to all users who have a group-related user attribute. They can manage the authority in the scope of the groups to which they are connected. For instance, one user has the group-SPECIAL user attribute within her/his connected groups she/he is able to use the same RACF commands a user with the system-SPECIAL user attribute can use. User attributes that also can be assigned to the system or group level are the AUDITOR, OPERATIONS, ADSP and REVOKE attributes. In contrast to those user attributes the CLAUTH user attribute is not assigned to the group level, it is assigned only to users who defines and modifies profiles in general resource classes. The administrator who wants to enable CICS security to a CICS region must have the system-SPECIAL user attribute.

Beside the basic RACF segment there could be defined some more segments to a user/group profile. They can contain user resp. group information for another secured and activated subsystem on the OS/390-server. For a user and for a group there can be defined same segments, but the user profile can obtain a few more segments. A segment created to both is the OMVS segment. It specifies information about the OS/390 UNIX subsystem. The CICS segment, containing CICS terminal user data, is only defined to the user profile. It is not necessary to define an appendant segment in a profile to use a resource. RACF will take the default built-in system values. If required, in case of future demands, a non-activated segment can be added to a profile later.

Resource authorisation provides access control to the OS/390-components and their associated data. Access control is the process of determining who – the user – has access to what – the protected resource. At the end of a security request sent to the resource manager an access control decision is delivered back. Immediately, the manager allows or denies the access to the demanded resource. Each access control decision depends on transmitted factors to determine which access to the resource is demanded. Should have the user a read access to get information about the resource, or may he modify, execute, or delete resp. remove the resource? May the user access the resource on the date he wants the access?

Referring to a subsystem installed on OS/390, an access request to a resource is always sent first to a security manager, in this case RACF. The security manager identifies and authenticates the originator of the access request. It does not matter whether a system or a user-implemented resource is accessed. RACF protects such resources with resource security profiles, in which all the required information to access that resource is stored. For example, the access list of such a profile lists the users resp. groups with their authority to access the resource. Following authorisation levels are valid in an access list, specified from the highest security level to the lowest: ALTER, CONTROL, UPDATE, READ, EXECUTE, NONE (refer to Table 10 on page 166). In addition to own defined access rights the user gets the same rights the group has as long as he is connected to the group. If the user has no own access list he inherits the rights to access a resource from the connected group or groups.

Furthermore, RACF is able to log any security-related event and informs the associated user about the event. Consequently, RACF is used by security administrators, auditors, storage administrators, TSO, SDSF and CICS administrators, management and others, if the assigned access level permits the use. An administrator must have
the SPECIAL attribute, whereas the user attributes AUDITOR or OPERATIONS should only be assigned to those users who monitor RACF.

More information about the authorities required to control RACF and to work with CICS is given in the books [RAG98], [RUG98], and in the paper [GUS01].

6.2.2 RACF commands

Managing the RACF database means adding resource security profiles to and deleting these from the database, as same as modifying and listing them. These functions are started by RACF commands entered on the command line or started within special RACF panels. On the ISPF/PDF application panels the commands can be typed-in on the command panel. This panel can be reached when P. 6 is entered in the OPTION line of the CMAM. RACF commands can also be executed from the TSO/E command line everywhere where this line turns up, for example in the SDSF application. However, it is noticed, that the system prefix / must be written in front of the commands (refer also to footnote 10 on page 96). A better and common way is to build command scripts called TSO/E CLISTs16, that include the RACF commands, in particular to summarise RACF commands.

Table 8 on the next page lists the special commands to list, add, change, or delete profiles to or from the RACF database. Within the table there are also listed commands to connect resp. remove users to or from groups. An access for a user/group to data sets resp. general resources is given with the RACF command PERMIT. The RACF command PASSWORD is used to change or reset the user's password, or to change the password interval. With the command SEARCH it is possible to search the RACF database for data set and general resource profiles. Setting RACF options is done by the command SETROPTS.

The command HELP displays information about the function, syntax, and operands of RACF commands. The RACF command DISPLAY is only available for users/groups having the OPERATOR attribute. It displays information held in the signed-on-from list. Entries in the signed-on-from list possess user IDs, groups, APPLs (the names of the local Logical Units (LU) from which the user is signed on), POEs (the names of the partner LUs from which the user is signed on), and SECLABEL (security labels to search for). There are a few more not so important RACF commands, for a complete list refer to [RCR98].

However, the most important command is RVARY which activates resp. deactivates RACF databases and hence, RACF, too. The command RVARY LIST displays all currently activated RACF databases on the OS/390-server. On the JEDI OS/390-server only one active RACF database is stored in the sequential data set SYS1.RACF on the volume DAVS7A. When the command RVARY LIST is executed on the JEDI OS/390-server following information is displayed:

<table>
<thead>
<tr>
<th>ICH15013I</th>
<th>RACF DATABASE STATUS:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTIVE</td>
<td>USE NUMBER VOLUME DATASET</td>
</tr>
<tr>
<td>YES PRIM 1</td>
<td>DAVS7A SYS1.RACF</td>
</tr>
</tbody>
</table>

16 Speak: “Cee lists”.
### 6.2.3 Data set and general resource profiles

CICS resources are subdivided into general resources and data sets. Data sets can be installed and created on hard disk drives like a DASD or on tape volumes. Indeed, DASDs and tape volumes are general resources. For instance, the following resources are also general resources: load modules (programs), terminals (VTAM), application resources (such as resources for IMS, CICS, and DB2), and other installation-defined-resources. For each CICS resource there can be created a security profile protected by RACF. This resource profile is stored in a profile class which is listed in the class descriptor table (CDT). CICS resource profiles can be organised in groups to simplify the maintenance of the RACF database.

CICS resource profiles can be classified into generic and discrete profiles. Generic profiles protect several data sets resp. resources with one profile. RACF detects these profiles by substitute characters: the percent character (%) stands for exact one character, the asterisk character (*) stands for any character string in 1 qualifier, and the double asterisk character (**) is for any character string in n-qualifier. It is noticed, that Enhanced Generic Naming (EGN) has to be in effect to create generic data set profiles, for general resource profiles EGN is always in effect. **EGN** for data set profiles is activated when the RACF command **SETROPTS (SETR)** is executed:

```plaintext
SETROPTS  EGN
```

### Table 8: RACF commands

<table>
<thead>
<tr>
<th>USERS</th>
<th>GROUPS</th>
<th>DATA SET</th>
<th>GENERAL RESOURCES</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST</td>
<td>LISTUSER (LU)</td>
<td>LISTDSD (LD)</td>
<td>RLIST (RL)</td>
</tr>
<tr>
<td>ADD</td>
<td>ADDUSER (AU)</td>
<td>ADDSD (AD)</td>
<td>RDEFINE (RDEF)</td>
</tr>
<tr>
<td>CHANGE</td>
<td>ALTUSER (ALU)</td>
<td>ALTDS (ALD)</td>
<td>RALTER (RALT)</td>
</tr>
<tr>
<td>DELETE</td>
<td>DELUSER (DU)</td>
<td>DELDSD (DD)</td>
<td>RDELETE (RDEL)</td>
</tr>
<tr>
<td></td>
<td>CONNECT (CO)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>REMOVE (RE)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PASSWORD (PW)</td>
<td>SEARCH (SR)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>PERMIT (PE)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RVARY</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SETROPTS (SETR)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>HELP</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DISPLAY</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ICH15020I RVARY COMMAND HAS FINISHED PROCESSING.
Discrete profiles may not have any substitute token, they always identify the whole data set. However, such profiles should not be used because of their disadvantages in protecting each data set with an own profile. This results in reducing the number of profiles needed to protect data sets, and in reducing the size of the RACF database by using generic data set profiles.

Furthermore, RACF distinguishes also between two other types of CICS resource profiles – profiles for user data sets/single general resources, and profiles for group data sets/group of profiles. Each data set profile has to be associated with an own RACF user/group ID – the profiles for user data sets with a RACF user ID and the profiles for group data sets with a RACF group that has the same name as the data set's high-level qualifier name. All data set profiles have to be stored in the RACF profile class DATASET. There have to be created data set profiles for the CICS transaction server install data set and for each CICS region data set. Usually they are defined as generic data set profiles.

General resource profiles for CICS are managed in a set of IBM-supplied resource classes for CICS, in own defined resource classes, and in IBM-supplied RACF general resource classes affecting CICS. The default resource classes for CICS can protect several resources stored in a member resource class as same as a group of resources stored in a group resource class. Table 9 on the next page gives an overview of the IBM-supplied CICS resource class names.

There have been defined general resource profiles for CICS transactions and CICS commands for the CICS region A06C001 of the JEDI OS/390-server. For instance, several CICS transactions (e.g. the business transaction NACT) have been secured using the member resource class TCICSTRN whereas CICS transactions listed in groups (e.g. CICS system transactions) have been protected using the resource class GCICSTRN. How to enable the CICS transaction and the CICS command security and how to define profiles for their associated resource classes is described in chapter 6.4 “Securing the resources for the CICS region A06C001” on page 185.

Following IBM-supplied RACF resource classes affect CICS: APPCLU, APPL, CONSOLE, DIGTCERT, FACILITY, FIELD, LOGSTRM, OPERCMDS, PROPCNTL, PTKTDATA, RACFVARS, RACGLIST, STARTED, SUBSYSNM, SURROGAT, TERMINAL, and VTAMAPPL. How to use the class names that are formatted bold is explained in the next chapters. For example, within the resource class APPL there are defined security profiles to control the terminal user access to a CICS region (refer to chapter 6.5 “Authorising access to the CICS region" on page 205). Data set and general resource profiles consist of authority levels to decide which protected CICS resource the RACF-users resp. RACF-groups can access in which way (Table 10, next page).

<table>
<thead>
<tr>
<th>Class name</th>
<th>Type</th>
<th>Protect these resources:</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCICSTRN</td>
<td>Member</td>
<td>CICS transactions</td>
</tr>
<tr>
<td>GCICSTRN</td>
<td>Group</td>
<td></td>
</tr>
<tr>
<td>GCICSCMD</td>
<td>Member</td>
<td>CICS system commands &amp; CICS FEPI system commands</td>
</tr>
<tr>
<td>VCICSCMD</td>
<td>Group</td>
<td></td>
</tr>
</tbody>
</table>
Generation of a Java front end for a standalone CICS application accessed through MQSeries & Securing CICS with RACF

<table>
<thead>
<tr>
<th>Member</th>
<th>Group</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACICSPCT</td>
<td>Member</td>
<td>CICS transactions started with the EXEC CICS START command (known as started transactions) &amp; following CICS commands: COLLECT STATISTICS TRANSACTION DISCARD TRANSACTION INQUIRE TRANSACTION SET TRANSACTION INQUIRE REQID CANCEL</td>
</tr>
<tr>
<td>BCICSPCT</td>
<td>Group</td>
<td>CICS files</td>
</tr>
<tr>
<td>FCICSFCT</td>
<td>Member</td>
<td>CICS files</td>
</tr>
<tr>
<td>HCICSFCT</td>
<td>Group</td>
<td>CICS programs</td>
</tr>
<tr>
<td>MCICSPPT</td>
<td>Member</td>
<td>CICS programs</td>
</tr>
<tr>
<td>NCICSPPT</td>
<td>Group</td>
<td>CICS DL/I Program Specification Blocks (PSBs) for IMS</td>
</tr>
<tr>
<td>PCICSPSB</td>
<td>Member</td>
<td>CICS DL/I Program Specification Blocks (PSBs) for IMS</td>
</tr>
<tr>
<td>QCICSPSB</td>
<td>Group</td>
<td>CICS journals and log streams</td>
</tr>
<tr>
<td>JCICSJCT</td>
<td>Member</td>
<td>CICS journals and log streams</td>
</tr>
<tr>
<td>KCICSJCT</td>
<td>Group</td>
<td>CICS journals and log streams</td>
</tr>
<tr>
<td>SCICSTST</td>
<td>Member</td>
<td>CICS journals and log streams</td>
</tr>
<tr>
<td>UCICSTST</td>
<td>Group</td>
<td>CICS journals and log streams</td>
</tr>
<tr>
<td>DCICSDCT</td>
<td>Member</td>
<td>CICS transient data queues</td>
</tr>
<tr>
<td>ECICSDCT</td>
<td>Group</td>
<td>CICS transient data queues</td>
</tr>
</tbody>
</table>

Table 9: RACF commands

<table>
<thead>
<tr>
<th>Authorisation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTER</td>
<td>Full access to data set/general resource (profile is also accessible) Allowing access for other users/groups to the data set/general resource Add and delete data sets/general resources</td>
</tr>
<tr>
<td>CONTROL</td>
<td>Used only for VSAM-data protecting mechanism (only data set) User/group can modify the VSAM file</td>
</tr>
<tr>
<td>UPDATE</td>
<td>User/group can modify the data set/general resource Data set: User/group can delete PDSs User/group cannot modify a VSAM file</td>
</tr>
<tr>
<td>READ</td>
<td>User/group can read the data set/general resource, and can print and copy the data set</td>
</tr>
<tr>
<td>EXECUTE</td>
<td>User/group can execute predefined programs (only data set)</td>
</tr>
<tr>
<td>NONE</td>
<td>User/group has no access to data set/general resource</td>
</tr>
</tbody>
</table>

Table 10: Authorisation Levels

A general resource class must be activated first before the general resource profiles can be used by RACF. The RACF-command SETROPTS CLASSACT activates such classes stored in the CDT. Usually, each RACF resource class needs to be activated separately. In case, a few general resource classes have the same position number (POSIT-number) in their CDT definitions only one resource class out of them needs to be activated. For example, all the CICS general resource classes have the same POSIT-number in their CDT definitions. It is noticed, that the group resource classes cannot be activated because they are managed by the member resource class:

```
SETROPTS CLASSACT(class_name)
```
Example: SETROPTS CLASSACT(TCICSTRN)

The command *SETROPTS NOCLASSACT* deactivates any protection for the profiles of the general resource class. However, when using this command the class is not deleted:

```sql
SETROPTS NOCLASSACT(class_name)
```

Example: SETROPTS NOCLASSACT(TCICSTRN)

Although, EGN is active for general resource resp. data set profile names, their associated profile classes need also to be set up to store such generic profiles with the command *SETROPTS GENERIC* (Example 1). Each class that stores generic profiles has to be refreshed after a new generic profile has been added to the class as Example 2 shows:

```
SETROPTS GENERIC(class_name) [REFRESH]
```

Example 1: SETROPTS GENERIC(DATASET)
Example 2: SETROPTS GENERIC(DATASET) REFRESH

For performance reasons, consider activating the sharing of the general resource profiles using the RACF command *SETROPTS RACLIST* (Example 1). In that case, discrete and generic profiles are stored in main/virtual storage and RACF will not require accessing its database stored in a file when making an access decision. Each time such an in-storage profile is changed, it must be refreshed using the attribute REFRESH (Example 2). This storage method is recommended for all CICS general resource classes but not for the OS/390 DATASET class because data set profile may not stored to the main/virtual storage.

```
SETROPTS RACLIST(class_name) [REFRESH]
```

Example 1: SETROPTS RACLIST(TCICSTRN)
Example 2: SETROPTS RACLIST(TCICSTRN) REFRESH

For a small number of frequently referenced generic profiles stored in general resource profiles in the main/virtual storage can be also used the *SETROPTS GENLIST* command (Example 1). As same as for RACLIST, each time a generic profile is created, the main/virtual storage needs to be refreshed (Example 2, next page). Data set profiles also cannot use GENLIST processing.

```
SETROPTS GENLIST(class_name) [REFRESH]
```

Example 1: SETROPTS GENLIST(CCICSCMD)
Example 2: SETROPTS GENLIST(CCICSCMD) REFRESH

The RACLIST and GENLIST of all CICS general resource classes can also be refreshed with the CICS transaction CEMT. To refresh the main/virtual storage execute from a CICS terminal:
CEMT PERFORM SECURITY REBUILD

For the general resource classes eligible for the RAclist and GENLIST processing see chapter Appendix C in [RM98]. For more information on data set profiles and general resource profiles, that are not described in this thesis, and on how to create own resource class names refer to [RSG03].
6.3 Implementing RACF protection for the CICS region A06C001

6.3.1 The CICS region's SIT

In the SIT are stored CICS system initialisation parameters that specify system attributes. Some are required to secure the CICS region, some set attributes for the MQSeries CICS connection (refer to the chapters 5.4.2.2 “Configuring CICS to use the MQSeries CICS Bridge” and 5.4.2.5 “An automatic start job for the MQSeries CICS Bridge” on page 101 resp. 109), and some set other CICS system parameters (refer to chapter 4.5.2 “Setting up the CICS resources” on page 74). The SIT is assembled as a load table during the CICS region's start up. The information stored in it is used to suit the region environment to a well-suited level. Some CICS regions can share one SIT together, but if required, the CICS region can also have an own SIT. There should be used several scripts that built one SIT for a CICS region. Hence, the script for the SIT provided with the CICS installation leaves original. In another script there could be specified the modified parameters. If a failure happens due to modifying the SIT-parameters the original script can be used to remedy it. For instance, the SIT for the CICS region A06C001 is built by three scripts that are loaded within the SYSIN section of the CICS region start script CICSC001 (Listing 8). This start script resides in the data set SYS1.PROCLIB.

Listing 8: Extract from the data member SYS1.COMMON(CICSC001)

```plaintext
01 //CICSC001 PROC SYSIDNT=C001,

  50 //SYSIN DD DISP=SHR,DSN=CICS.COMMON.SYSIN(COMMON)

  51 // SYSSIN DD DISP=SHR,DSN=CICS.COMMON.SYSIN(&SYSIDNT)

  52 // SYSSIN DD DISP=SHR,DSN=CICS.COMMON.SYSIN(END)
```

During startup of CICS the default SIT for the CICS region is assembled from the default script COMMON that is stored in the CICS region data set CICS.COMMON.SYSIN (Listing 49, page 245). The second SIT script C001 (Listing 50 & Listing 51, both page 245), that is loaded after the COMMON script has been executed, changes some default SIT entries. It also specifies some new attributes for the CICS region, for example the region’s application identifier APPLID. The script C001 is located in the data set CICS.COMMON.SYSIN, too. As last script the third SIT script END is executed and the SIT is created. This script should be used for entries that specify some attributes for the CICS region shut down. However, this script is empty at this time.

Table 11 on the next page lists all SIT-parameters that have been modified during the process of securing the CICS region. The set options have been marked bold. Consider to activate the SIT-parameter SEC=YES rather at the end of the whole security procedure because it secures the CICS region completely. When no additional security mechanism has been previously set up and the SIT-parameter SEC is already activated, CICS resources cannot be accessed or executed.

The region's SIT can only be reassembled when the CICS region is restarted. For restarting the CICS region refer to APPENDIX C.1 “Restarting the CICS region” on page 239. Explanations about all CICS SIT parameters, if not described in this thesis, can be obtained from the book [SDG03] in part 3.
<table>
<thead>
<tr>
<th>SIT-parameter</th>
<th>Description</th>
<th>Refer to chapter</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEC = YES</td>
<td>NO</td>
<td>Activates the CICS security resp. initialises the external security interface</td>
</tr>
<tr>
<td>DFLTUSER = &lt;userid&gt;</td>
<td>Specifies the name of the DCU</td>
<td>6.3.4, page 175</td>
</tr>
<tr>
<td>PLTPISEC = RESSEC CMDSEC</td>
<td>NONE Specifies whether to use resource or command security checking for PLT-programs executed during CICS initialisation</td>
<td>6.3.5, page 177</td>
</tr>
<tr>
<td>PLTPIUSR = &lt;User ID&gt;</td>
<td>Specifies the name of the user ID used for PLT-programs running during the CICS initialisation</td>
<td>6.3.5, page 177</td>
</tr>
<tr>
<td>GMTEXT = &lt;text&gt;</td>
<td>The “Good Morning” text to be displayed on the first screen after log/sign on</td>
<td>6.3.7, page 182</td>
</tr>
<tr>
<td>GMTRAN = &lt;TRID&gt;</td>
<td>The transaction that display the “Good Morning” text on the terminal</td>
<td>6.3.7, page 182</td>
</tr>
<tr>
<td>GNTRAN = &lt;TRID&gt;</td>
<td>Specifies the transaction that CICS invokes when a user's terminal-timeout period expires</td>
<td>6.3.8, page 183</td>
</tr>
<tr>
<td>PSBCHK=YES</td>
<td>NO</td>
<td>PSB authorisation check for remote terminal users</td>
</tr>
<tr>
<td>SECPRFX = YES</td>
<td>NO</td>
<td>Specifies whether to use the CRU as a prefix for all resource names, or not.</td>
</tr>
<tr>
<td>XUSER = YES</td>
<td>NO</td>
<td>&lt;name&gt;</td>
</tr>
<tr>
<td>XTRAN = YES</td>
<td>NO</td>
<td>&lt;name&gt;</td>
</tr>
<tr>
<td>XCMD = YES</td>
<td>NO</td>
<td>&lt;name&gt;</td>
</tr>
<tr>
<td>CMDSEC = ASIS</td>
<td>ALWAYS</td>
<td>ASIS means that CICS obeys the CMDSEC option in the resource definition. When ALWAYS is chosen, the CMDSEC option of the resource definition is ignored and a command check is always done.</td>
</tr>
<tr>
<td>RESSEC = ASIS</td>
<td>ALWAYS</td>
<td>ASIS means that CICS obeys the RESSEC option in the resource definition. When ALWAYS is chosen, the RESSEC option of the resource definition is ignored and a resource check is always done.</td>
</tr>
<tr>
<td>APPLID = &lt;name&gt;</td>
<td>Specifies the name of the CICS region</td>
<td>6.5, page 205</td>
</tr>
</tbody>
</table>

Table 11: SIT-parameters set resp. modified for the CICS region A06C001
6.3.2 User management of the CICS region

The consideration which RACF user may access the CICS region has to precede the authorisation management. This results in thinking about which user should access the subsystem in which relationship. On the JEDI OS/390-server all users accessing TSO/E may also access the CICS region A06C001. These TSO/E-users have been summarised in following RACF user groups (date 1.3.2004):

**RACF User Group: RACF User IDs:**

<table>
<thead>
<tr>
<th>ADMIN:</th>
<th>BOSCH, BREITI, BUWD, BUWD1, DRECKER, HROMBA, JHORSWI, JOANNE, KDEGI, NILSM, NMICHA, VPET, PHERRM, RECKER, SPRUTH, MUELLER, EKOPP, WGS, WGREIS, TBOEHM, ELPUR, USTEMP, USETEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIPLOM:</td>
<td>MBAYER, RRONNE, SMUNZ, STEFANM, TBUSSE, YCUBAS, OAVIEN, ORAIB, AZIEN, MSCHLO</td>
</tr>
<tr>
<td>GAST:</td>
<td>GAST1, GAST2, GAST3, GAST99</td>
</tr>
<tr>
<td>PRAKT:</td>
<td>PRAKT1 ... PRAKT22, PRAKT24, PRAKT30 ... PRAKT53, PRAKT55 ... PRAKT69, PRAKT71 ... PRAKT100, PRAKT100 ... PRAKT183, PRAKT190, PRAK100</td>
</tr>
<tr>
<td></td>
<td>PRAK500</td>
</tr>
</tbody>
</table>

All the listed users/groups should access CICS with different permissions to the resources of CICS. The users of the group **GAST** should only play with some resources of the CICS subsystem without modifying anything. Trainees who are organised in the group **PRAKT** should access CICS to solve their study exercises with rights to modify only those resources that are needed during their exercises. Users defined in the group **ADMIN** and **DIPLOM** require an access to CICS to control and update the resources of the CICS region.

Additionally, the two required CICS user IDs – the CRU **STCCICS** and the DCU **C001DEF** – need also an access to special CICS resources. The user ID **PLTCICS** needed for the PLT-programs, that are started during the initial start of the CICS region, requires also an access to some specified CICS resources:

**RACF User Group: RACF Users:**

<table>
<thead>
<tr>
<th>CICSDEF</th>
<th>C001DEF</th>
</tr>
</thead>
<tbody>
<tr>
<td>CICSREG</td>
<td>STCCICS</td>
</tr>
<tr>
<td>SURRGRP1</td>
<td>PLTCICS</td>
</tr>
</tbody>
</table>

6.3.3 The CRU

6.3.3.1 What is it?

From RACFs point of view each started CICS region on OS/390 has to be associated with an own unique RACF user ID, since a CICS region, as a subsystem of OS/390, is treated like a user and hence it identifies itself to RACF. The CRU is required to start the CICS region under RACF control. With its authority RACF decides
which protected resources on the JEDI OS/390-server the CICS region may access. Furthermore, a CRU becomes important, when more than one CICS region are installed on an OS/390-server. In this case, CICS uses the region user ID to prefix resource definitions before sending a request to RACF. For example, a CICS transaction should be secured by a RACF transaction security profile. In that case, this profile has to stored with the CRU as prefix to use the transaction in the preferred region.

6.3.3.2 Defining the CRU to RACF

The CRU STCCICS is defined to RACF after the data for the CICS region have been installed. But firstly, its default user group CICSREG has to be added to the RACF database using the command ADDGRP (AG):

Example: ADDGROUP CICSREG OWNER(SYS1) SUPGRP(SYS1) OMVS(GID(1))

The owner SYS1 is also the default group and the superior group of the user group CICSREG. SYS1 is an IBM-supplied group and resides always on the highest level of the user group hierarchy. Within the OMVS keyword there are added the default OS/390 UNIX group identifier (GID) to the OMVS segment of the group (OMVS = OpenMVS, the OS/390 UNIX substem). For more information on this segment please refer to [RSA98]. Into the group CICSREG there can also be added other CRUs to assign one authority level to that group resp. to all CRUs connected within the group.

The user ID for the CICS region A06C001 is defined using the RACF command ADDUSER (AU):

ADDUSER user_id DFLTGRP(group_id) NAME(user_id_description)+
OWNER(owner_id) PASSWORD(******) +
segment_1(attributes) ... segment_n(attributes)

Example: ADDUSER STCCICS DFLTGRP(CICSREG) NAME(A06C001-REGION_ID) +
OWNER(CICSREG) PASSWORD(******) +
OMVS(UID(1) HOME(/) PROGRAM(/bin/sh))

Within the keyword DFLTGRP the CRU for A06C001 has been added to its default user group CICSREG. A name for the user ID has been specified with the NAME keyword whereas the OWNER of the user ID is IBMUSER. In case, that the CICS region wants to access some resources on the OS/390 UNIX subsystem, it has been added the OMVS segment with some default parameters to the user profile. As a security aspect, always define a password for the CRU, do not set it to the default one which is the name of the default group. The password is never checked by RACF to access resources.

After the CRU is created, it is used to run the CICS region. There are three ways to run the CICS region: as a Started Task using the RACF started procedure table ICHRIN03, as a Started Job, or simple as a Job. The CICS region A06C001 runs as a Started Job. For information on how to start CICS as a Started Task and as a Job refer to [RSG03].
6.3.3.3 Authorising the CRU to invoke CICS as a Started Job

Invoking the CICS region A06C001 as a Started Job requires that the CRU is specified on a STARTED general resource class profile. This profile refers to the CICS region start up script name CICSC001 and to the script's job name, that is also CICSC001\(^{17}\). Therefore, the discrete STARTED general resource profile is named as CICSC001.CICSC001. In case, another CICS region should be started with the same script using another job name, for example CICSC002, the discrete STARTED general resource profile has to be named as CICSC001.CICSC002. If one CRU should be used for different CICS regions started by the same start script but with different job names, the profile could be named as CICSC001.CICSC**.

The profile CICSC001.CICSC001 has been added to the STARTED general resource class with the RACF command *RDEFINE(RDEF)*:

```
RDEFINE STARTED script_name.job_name OWNER(owner_id) +
STDATA(USER(CRU))
```

*Example:* RDEFINE STARTED CICSC001.CICSC001 OWNER(IBMUSER) +
STDATA(USER(STCCICS))

The *OWNER* of the profile is IBMUSER and the CRU is added to the associated *STDATA* segment using the keyword USER. The profile can be displayed with the RACF command *RLIST(RL)*. The connected CRU is displayed if the segment name is entered as showed on Figure 74 on the next page.

After defining the STARTED general resource profile to the general resource class STARTED, the profiles in main/virtual storage need to be refreshed using the command *SETROPTS RACLIST*:

*Example:* SETROPTS RACLIST(STARTED) REFRESH

In case, the general resource class has not been activated and has not been stored in main/virtual storage previously, it must be firstly activated using the *SETROPTS CLASSACT* command together with the *RACLIST* command:

*Example:* SETROPTS CLASSACT(STARTED) RACLIST(STARTED)

---

\(^{17}\) The script is stored in the data set SYS1.PROCLIB on the JEDI OS/390-server.
6.3.3.4 Authorities required for the CRU

The CRU needs an access to all the resources CICS itself needs to use. There are two types of these resources – resources external to CICS, such as files on hard disks, or the spool system and resources internal to CICS, such as some CICS transactions. The authority to access external resources is obtained from the CRU, not from the CICS terminal user. Each CRU has to be associated with all the disk data sets that it uses. On JEDI OS/390-server the CRU has got an access to the whole CICS TS install data set having the high-level qualifier `CICSTS13`.

Authorising the access for the CRU to that data set is described in chapter 6.3.6 “The CICS region data set protection” on page 178. More authorities required for CICS region user IDs to access external resources of CICS can be obtained from the book [RSG03] in chapter 3 “CICS data set and system security”.

Internal resources like CICS data sets and CICS system transactions need also to be authorised by the CRU. Authorising these so-called Category-1-transactions is explained in chapter 6.4.4 “The CICS Transaction Security” on page 189. When the SIT-parameter `XUSER` has been activated, the CRU has to be used as a surrogate user of some other user IDs. There are three types of using the CRU as a surrogate user ID – for the DCU, for the PLTPIU, and for the user ID used for transient data trigger transactions. How to use the Surrogate User Security is described in chapter 6.4.3 “The Surrogate User Security” on page 187. To define the PLTPIU refer to chapter 6.3.5 “The PLTPIU” on page 177.
6.3.4 The DCU

6.3.4.1 What is it?

When a CICS region is to be secured, it is needed to define the DCU. This user ID must match the value of the SIT-parameter DFLTUSER. If no value is specified on this parameter, CICS refers to the IBM-supplied user ID CICSUSER. In that case, a RACF user profile for CICSUSER must be defined to the RACF database. Each CICS region should have an own DCU as an aid to debugging. However, one DCU can be shared for all CICS regions.

CICS always signs on the DCU each time a CICS region starts up. If no DCU has been created to RACF resp. specified on the SIT-parameter DFLTUSER, CICS cannot sign on the DCU, and therefore, the CICS initialisation is terminated and a failure message is issued. When the sign on of the DCU is successful, its security attributes are assigned to all terminal users before they sign on. Therefore, a user connected to the CICS terminal is always logged on to CICS, but not signed on. If this user wants to sign on to CICS, the DCU must be able to execute a sign on transaction. This could be the IBM-supplied CICS transaction CESN, which opens the “Signon to CICS” panel (Figure 75). On this panel a valid RACF user ID with its associated password must be entered to access the CICS region and to execute some authorised resources. Furthermore, some other transactions, that need no protection, can also be authorised for the DCU. For example, each CICS user may additionally use the CMAC transaction to display on-line messages and codes without signing on to the CICS region. In that case, the DCU must also have a permission to initiate the CMAC transaction. To execute such “unsecured” transactions by the DCU a CICS general resource profile for those transactions has to be created. How to define such a profile is described in chapter 6.4.4 “The CICS Transaction Security” on page 189.

Figure 75: The CICS SignOn panel
And last, but not least, the CICS segment of the DCU’s terminal user data is established for user who have no own CICS segment data in their RACF user profile.

### 6.3.4.2 Defining the DCU to RACF

For the CICS region A06C001 the DCU has been already defined to the RACF database. But, this user ID matched the CRU of that CICS region. Such a combination is not a good choice, because both are two different kinds of user IDs. To rectify this problem, a new DCU has been created. Firstly, a group for the DCU has been added to RACF. Into this group there can also be added other DCUs to give them the same authority level as the group have. If required, there can also be created another group that holds DCUs having another authority level. The group for the DCU C001DEF has been named as CICSDEF having the superior group and the owner SYS1:

*Example:* ADDGROUP CICSDEF SUPGROUP(SYS1) OWNER(SYS1) OMVS(GID(1))

After that, the DCU C001DEF has been created using the RACF command `ADDUSER`:

*Example:* ADDUSER C001DEF DFLTGRP(CICSDEF) NAME(A06C001-DEFAULT_ID) +

OWNER(CICSDEF) PASSWORD(*******) +

OMVS(UID(1) HOME(/) PROGRAM(/bin/sh)) CICS(OPCLASS(1))

The default group of the DCU is CICSDEF, has the owner SYS1, and the well fitting name description `A06C001-DEFAULT_ID`. A password should always be defined for the DCU. If omitted, it is set to the default one – the group identifier, but this is not suggested. Within the OMVS segment there has been specified some parameters used for the OS/390 UNIX subsystem: the UNIX identifier, the home directory, and the program directory. For the DCU there has been also added a CICS segment with some default operator data, that should be differently for each DCU defined to RACF. The CICS segment for the DCU C001DEF contains the operator class 1 specified within the keyword OPCLASS. If no CICS segment is specified on the user's profile, CICS assigns built-in system default values for the DCU as same as for each user who has no own CICS segment and who signs on to the CICS region. On the CICS segment there can be specified some more terminal operator data – the operator identification OPIDENT, the operator priority OPPRTY, the TIMEOUT, and the XRFSOFF parameters. These parameters are explained in the book [RSG03] in detail. After the DCU is defined to the RACF database, its information can be displayed with the RACF command `LISTUSER` as used in Figure 76 on the next page.

Furthermore, the DCU has to be defined on the access list of the CICS region's APPL general resource profile to access the CICS region itself. For information on how to grant this access browse to chapter 6.5 “Authorising access to the CICS region” on page 205. If the SIT-parameter XUSER is activated, the CRU should be authorised to be the surrogate user of the DCU. How to authorise the CRU to be a surrogate of the DCU is explained in chapter 6.4.3 “The Surrogate User Security” on page 187.
The DCU is assigned to the CICS region when it is defined to RACF and when it is set on the SIT-parameter DFLTUSER. However, it is not required to specify a DCU on this SIT-parameter. In this case, the IBM-supplied DCU CICSUSER is chosen. Therefore, a user profile for that user ID must be defined to RACF. The DCU of the CICS region A06C001 C001DEF has been specified in the default SIT script C001 as same as the parameter SEC:

<table>
<thead>
<tr>
<th>11 DFLTUSER=C001DEF,</th>
<th>DEFAULT CICS USER</th>
</tr>
</thead>
<tbody>
<tr>
<td>37 SEC=YES,</td>
<td>CICS SECURITY (STANDARD = YES)</td>
</tr>
</tbody>
</table>

**Figure 76: LISTUSER C001DEF NORACF CICS OMVS**

6.3.5 The PLTPIU

All CICS programs that are executed during the initialisation of the CICS region have to run under an own user ID that is called the PLTPIU. Those programs are compiled to the PLT that is loaded during the CICS system initialisation. For each CICS region this user ID is specified in the SIT on the parameter PLTPIUSR. If this parameter is omitted the CRU is used for all PLTPI programs. Therefore, the CRU must have an access to all the resources that these programs use including all the transactions started within the PLT-programs. On the CICS region A06C001 an own PLTPIU is used for programs started during such post-initialisation processing. The new user ID PLTCICS stored to the user group SURRGRPI has been created to the RACF database using the two commands ADDGROUP and ADDUSER:
Example 1: ADDGROUP SURRGRP1 OWNER(SYS1) SUPGROUP(SYS1)

Example 2: ADDUSER PLTCICS NAME(PLT_USER-ID) OWNER(SURRGRP1) +DFLTGRP(SURRGRP1)

After that, the created PLTPIU has to be specified on the SIT-parameter PLTPIUSR (refer to Listing 51, page 245). For the PLTPIU exists an additional SIT-parameter called PLTPISEC. It specifies whether or not to use resource (PLTPISEC=RESSEC) or command security (PLTPISEC=CMDSEC) checking for PLT-programs executed during CICS initialisation. The resource security has been chosen for the PLT-programs (Listing 51).

CICS always performs a surrogate user security check to authorise the CRU acting as a surrogate user of the user ID specified within the SIT-parameter PLTPIUSR. A surrogate user security check is not required if no PLTPIU is specified resp. the CRU is the PLTPIU. For information on how to enable the surrogate user security checking for the PLTPIU refer to chapter 6.4.3 “The Surrogate User Security” on page 187.

### 6.3.6 The CICS region data set protection

#### 6.3.6.1 Protecting the CICS region data sets

All data, that is required to start and run the CICS region A06C001, is installed to the data sets having the high-level qualifier “CICS” and “CICSTS13”. After the installation, both data sets have to be secured by the RACF data set protection. To secure the data sets generic group data set profiles have to be defined to the DATA-SET resource class. For the whole CICS TS install data, the RACF data set profile CICSTS13.** is created whereas the whole CICS region data set is protected by the data set profile CICS.**. Following listing shows how to use the RACF command ADDSD (shortcut AD) to create the generic data set profile CICS.**:

```
ADDSD 'data_set_name' OWNER(user_id) UACC(option)
```

Example: ADDSD 'CICS.**' OWNER(ADMIN) UACC(NONE)

Because of using the double asterisk, RACF knows to store the profile in its database as a generic one. It protects all data having the high-level qualifier “CICS” with this one profile. When executing this command, nobody has an access to the data stored within this data set. This is specified within the keyword UACC meaning universal access. The generic data set profiles for the CICS install data set having the high-level qualifier “CICSTS13” have been created using the same way.

Due to a failure occurred during the CICS region reinitialisation after an IPL of the JEDI OS/390-server, an additional data set profile is created to store the CICS system log having the qualifiers CICS,A06C001.**. This data set profile secures the VSAM KSDSs used for the primary and secondary CICS system logs. Differently from the other two data set profiles the universal access has been set to UPDATE because CICS requires it to
write CICS log messages to the data set. If this access is not permitted, the primary CICS system log \textit{DFHLOG} will not be reinitialised after the CICS region is restarted due to an IPL of an OS/390-server. Usually, it is not recommended to specify an UPDATE authority to the \textit{UACC}-parameter because any user is in the position to delete the data set. Therefore, the a few CICS terminal users have been excluded from the access; see next chapter for the specification of an access list to this data set. How to patch such an access failure is described in chapter Appendix C.2 “Correction of a CICS system log failure after an OS/390-server IPL and a CICS restart” on page 240.

Furthermore, there have been defined four more data set profiles. Two of them protect the CICS load modules libraries named as \texttt{CICSTS13.CICS.SDFHLOAD} and \texttt{CICSTS13.CICS.PRAKLOAD} – both parts of the CICS TS install data set. The other two protect the transient data intrapartition data set stored in \texttt{CICS.C001.DFHINTRA} and the temporary storage data set \texttt{CICS.C001.DFHTEMP} – both parts of the CICS region data sets. They all have been specified using the \texttt{ADDS} command as described above. The data set profile \texttt{CICSTS13.CICS.PRAKLOAD} is used as an example on how to define a generic data set profile having no substitute characters. The protected data set \texttt{CICSTS13.CICS.PRAKLOAD} is used to store CICS programs created by CICS users. It has been protected using following RACF command:

\textbf{Example:} \texttt{ADDS 'CICSTS13.CICS.PRAKLOAD' GENERIC OWNER(SYS1) UACC(READ)}

To secure this profile even though as a generic data set profile, the keyword \textit{GENERIC (GEN)} must be specified within the \texttt{ADDS} command. In case of modifying or deleting this profile, \textit{GENERIC} must always be written within the RACF command behind the data set profile name. If it has been omitted, this data set profile is not found by RACF but the discrete profile is displayed, if defined.

Summarised, the \textit{UACC}-parameter for all the defined data set profiles has been set as follows:

\begin{verbatim}
CICS.**                UACC(NONE)
CICSTS13.**           UACC(NONE)
CICS.A06C001.**       UACC(UPDATE)
CICSTS13.CICS.PRAKLOAD UACC(READ)
CICSTS13.CICS.SDFHLOAD UACC(READ)
CICS.C001.DFHINTRA    UACC(NONE)
CICS.C001.DFHTEMP     UACC(NONE)
\end{verbatim}

It is noticed, when a data set profile is to be created, the high-level qualifier of the associated data set must be firstly defined as a group ID to RACF. For the generic data set profiles the group IDs \texttt{CICS (Example 1)} resp. \texttt{CICSTS13 (Example 2, next page)} have been defined using the RACF command \texttt{ADDGROU\textit{P} (AG)}:

\textbf{Example 1:} \texttt{ADDGROU\textit{P} CICS OWNER(SYS1) SUPGRP(SYS1)}

\textbf{Example 2:} \texttt{ADDGROU\textit{P} CICSTS13 OWNER(SYS1) SUPGRP(SYS1)}
Attention, do not forget updating the RACF database using the `SETROPTS` command to activate the data set protection:

*Example:* `SETROPTS GENERIC(DATASET) REFRESH`

### 6.3.6.2 Authorising access to the CICS data sets

For all the six defined data set profiles there have to be added RACF users/groups to the access list of the profiles. As an example on how to add such an access list to a data set profile, the profile `CICS.**` is chosen. Since the universal access has been set to `NONE`, no user has an access to the data set until the user is added with an appropriate authority level to the data set profile. For all users there is needed a well configured access list to the data set. It should be accessed by the `ADMIN` and `DIPLOM` group with an `ALTER` access permission because both need the full control of this data set (Example 1). The CRU `STCCICS` gets at least the `UPDATE` access because it changes some data in the data sets during the CICS region runs (Example 2). Other users get no access to the whole data set at this time. If required, those users, who need also an access, can be added to the access list with the appropriate permission. Using the RACF command `PERMIT` adds the user/group IDs with their assigned permission to the data set profile `CICS.**`. This command must be executed for each security level:

```plaintext
Example 1: PERMIT 'CICS.**' ID(ADMIN,DIPLOM) ACCESS(ALTER)
Example 2: PERMIT 'CICS.**' ID(STCCICS) ACCESS(UPDATE)
```

Defining an access list for a data set profile consisting of non-substitute characters is explained using the data set profile `CICSTS13.CICS.PRAKLOAD` as an example. Everybody has at least `READ` access to the protected data as specified within the `UACC` parameter. Additionally, the user groups `ADMIN` and `DIPLOM` get an `ALTER` access to have the full control of the data set (Example 3). Because this data set should be used by trainees to store, modify, and delete their load modules created during an educational course, the group `PRAKT` has to get an `UPDATE` access (Example 4, next page). The following definitions adds the user groups with their permissions to the access list of such a generic data set profile having no substitute characters. Each command must include the keyword `GENERIC`, hence, the access list will be added to the generic data set profile instead the discrete one, if it exists. Otherwise an error message is displayed:

```plaintext
Example 3: PERMIT 'CICSTS13.CICS.PRAKLOAD' GENERIC ID(ADMIN,DIPLOM) + ACCESS(ALTER)
Example 4: PERMIT 'CICSTS13.CICS.PRAKLOAD' GENERIC ID(PRAKT) + ACCESS(UPDATE)
```
Summarised, the following table lists all the specified access lists of the data set profiles:

<table>
<thead>
<tr>
<th>Data set security profile</th>
<th>Users/Groups</th>
<th>Access permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>CICS.**</td>
<td>ADMIN, DIPLOM</td>
<td>ALTER</td>
</tr>
<tr>
<td></td>
<td>STCCICS</td>
<td>UPDATE</td>
</tr>
<tr>
<td>CICSTS13.**</td>
<td>ADMIN, DIPLOM</td>
<td>ALTER</td>
</tr>
<tr>
<td></td>
<td>STCCICS</td>
<td>READ</td>
</tr>
<tr>
<td>CCIS.A06C001.**</td>
<td>ADMIN, DIPLOM</td>
<td>ALTER</td>
</tr>
<tr>
<td></td>
<td>PRAKT, GAST</td>
<td>NONE</td>
</tr>
<tr>
<td>CICSTS13.CICS.PRAKLOAD</td>
<td>ADMIN, DIPLOM</td>
<td>ALTER</td>
</tr>
<tr>
<td></td>
<td>PRAKT</td>
<td>UPDATE</td>
</tr>
<tr>
<td>CICSTS13.CICS.SDFHLOAD</td>
<td>ADMIN, DIPLOM</td>
<td>ALTER</td>
</tr>
<tr>
<td>CICS.C001.DFHINTRA</td>
<td>STCCICS</td>
<td>CONTROL</td>
</tr>
<tr>
<td>CICS.C001.DFHTEMP</td>
<td>STCCICS</td>
<td>CONTROL</td>
</tr>
</tbody>
</table>

Table 12: Access permission to the CICS data sets

The access list of the generic data set profile CICSTS13.** is created using the command `PERMIT` as described within the data set profile CICS.**. The other three generic data set profiles having no substitute characters have been defined as described within the data set profile CICSTS13.CICS.PRAKLOAD.

The CICS transaction NACT of the CICS application can only be executed if the CRU STCCICS has at least the `READ` authority to the data set with the high-level qualifier “TBUSSE.CICSADP”. But with this authority only the browse function can be used, if another function like modifying or deleting should be executed, the application replies with an error message. These functions can only be executed, when the CRU has an `UPDATE` access to the data set. The command to define the appropriate authority for the CRU reads as follows:

```
Example: PERMIT 'TBUSSE.CICSADP.**' ID(STCCICS) ACCESS(UPDATE)
```

But beware, this data set is allocated to the CICS start up libraries. Each changing of the access list of their associated data set profiles implies a restart of the CICS region. Only after restarting, the access list is updated. How to add such a data set to the CICS start up libraries is explained in chapter 4.5.2 “Setting up the CICS resources” on page 74.

After the RACF data set profiles have been added, they can be listed using the command `LISTDSD (LD)`. For displaying the generic data set profiles having no substitute characters the keyword `GENERIC` must also be used. When `AUTH` is specified only authority information is displayed, but when specifying `ALL` the whole profile information is listed – for example the creation date, the authority information, too, and some more. There can be specified some more keywords, for using them refer to [CSB98].

```
LISTDSD DATASET('data_set_name') [AUTH] [GENERIC] [ALL]
```

```
Example 1: LISTDSD DATASET('CICS.**') ALL
```
Example 2: \texttt{LISTDSD DATASET('CICSTS13.CICS.PRAKLOAD') GEN AUTH}

The command \texttt{LISTDSD} can also be used to display the profile information of all profiles beginning with the same characters. It is only needed to know a few prefix characters of the data sets, for example \texttt{CICS}:

\texttt{LISTDSD PREFIX(prefix_characters) [AUTH] [GENERIC] ALL}

Example 1: \texttt{LISTDSD PREFIX(CICS) ALL}

Example 2: \texttt{LISTDSD PREFIX(CICS) GENERIC ALL}

If no prefix is known, the RACF command \texttt{SEARCH (SR)} lists the names of all data set profiles stored in the RACF database when \texttt{NOMASK} is specified (Example 1). Some characters specified within the \texttt{MASK} parameter provides the same result as the \texttt{LISTDSD PREFIX} command (Example 2). \texttt{CLASS} needs not to be specified because it is the default value when no alternative item or operand is specified:

Example 1: \texttt{SEARCH CLASS(DATASET) NOMASK}

Example 2: \texttt{SEARCH CLASS(DATASET) MASK(CICS)}

As next, it is necessary to specify some entries in the SIT definition script of the CICS region to prepare the activation of the CICS region security.

### 6.3.7 Informing CICS terminal users about the forthcoming security change

Before activating the security for the CICS region, all CICS users have to be informed about the forthcoming update. This information should appear on the first screen each time a CICS user logs on to CICS until the security is activated. Such a message text can be specified on the SIT-parameter \texttt{GMTEXT}. To display it there can be used two IBM-supplied CICS transactions – CSGM known as the “Good Morning” transaction, or CESN known as the “Sign On” transaction. Before the CICS region has been RACF-protected, the CICS transaction CMSG has been always executed as first transaction after a user has been logged on to CICS terminal entry panel because it was specified on the SIT-parameter \texttt{GMTRAN} (cf. Listing 50, page 245). As the other possibility, the CICS transaction CESN can also be specified on this SIT-parameter (cf. Listing 51, page 245):

\[
\begin{array}{l}
\text{C001-temporary version:} \\
15 \text{GMTRAN=CSGM,} \quad \text{ENTRY PANEL} \quad \text{CSGM|CESN}
\end{array}
\]

\[
\begin{array}{l}
\text{C001 - final version:}
\end{array}
\]
The message text specified within \texttt{GMTEXT} can contain up to 246 characters and can be three lines at all. The following text has been used to inform the users of the security update (cf. Listing 50, page 245):

\begin{verbatim}
C001-temporary version:
12 GMTEXT='CICS C001 A06C001 -- WELCOME AT UNIVERSITY OF LEIPZIG
13     !!ACHTUNG!! AB MITTWOCH, 26.02.2003, MUSS SICH JEDER FÜR CICS
14     ANMELDEN! WIR REAGIEREN DAMIT AUF DIE SICHERHEITSPROBLEME!',
\end{verbatim}

After the CICS region has been restarted to generate the new SIT, the information about the security update appears on the entry screen as shown on Figure 77. Now each users knows that the CICS region will be secured until the date 26.02.2003.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure77.png}
\caption{Terminal screen after log on to CICS (without security)}
\end{figure}

\subsection*{6.3.8 Other parameters necessary for CICS security}

There have been set resp. modified two more important parameters in the SIT definition script \texttt{C001} to overwrite the parameters of the \texttt{COMMON} script – \texttt{SECPFX}, \texttt{GNTRAN}, and \texttt{PSBCHK} (Listing 51, page 245). The parameter \texttt{SECPFX} is activated in the default SIT definition script \texttt{COMMON}, but is only used if multiple CICS regions are running on the OS/390-server. With it, CICS prefixes the resource names of the different CICS regions to pass them to RACF for authorisation. \texttt{SECPFX} has been set inactive in the script \texttt{C001} – \texttt{SECPFX=NO}
(Listing 51, line 38). In contrast to the `GMTRAN` parameter, the `GNTRAN` ("Good Night" TRANsaction) specifies the CICS transaction invoked due to the expiration of the terminal time-out period. The default parameter in `COMMON` is set to `GNTRAN=CESN` to return to the Sign-On panel after the terminal time-out. However, in case of such a time-out CICS should log off the user from the terminal. Therefore, the transaction CESF with the option `LOGOFF` has been specified on the `GNTRAN` parameter (Listing 51, line 19). Furthermore, when the user signs off with transaction CESF plus option `GOODNIGHT`, CICS logs off the user, too.

The SIT-parameter `PSBCHK` can be activated, when a PSB authorisation check should be made for remote terminal users. Because this parameter has been set to `PSBCHK=NO` CICS checks only the remote link, but not the remote user.
6.4 Securing the resources for the CICS region A06C001

6.4.1 Decision about useful and necessary security mechanism

Before any user may access a secured CICS region it is important to find out what security levels RACF provides for the transaction monitor. The access to the region is controlled by the following different security mechanisms (according to [RSG03]):

- Terminal User Security
- Preset Terminal Security
- Surrogate User Security
- CICS Transaction Security, also called Transaction-Attach Security
- CICS Command Security
- CICS Resource Security, subdivided into security for:
  - Advanced Program-to-Program Communication through an LU6.2 session (APPC LU6.2)
  - DB2 Resource Classes
  - Transient Data
  - VSAM and BDAM Files
  - Journals and Log Streams
  - STARTED- and XPCT-checked Transactions
  - CICS Application Programs
  - PSBs
  - Temporary Storage
- QUERY SECURITY Command Security as part of the CICS Command security
- CICS Web Support Security
- MultiRegion Operation (MRO) Security
- Front End Programming Interface (FEPI) Security
- CICS Business Transaction Services (BTS) Security
- RACF PassTickets.

For the CICS region A06C001 on the JEDI OS/390-server there has been enabled the Terminal User Security, the Surrogate User Security, the CICS Transaction Security, and the CICS Command Security. The terminal user security is not more as defining user IDs to the RACF database. These user IDs have to be used to sign on the CICS region. When they enter their user IDs with their associated passwords using a Sign On transaction (for example CESN), CICS verifies the user IDs and the passwords. If both are valid the user may access the CICS region. Although the user is signed on to CICS, he/she cannot use any resource of the CICS region until the user gets the authorisation to use CICS transactions, for example. For using the Terminal User Security refer to chapter 6.4.2 “The CICS Terminal User Security” on page 187.

With the Surrogate User Security CICS is able to check whether a surrogate user is authorised to act for another user without knowing the user’s password. CICS can enforce surrogate user security checking for the DCU, for preset terminals, and for the user ID of started transactions, of PLT post-initialisation programs, of BTS pro-
cesses, and of EXCI calls. Also, the user ID associated with transient data queues, and the user ID supplied on
DB2-parameters can have a surrogate user ID. On the CICS region A06C001 there is performed a surrogate user
checking against the CRU, if it is authorised to act as a surrogate user of the DCU and of the user ID of the PLT
programs (cf. chapter 6.4.3 “The Surrogate User Security” on page 187).

Using the CICS Transaction Security secures one part of the CICS resources – the CICS transactions. The
CICS users may only initiate those transactions in the CICS region, for which they have a permission. Such a
transaction is for example the CICS-supplied transaction CEMT (CICS Execute-level Master Terminal). With
this transaction there can be displayed or altered CICS environmental information. Another CICS-supplied trans-
action that is to be protected is the CEDA transaction. CEDA can be used to define and/or delete CICS resources. Implementing the transaction security is described in chapter 6.4.4 “The CICS Transaction Security” on page
189.

After activating these security mechanisms, a problem with the CICS Transaction Security has been occurred
on the CICS region. Due to security reasons, the CEMT transaction has been only allowed to execute by design-
nated users because this transaction assumes the control of CICS system parameters. These parameters can be
displayed and changed with CICS commands called System Programming (SP)-type commands. Aside changing
system parameters using the transaction CEMT, they can also be changed by the SP-type commands executed
within CICS application programs. However, SP-type commands also provide information about user- and sys-
tem-started resources. A simple example illustrates such a common situation. On the one hand, only authorised
users may shut down the CICS region using the CICS SP-type command \texttt{PERFORM SHUTDOWN} executable within
the transaction CEMT. It is simple to understand that a student or trainee should not have the right to perform
this command. On the other hand, a trainee, who wants to monitor the execution of a started resource (e.g. a
CICS program), gets this information when the CICS SP-type command \texttt{INQUIRE TASK} is executed within the
transaction CEMT. Hence, users who may shut down a CICS region and users who only want information about
a started resources need different access permissions for the SP-type commands such as \texttt{READ}, \texttt{UPDATE}, or \texttt{-ALTER}. Such CICS SP-type commands are secured with the CICS Command Security mechanism. Please refer to
chapter 6.4.5 “The CICS Command Security” on page 197 for using this security mechanism.

When all these security options are set, the JEDI OS/390-server will have a right secured CICS region at the
demanded level. In case of future security demands, it could be possible that more security options have to be ac-
tivated. For example, a secured preset terminal gives only a special local terminal and its associated user access
to CICS transactions without sign on. Preset Terminal Security is usually used for terminals without keyboards.
More security levels could be set when intercommunication between different CICS regions is used; either if con-
nected by an SNA access method in various sysplexes like APPC, or using MRO in a single sysplex. If full re-
source control is required for CICS application programs, program specification blocks, files, journal logging, or
for specific queues the associated security parts of the CICS Resource Security may be activated.

All security mechanisms for the CICS region are activated within the CICS region's SIT.
6.4.2 The CICS Terminal User Security

The terminal user security is the main CICS security mechanism. This mechanism controls whether a user is defined to RACF when the user wants to sign on to the CICS region. Such a user must have a predefined RACF user ID profile. However, all users may log on to the CICS region. A user who is not signed on can use only those resources, for example transactions, the DCU \textit{C001DEF} may use or transaction that have a universal access. Which resources the user may use after a sign on to the CICS region depends on other CICS security mechanisms, for example the transaction and command security. These security mechanisms secure the CICS resources using associated resource security profiles. If a user or a user group is added to the access list of a resource security profile they may use all the resources specified within the member list of the security profile.

The consideration which user may sign on to the CICS region A06C001 has to precede the user management of CICS. This results in thinking about which user should access the subsystem in which relationship. On the JEDI OS/390-server users of the RACF groups \textit{ADMIN}, \textit{DIPLOM}, \textit{GAST}, and \textit{PRAKT} may access the CICS region. Refer to chapter 6.5 “Authorising access to the CICS region” on page 205 on how to give these four user groups an access to the CICS region.

When the SIT-parameter \textit{SEC} is activated (\textit{SEC=YES}), all users must sign on to the CICS region with their user ID and the associated password. But at this point no other security mechanism is set. Because of this, the signed on users cannot use any of the CICS resources in the CICS region A06C001. Therefore, some more security mechanisms have to be established, firstly it is described how to activate the CICS transaction security mechanism. After all required security mechanisms have been activated the SIT-parameter \textit{SEC} may be set (cf. line 36, Listing 51, page 245).

6.4.3 The Surrogate User Security

On the CICS region A06C001 the CRU \textit{STCCICS} is used as a surrogate user for the DCU \textit{C001DEF} and for the user ID of PLT programs resp. started transactions \textit{PLTCICS}. Surrogate user checking is enabled when \textit{XUSER=YES} is specified as a parameter in the SIT (cf. line 45, Listing 51, page 245). On this parameter it can only be chosen between the values \textit{YES} or \textit{NO} because this SIT-parameter always refers to the IBM-supplied general resource class \textit{SURROGAT}. An own defined resource class cannot be used. The surrogate user security comes only in effect, when the SIT-parameter \textit{SEC} is also activated.

There are three forms of surrogate security profiles, which must conform the following naming conventions:

- \textit{userid.DFHINSTL}: This form of a surrogate class profile is required for the PLTPIU if specified on the SIT-parameter \textit{PLTPIUSR}, for the DCU specified on the SIT-parameter \textit{DFLTUSER}, for user IDs associated with trigger-level transactions, for user IDs specified for preset terminal security, and for user IDs specified within the DB2 resource definition. On the CICS region A06C001 there have been created two
of such profiles, one for the PLTPI user ID, and one for the DCU. The CRU STCCICS has to be author-
ised to act on behalf of these both users.

- \textit{userid.DFHSTART}: Such a surrogate security profile can be created for user IDs of started transactions
  or for user IDs of CICS BTS processes started by a \textit{RUN} command. One profile has been created to start
  the MQSeries CICS Bridge under the authority of the surrogate user PLTCICS instead of the DCU.

- \textit{userid.DFHEXCI}: This surrogate security profile is only needed for EXCI calls. Such a profile has not
  been defined the \textit{SURROGAT} general resource class. For information on how to use this profile refer to
  chapter 7 “Surrogate user security” in [RSG03].

After the DCU C001DEF has been created to the RACF database and it has been specified within the SIT-
parameter \textit{DFLTUSER} its surrogate security profile has to be created using following command:

\begin{verbatim}
RDEFINE SURROGAT DCU.DFHINSTL UACC(option) OWNER(DCU)
Example: RDEFINE SURROGAT C001DEF.DFHINSTL UACC(NONE) OWNER(C001DEF)
\end{verbatim}

The CRU STCCICS has been added to the access list of the surrogate profile C001DEF.DFHINSTL using the
RACF command \textit{PERMIT}. It is noticed, that each surrogate user must have at least \textit{READ} access:

\begin{verbatim}
PERMIT DCU.DFHINSTL CLASS(SURROGAT) ID(CRU) ACCESS(option)
Example: PERMIT C001DEF.DFHINSTL CLASS(SURROGAT) ID(STCCICS) +
           ACCESS(READ)
\end{verbatim}

As next, the CRU STCCICS must also be authorised as a surrogate user to run the PLT-programs during the
CICS region initialisation on behalf of the PLTPIU PLTCICS. Following commands has been executed:

\begin{verbatim}
Example 1: RDEFINE SURROGAT PLTCICS.DFHINSTL UACC(NONE) OWNER(PLTCICS)
Example 2: PERMIT PLTCICS.DFHINSTL CLASS(SURROGAT) ID(STCCICS) +
           ACCESS(READ)
\end{verbatim}

For example, during the CICS region initialisation” there is started the MQSeries CICS Bridge using the
PLT-program \texttt{STRTCKBR} (refer to ch. 5.4.2.5 “An automatic start job for the MQSeries CICS Bridge”, page
109). All the resources started within PLT-programs must run under the authority of the PLTPIU. Therefore, this
user ID must be authorised to all the resources referenced by \texttt{STRTCKBR}, for example the transaction CKBR
which starts the MQSeries CICS Bridge Monitor Task. This transaction again starts another one – the MQSeries
CICS DPL Bridge Task transaction CKBP. However, this transaction may always run under the authority of the
DCU (see chapter 6.4.4.6 “Securing the MQSeries CICS transactions used for the NACT application”, page
195)!

CICS requires that each user ID who starts a transaction without using a terminal is a surrogate of the user ID
that runs the transaction. Therefore, the PLTPIU which starts the CICS transaction CKBP must be a surrogate
user of the DCU under which CKBP is running. Following commands create such a surrogate class profile for the DCU and add the PLTPIU as the surrogate user to the access list of the profile:

Example 1:  RDEFINE  SURROGAT  C001DEF.DFHSTART  UACC(NONE)  OWNER(C001DEF)

Example 2:  PERMIT  C001DEF.DFHSTART  CLASS(SURROGAT)  ID(PLTCICS)  +
ACCESS(READ)

After all these profiles have been defined, the general resource class SURROGAT stored in main/virtual storage must be refreshed using the RACF command SETROPTS RACLIST. If this security class is not activated yet, the SETROPTS CLAS ACT command has to be entered. For the general resource class SURROGAT it is also required to activate sharing of the in-storage profiles, therefore RA CLIST should also be specified within the SETROPTS command:

Example 1:  SETROPTS  RA CLIST(SURROGAT)  REFRESH

Example 2:  SETROPTS  CLAS ACT(SURROGAT)  RA CLIST(SURROGAT)

6.4.4 The CICS Transaction Security

6.4.4.1 The CICS Transaction Security Mechanism

As next, the CICS transactions as part of the CICS resources have to be secured. The permission to execute a CICS transaction is controlled by the CICS Transaction Security Mechanism whether the transaction is initiated by the user, by another transaction, or by programs external to CICS using the CICS terminal. This method is sometimes referred to as Transaction-Attach Security. A transaction that is executed without accessing the terminal directly, for instance during the CICS region start up, is also RACF checked. This method of the transaction security mechanism is also called as the Non-Terminal Transaction Security.

The Transaction Security Mechanism is activated by the associated SIT-parameter XTRAN set in the CICS SIT which becomes only active when the SIT-parameter SEC is also set to YES. If the SIT-parameter XTRAN is set to YES, RACF has to use the security profiles stored within the IBM-supplied RACF resource classes TCICSTRN and GCICSTRN, whereas specifying an own defined resource class name on the SIT-parameter XTRAN implies, that RACF has to use the security profiles stored in this resource class.

Transactions existing on CICS regions can be subdivided into CICS transactions supplied by IBM and transactions installed by CICS users. CICS-supplied transactions can be indicated as those IBM-supplied transactions that start with the letter C. They cannot be modified or deleted. In contrast, the transactions installed by CICS users are, for instance, the MQSeries CICS transactions, and, of course, the transaction NACT of the bank customer account application NACT.
IBM groups its supplied CICS transactions into three default categories. All transactions which CICS needs for internal use are called CAT1-transactions. No user except the CRU STCCICS requires an access to initiate these transactions. In contrast to these CICS-supplied internal transactions there exist two other transaction categories – CAT2- and CAT3-transactions. Transactions which CICS does not require for internal processing and which should not be executed by everybody are called CAT2-transactions. CAT2-transactions can be initiated by CICS users in dependence on their access permission, whereas CAT3-transactions are exempt from any security check. All CICS users must have an access to the CAT3-transactions, therefore, CICS permits all users to use these transactions. Protecting the CICS transactions is done by adding the user/group ID to the access list. Because it is only checked whether a user may initiate the transaction, it is only required a READ access to them.

6.4.4.2 Using security profiles to protect CICS transactions

CICS transactions can be secured by RACF transaction security profiles stored into IBM-supplied or own defined security classes. Single transactions should be secured by profiles stored within a member security class instead of creating a group consisting of this one and only transaction. When the IBM-supplied security classes are used the member security class is called TCICSTRN. Grouped transactions have to be secured by profiles stored within a group security class. IBM provides for them the GCICSTRN security class. Using transaction group profiles simplifies the security administration. The users need only an access to the transaction group to use its transactions.

Each profile and its access list can be added to the RACF classes when submitting appropriate RACF commands on the TSO/E command line. A better and common way is to build TSO/E CLISTS. They can contain special TSO/E commands or complex programming tasks written with the interpretive CLIST language. There are some important statement each CLIST contains. The PROC statement must be always the first command of a CLIST. It is followed by the CONTROL statement which defines the processing options for the CLIST. If the option MAIN is specified the main CLIST follows. ASIS means that the character strings have to be read as they were written, for example if written in lower case the letters will not be converted into upper case. The SET statement assigns values to a symbolic or control variable. The WRITE and WRITENR statements display a message on the screen. Using the WRITE statement causes the cursor to return to the beginning of the next line after the message. When using the WRITENR statement, the cursor remains at the end of the message text (NR is for “No Return”). Further information about TSO/E CLISTS is available from the book [TEC99].
In the case, the SIT-parameters \textit{SEC=YES} and \textit{XTRAN=YES} are set and the CICS region is restarted before any transaction security profile is defined, nobody has an access to initiate a CICS transaction. Hence, there has been firstly defined one generic profile called ** stored to the member security class \textit{TCICSTRN}. This profile gives all CICS users an access to all transactions of the CICS region, the transactions defined on their own, too. Therefore, \textit{UACC=READ} has to be set. However, before any generic profile can be defined to the RACF security class \textit{TCICSTRN} following command must be issued to activate the storing of generic profiles to the member security class \textit{TCICSTRN}:

\textit{Example: \texttt{SETROPTS GENERIC(TCICSTRN)}}

Afterwards, the transaction security profile is created using the RACF command \texttt{RDEFINE}:

\textit{Example: \texttt{RDEFINE TCICSTRN ** UACC(READ)}}

Because all transactions should not be accessible by all CICS users some transactions have been protected against an unauthorised access. Such transactions are for instance the IBM-supplied CICS transactions and the transactions for CICS transactions supplied by MQSeries. They are grouped in different group profiles to be stored to the \textit{GCICSTRN} security class. Although, this class stores profiles for groups of transactions, each transaction itself is secured by the super class \textit{TCICSTRN}. Only three transactions in the CICS region A06C001 are secured using member profiles stored to the \textit{TCICSTRN} class directly. For defining the transaction security profiles to the \textit{TCICSTRN} and \textit{GCICSTRN} transaction security classes there can be used the two example scripts \texttt{DFHSCAT1} and \texttt{DFHSCAT2} stored in the CICS install library \texttt{CICSTS13.CICS.SDFHSAMP}. According to these CLISTS, there have been created five scripts \texttt{CAT1JEDI}, \texttt{CAT2JEDI}, \texttt{CAT3JEDI}, \texttt{MQSJEDI}, and \texttt{USERJEDI} stored in the data set \texttt{CICS.COMMON.RACF}.

The CAT1-transactions have been secured within one group profile specified in the script \texttt{CAT1JEDI}, whereas \texttt{CAT2JEDI} specifies groups containing CAT2-transactions. The \texttt{CAT3JEDI} also contains only one group that secures the CAT3-transactions though they are exempt from any security check per default. A security check for those transactions is only used for the \texttt{QUERY SECURITY Command Security} (see chapter 9 in [RSG03]). \texttt{MQSJEDI} defines one member and two group profiles to secure CICS transactions supplied by MQSeries. Within \texttt{USERJEDI} some user-relevant transactions have been still secured as member profiles. All the profiles are stored into the RACF database and in the main/virtual storage of JEDI OS/390-server.

\subsection{Securing the IBM-supplied CAT1-transactions}

Based on the CLIST \texttt{CAT1JEDI} it is described how to built such a script to add a security profile to the CICS RACF group class (Listing 9 on the next page resp. Listing 56 on page 246). Line 1 of \texttt{CAT1JEDI} starts with the CLIST command \texttt{PROC}. The number behind this command refers to the number of positional parameters to expect, 0 represents none. The \texttt{CONTROL} statement in line 29 introduces the main CLIST statements. Firstly, some variables have been set within the lines 34-37 using the \texttt{SET} statement. The CAT1-transactions to be secured
have been summarised in the group profile *CAT1* created with the RACF command *RDEFINE* (lines 41-50). Within the parameter *ADDMEM* the transaction identifiers are added to the profile *CAT1*. Authorised users have been added to the access list with the RACF command *PERMIT* in line 54. It is not needed to specify the *ACCESS* attribute together with an authority level, in case, a READ access is wanted as the level. Per default RACF assigns the READ access. The authorised user – the CRU STCCICS – has been set in line 37 using the variable *ACCESSLIST*. All messages to be displayed on the screen have been specified to the script using the CLIST commands *WRITE* resp. *WRITENR* (line 52, 56-58). Within the *ADDMEM* parameter there have been listed only the CAT1-transactions available on the CICS region A06C001. However, there are some more CAT1-transactions IBM usually provides within the CICS installation. For more information about these missing transactions and for a description of the function of the listed transactions refer to chapter 10 of [RSG03].

After the first time, this script has been successfully executed, the CICS resource class *TCICSTRN* needs to be activated using the RACF command *SETROPTS CLASSACT* to activate the CICS-supplied transaction security class *TCICSTRN*. Because the profiles should be stored in main/virtual storage the attribute *RACLIST* has been also specified:

*Example:* *SETROPTS CLASSACT(TCICSTRN) RACLIST(TCICSTRN)*

Using this command activates also all other CICS general resource and stores them to the main/virtual storage because they all have the same POSIT-number. Therefore, only one of the member classes needs to be refreshed with the *SETROPTS* command after a profile – group or member – has been modified or added to the RACF.

```
01 PROC 0
29 CONTROL MAIN ASIS
34 SET NOTIFY = NILSM
35 SET OWNER = SYS1
36 SET CLASSNAME = GCICSTRN
37 SET ACCESSLIST = STCCICS
   41 RDEFINE &CLASSNAME CAT1 UACC(NONE) +
      NOTIFY(&NOTIFY) +
      OWNER(&OWNER) +
      ADDMEM(CATA, CATD, CDBD, CDBF, CDBO, +
      CDBQ, CDT5, CEDC, CEDX, CFTS, +
      CIDO, CIOM, CIOR, CIRO, CIRO, +
      CRMN, CRMF, CRSM, CRSY, CSFU, +
      CSSP, CSSP, CSNE, CSPQ, CSQC, +
      CSTE, CSZI, CWBG, CWXN, CXCU, +
      CXML)
   52 WRITENR CICS CAT1-Transactions have been defined to RACF.
   54 PERMIT CAT1 CLASS(&CLASSNAME) ID(&ACCESSLIST)
   56 WRITENR The specified users have now access to the CAT1-Transactions.
   57 WRITE
   58 WRITENR End of CAT1JEDI CLIST.
```

*Listing 9: Extract from the CLIST CAT1JEDI stored in the data set CICS.COMMON.RACF*
database. For CICS general resource classes and POSIT-numbers refer to chapter 6.2.3 “Data set and general resource profiles” on page 164.

### 6.4.4.4 Securing the IBM-supplied CAT2-transactions

The sample `DFHSCAT2` supplied in the library `CICSTS13.CICS.SDFHSAMP` can be used as a template to define all the other transaction security profiles. From that script the CLIST `CAT2JEDI` has been created to secure the CAT2-transactions containing ten group profiles stored to the `GCICSTRN` security class (cf. Listing 55, page 246): `ALLUSER`, `CICSGAST`, `DBCTL`, `DEVELOPER`, `INQUIRE`, `INTERCOM`, `OPERATOR`, `PRAKT`, `SYSADM`, and `WEBUSER`. Table 13 on the next page lists all the profile groups consisting of some transactions defined within the script `CAT2JEDI`. The profiles `DBCTL` and `WEBUSER` have been created to deny any access to transactions specified within. Therefore, no access list has been defined to both profiles. The owner of all the created profiles is the RACF super user group `SYS1` (line 47, Listing 55). The access to the CAT2-transactions have been given in necessity of the needs by the CICS users. The `UACC` parameter in all CAT2-transaction group profiles is set to `NONE` except the one in the profile `ALLUSER`, where the universal access has been set to `READ` (lines 63-66, Listing 55). The three transactions secured with this group profile are needed by all CICS terminal users. The other CAT2-transactions are closed against a universal access. Only users added to the access list of the security profiles with at least a `READ` authority are allowed to execute those transactions protected by the profiles.

The profile `CICSGAST` permits an access to transactions executable by the users of the GAST TSO/E-group (lines 69-73, Listing 55) The GAST-users should only test the CICS environment, hence, they have gotten an access to some special transactions which can demonstrate some capabilities of the system. The user group `PRAKT` has an access to the transactions of the profiles `DEVELOPER` and `PRAKT` (lines 84-88, and 113-117, Listing 55). Access to the transactions of the profile `DEVELOPER` also have the user groups `ADMIN`, and `DIPLOM`. Furthermore, they have an access to the transactions of the profiles `INQUIRE` (lines 91-95, Listing 55), `OPERATOR` (lines 105-109, Listing 55), and `SYSADM` (lines 120-124, Listing 55). The profile `INTERCOM` secures transactions to be used for intercommunication within CICS. For future usage, the user group `STCGRP` containing some of the important user IDs of applications running on OS/390 has been given the authority to initiate these transactions.

Furthermore, there have been defined on the script `CAT2JEDI` two more profiles which should protect transactions in the future. The members `CDBC` (interface menu transaction), `CDBI` (interface inquiry transaction), `CDBM` (interface operator transaction), `CDBT` (interface disconnection transaction) have been specified for the profile `DBCTL` (lines 85-89, Listing 55). These transactions should be protected, but no one needs them at this time, so no one has an access. The last listed profile `WEBUSER` protecting the single transaction CWBA is also protected by RACF but not accessible by any CICS user at this time (lines 137-141, Listing 55).

<table>
<thead>
<tr>
<th>Profile Name</th>
<th>Transaction ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALLUSER</td>
<td>CMAC</td>
<td>CICS Messages And Codes transaction</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------------</td>
<td>-------------------------------------</td>
</tr>
<tr>
<td></td>
<td>CRTX</td>
<td>Routing transaction</td>
</tr>
<tr>
<td></td>
<td>CSGM</td>
<td>Good Morning transaction</td>
</tr>
<tr>
<td>CICSGAST</td>
<td>CEMT</td>
<td>Master terminal transaction</td>
</tr>
<tr>
<td></td>
<td>CEOT</td>
<td>Inquires on user's own terminal transaction</td>
</tr>
<tr>
<td></td>
<td>CMSG</td>
<td>Message switching transaction</td>
</tr>
<tr>
<td></td>
<td>CWTO</td>
<td>Write to operator transaction</td>
</tr>
<tr>
<td>PRAKT</td>
<td>CEMT, CEOT, CMSG, CWTO</td>
<td>see profile CICSGAST</td>
</tr>
<tr>
<td></td>
<td>CEDA</td>
<td>Dynamic add resources transaction</td>
</tr>
<tr>
<td></td>
<td>CEST</td>
<td>Supervisor terminal transaction</td>
</tr>
<tr>
<td>INQUIRE</td>
<td>CDBI</td>
<td>DB2 control interface (DBCTL) inquiry transaction</td>
</tr>
<tr>
<td></td>
<td>CEDC</td>
<td>Transaction to view CICS resource definitions</td>
</tr>
<tr>
<td>OPERATOR</td>
<td>CEOT, CEST, CMSG, CWTO</td>
<td>see profile PRAKT</td>
</tr>
<tr>
<td></td>
<td>CRTE</td>
<td>Start of transaction routing sessions</td>
</tr>
<tr>
<td></td>
<td>CSFE</td>
<td>Test field engineering terminal transaction</td>
</tr>
<tr>
<td></td>
<td>DSNC</td>
<td>DB2 attachment facility transaction</td>
</tr>
<tr>
<td>SYSADM</td>
<td>CEDA, CEMT</td>
<td>see profile PRAKT</td>
</tr>
<tr>
<td></td>
<td>CDBC</td>
<td>DBCTL interface menu transaction</td>
</tr>
<tr>
<td></td>
<td>CESD</td>
<td>Shutdown assist transaction</td>
</tr>
<tr>
<td></td>
<td>CETR</td>
<td>Transaction for inquire and set trace options</td>
</tr>
<tr>
<td></td>
<td>CIND</td>
<td>In-doubt test tool transaction</td>
</tr>
<tr>
<td>INTERCOM</td>
<td>CDFS</td>
<td>Dynamic starts with interval transaction</td>
</tr>
<tr>
<td></td>
<td>CEHP</td>
<td>CICS OS/2 remote server mirror transaction</td>
</tr>
<tr>
<td></td>
<td>CEHS</td>
<td>CICS/VM remote server mirror transaction</td>
</tr>
<tr>
<td></td>
<td>CPMI</td>
<td>CICS OS/2 LU6.2 mirror transaction</td>
</tr>
<tr>
<td></td>
<td>CSMI</td>
<td>ISC mirror transaction</td>
</tr>
<tr>
<td></td>
<td>CSM1</td>
<td>ISC SYSMSG model transaction</td>
</tr>
<tr>
<td></td>
<td>CSM2</td>
<td>ISC scheduler model transaction</td>
</tr>
<tr>
<td></td>
<td>CSM3</td>
<td>ISC queue model transaction</td>
</tr>
<tr>
<td></td>
<td>CSM5</td>
<td>ISC DL/1 model transaction</td>
</tr>
<tr>
<td></td>
<td>CVMI</td>
<td>LU6.2 syncelevel 1 mirror transaction</td>
</tr>
</tbody>
</table>

Table 13: Profiles containing CAT2-transactions specified within the CLIST CAT2JEDI (according to [RSG03], ch. 10)

6.4.4.5 Securing the IBM-supplied CAT3-transactions

The script `CAT3JEDI` defines one transaction security profile called `CAT3` containing the CAT3-transactions (cf. Listing 54, page 246). Because all these transactions must be accessible by all the CICS terminal users the UACC parameter of the profile `CAT3` has been set to READ as same as for the profile `ALLUSER` defined within the script `CAT2JEDI`. Two important CICS transactions of the CAT3-transactions are the Sign-On and Sign-Off transactions with the identifier CESN resp. CESF. If the users do not have an access to these transactions they
cannot be sign on to the CICS terminal resp. sign off from it. Table 14 lists the transactions which have been added to the transaction security profile CAT3:

<table>
<thead>
<tr>
<th>Transaction ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSPP</td>
<td>3270 print support transaction</td>
</tr>
<tr>
<td>CSPG</td>
<td>BMS terminal paging transaction</td>
</tr>
<tr>
<td>CSPS</td>
<td>Scheduler for BMS terminal paging transaction</td>
</tr>
<tr>
<td>CLS1, CLS2, CLS3</td>
<td>ISC LU services model transactions</td>
</tr>
<tr>
<td>CLS4</td>
<td>Transaction to manage password expiry for ISC LU</td>
</tr>
<tr>
<td>CMPX</td>
<td>Transaction to ship ISC local queuing</td>
</tr>
<tr>
<td>CRSR</td>
<td>ISC remote scheduler transaction</td>
</tr>
<tr>
<td>CSSF</td>
<td>Transaction to cancel CRTE (see profile Operator)</td>
</tr>
<tr>
<td>CXRT</td>
<td>Transaction for the routing relay of transactions</td>
</tr>
<tr>
<td>CLQ2</td>
<td>Transaction for the outbound resynchronisation for APPC and MRO</td>
</tr>
<tr>
<td>CLR1</td>
<td>Transaction for the inbound resynchronisation for MRO</td>
</tr>
<tr>
<td>CLR2</td>
<td>Transaction for the inbound CNOS for APPC and MRO</td>
</tr>
<tr>
<td>CRSN</td>
<td>Transaction for the synchronisation of the 3614 message</td>
</tr>
<tr>
<td>CESN</td>
<td>Sign on transaction</td>
</tr>
<tr>
<td>CESF</td>
<td>Sign off transaction</td>
</tr>
<tr>
<td>CEGN</td>
<td>Goodnight transaction</td>
</tr>
<tr>
<td>CATR</td>
<td>Transaction to delete the auto-installed restart terminal</td>
</tr>
<tr>
<td>CQRY</td>
<td>Transaction to provide ATI query support</td>
</tr>
<tr>
<td>CSAC</td>
<td>Transaction to process the program abnormal condition</td>
</tr>
<tr>
<td>CSCY</td>
<td>Transaction to print the 3270 screen</td>
</tr>
<tr>
<td>CSPK</td>
<td>Transaction to provide the 3270 screen print</td>
</tr>
<tr>
<td>CSRK</td>
<td>Transaction to provide the 3270 screen print – release keyboard</td>
</tr>
</tbody>
</table>

Table 14: CAT3-transactions secured by the CLIST CAT3JEDI (according to [RSG03], ch. 10)

6.4.4.6 Securing the MQSeries CICS transactions used for the NACT application

When the transaction security is enabled on the CICS region A06C001 the MQSeries CICS transactions should also be secured. Because at this time, all the CICS transactions are accessible on the CICS region except the IBM-supplied CICS transactions distinguished into CAT1-, CAT2, and CAT3-transactions. Within the script MQSJEDI there are defined two group profiles called MQSeries and PLTMQS, and one member profile for the MQSeries CICS DPL Bridge Task transaction CKBP (cf. Listing 55, page 246). The group profile MQSeries secures the transactions to administer the MQSeries CICS adapter. Table 15 lists all the transactions of the profile MQSeries with their descriptions. These transactions have been authorised for the user groups ADMIN and DIPLOM (lines 35-39, Listing 55).

<table>
<thead>
<tr>
<th>Transaction ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Profile MQSeries:</td>
<td></td>
</tr>
</tbody>
</table>
### Transactions executed from the MQSeries CICS adapter control panels:

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKCN</td>
<td>Connects the MQSeries queue manager and the MQSeries CICS adapter</td>
</tr>
<tr>
<td>CKDL</td>
<td>Starts the line mode display</td>
</tr>
<tr>
<td>CKDP</td>
<td>Transaction for the full screen display</td>
</tr>
<tr>
<td>CKRS</td>
<td>Displays some statistics</td>
</tr>
<tr>
<td>CKSD</td>
<td>Disconnects the MQSeries queue manager from the MQSeries CICS adapter</td>
</tr>
<tr>
<td>CKSQ</td>
<td>Start/stops the MQSeries CICS adapter resp. the transaction CKTI</td>
</tr>
</tbody>
</table>

#### Profile PLTMQS:

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKAM</td>
<td>Handles unscheduled events (pending events) and generates messages to be sent to the terminal console</td>
</tr>
<tr>
<td>CKBR</td>
<td>Starts the MQSeries CICS BMT</td>
</tr>
<tr>
<td>CKTI</td>
<td>Starts the MQSeries CICS Task Initiator</td>
</tr>
</tbody>
</table>

#### Profile CKBP:

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKBP</td>
<td>Starts the MQSeries CICS BDT</td>
</tr>
</tbody>
</table>

### Table 15: MQSeries-supplied CICS transactions secured by the CLIST MQSJEDI

The transaction security profile PLTMQS consists of the three CICS transactions CKAM, CKBR, and CKTI (lines 42-45, Listing 55, and Table 15). These transactions are called when the MQSeries CICS adapter and bridge are started during the CICS region initialisation. CKBR starts the MQSeries CICS Bridge Monitor Task (BMT). In case a failure happens during the process, CKAM – the alert monitor of MQSeries – is called. For example, it reconnects MQSeries and CICS after an MQSeries restart, if the automatic connection is activated. CKTI monitors all the processes of the MQSeries CICS connection, for example it starts resp. monitors MQSeries-supplied CICS transactions, for example when a message is put onto a specific queue. Because all these transactions are referenced during the PLT-program STRTCKBR is loaded, the PLTPIU PLTCICS needs a READ access to them (line 46, Listing 55). For defining the PLTPIU refer to chapter 6.3.5 “The PLTPIU” on page 177. The PLT-program STRTCKBR is explained in chapter 5.4.2.5 “An automatic start job for the MQSeries CICS Bridge” on page 109.

After the MQSeries CICS BMT is started by the transaction CKBR and a message is put onto the MQSeries CICS Bridge queue the MQSeries CICS Bridge DPL Task (BDT) is initiated by the transaction CKBP (see Table 15). In case, no user ID has been specified in the message the MQSeries CICS BDT, the task always runs with the LOCAL level of authentication. Contrary, if a user ID is specified in a message there can be used three more levels – IDENTIFY, VERIFY_UOW, and VERIFY_ALL (refer to [RSG03], ch. 8.3.4.3, “Security considerations for the CICS bridge”). The MQSeries CICS Bridge has been started in the CICS region A06C001 with the LOCAL level of authentication. This means, that each CICS program run by the MQSeries CICS BDT is started with the DCU. Hence, the DCU needs an access to the MQSeries CICS BDT started by the transaction CKBP. This transaction has been secured by the transaction member profile CKBP. Consider, transaction member profiles must be stored in to the transaction security class TCICSTRN. The DCU C001DEF has been added to the
Securing CICS with RACF

access list (lines 50-53, Listing 55). It has also to be verified that the PLTPIU PLTCICS is the surrogate user of the DCU, that is a must (refer to ch. 6.4.3 “The Surrogate User Security”, page 187).

6.4.4.7 Securing the transactions NACT and CSKL

Within the CLIST USERJEDI there have been defined two more member profiles – profile NACT secures the CICS transaction NACT and profile CSKL secures the transaction for the CICS Socket Listener for TCP/IP CSKL (cf. Listing 56, page 246). Both transaction profiles have been stored to the security class TCICSTRN. READ access to the CICS transaction NACT have got the user groups ADMIN, DIPLOM, GAST, and PRAKT (lines 36-39, Listing 56). Hence, they can initiate this transaction from the CICS terminal.

The CICS Socket Listener transaction CSKL that handles all inbound socket requests of the TCP/IP-protocol has to be authorised for the PLTPIU PLTCICS because it is also started by a PLT-program (lines 42-46, Listing 56).

6.4.5 The CICS Command Security

6.4.5.1 The CICS Command Security Mechanism

Due to the problems with the CICS Transaction Security, as described in chapter 6.4.1 “Decision about useful and necessary security mechanism” on page 185, there has been enabled the CICS Command Security. This mechanism secures the so-called SP-type commands. Using the SIT-parameter XCMD=YES enables the CICS Command Security Mechanism which is only activated when the SIT-parameter SEC is also enabled (cf. line 44, Listing 51, page 245). Per default RACF loads the standard command security classes CCICSCMD for the command member profiles and VCICSCMD for the command group profiles. If there should be used own defined command security classes they have to be named on this SIT-parameter.

All the SP-type commands are executable within the CEMT transaction or within a CICS application program that calls a specified CICS resource. SP-type commands used in a CICS application are also named as EXEC CICS commands. The subsets of the CEMT transaction CEBT, CEOT, and CEST use some but not all SP-type commands. The transaction CECI uses the same command-level interpreter as CEMT to enter an EXEC CICS command. All these transactions refer to the command security profiles stored within the CICS Command Security. With the SP-type commands only predefined CICS resources can be accessed. It is only needed to define the required authority level to these CICS resources to get an access to them, which can be one out of these three: READ, UPDATE, and ALTER. Table 16 lists the SP-type commands with the required access to execute them. For example, the INQUIRE command on a CICS resource can only be executed, when a READ access to the required CICS resource is permitted. For descriptions of CICS resources and how to secure them see next chapter.

<table>
<thead>
<tr>
<th>Access permission</th>
<th>Command name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>


CICS Command Security applies also to transactions which have an enabled CMDSEC option in their transaction resource definition. If it is enabled and the transaction initiates an appropriate SP-type command, a command security check is always made on the security profiles stored within the command security classes CCICS\-SCMD resp. VCICSCMD. The CMDSEC parameter can also be set in the SIT globally. When specifying CMDSEC=ALWAYS in the SIT the transaction resource definition CMDSEC is set for all CICS transactions to CMDSEC=YES. However, this is not recommended because “invoking a command security check for every CICS command consumes extra overhead that reduces the performance of all your transactions.” ([RSG03], ch. 8)

Setting CMDSEC=ASIS in the SIT implies that the security check has to obey the CMDSEC option in the transaction resource definition.

It is noticed, that a user who wants to execute an SP-type command within the CEMT, CEBT, CEOT, CEST, CECI transactions as same as within transactions having an enabled CMDSEC option on their transaction resource definition must have the permission to initiate the transaction given by the CICS Transaction Security (see chapter 6.4.4 “The CICS Transaction Security” on page 189).

### 6.4.5.2 Securing predefined CICS resources subject to CICS Command Security

The CLIST COM1JEDI stored in CICS.COMMON.RACF secures all the predefined CICS resources accessible by the SP-type commands listed in Table 16 (cf. Listing 10 on the next page; for a full listing see also Listing 57 on page 246). The script starts with the obligatory CLIST commands PROC and CONTROL (lines 1&70) as described in chapter 6.4.4.2 “Using security profiles to protect CICS transactions” on page 190. With the SET commands (line 74-78) there have been set variables for the:

- owner of the profiles: OWNER=SYS1
- group class to store the profiles in: CLASSNAME=VCICSCMD
- access authority the following user access lists require: ALLUSER\_ACCESS\_LIST, CMDUPD\_ACCESS\_LIST, and CMDALT\_ACCESS\_LIST.

The first security profile ALLUSER is created with the RACF command RDEFINE and is stored to the group class VCICSCMD using the variable CLASSNAME (line 87). The ADDRVT parameter lists the CICS resource identifiers accessible with the READ authority (lines 88-97). Such a READ authority have got the user groups PRAKT, GAST, ADMIN, and DIPLOM (line 100). Hence, all the CICS terminal users can at least execute the SP-
type commands to read a CICS resource. The section ACCESS(READ) in Table 17, which spreads from page 201 to 204, lists the CICS resources that can be accessed by all CICS users using the SP-type commands COLLECT and INQUIRE. Only the predefined CICS resource identifier STATISTICS can be used as an option with the command COLLECT. For all other specified resource identifiers the INQUIRE command is used.

The second security profile is called CMDUPD and makes resources accessible for an update process (line 106, Listing 10). Using the SP-type commands SET, PERFORM, ENABLE, DISABLE, EXTRACT, and RESYNC to access a CICS resource requires the UPDATE authority. On the ADDMEM parameter in line 107 are specified the resource identifiers subject to the UPDATE authority. Access to the specified resources have got only the user groups ADMIN and DIPLOM (line 114). Most of the resources accessible with INQUIRE command can also be accessed with the SET command (Table 17). The PERFORM command can be executed on seven predefined CICS resource identifiers, whereas the SP-type commands DISABLE, ENABLE, EXTRACT, and RE-

Listing 10: Extract from the CLIST COM1JEDI stored in the data set CICS.COMMON.RACF
SYNC can only be executed within some CICS resources which are secured by the resource identifier EXIT-
PROGRAM.

Not all the resource identifiers accessible with the UPDATE authority in the Table 17 has been added to the
member list of the profile CMDUPD because they should also be accessible by the user groups ADMIN and
DIPLOM issuing the two SP-type commands CREATE and DISCARD. For executing these both commands on a
CICS resource they require an ALTER access. The profile which secures these resources is called CMDALT (line
120, Listing 10). The predefined CICS resources added to the security profile are listed within the ADDMEM para-
meter in lines 121-124. Issuing the RACF command PERMIT gives both user groups the ALTER authority
(line 127).

Usually, the resource identifiers specified on the EXEC CICS commands matches the resource identifiers spe-
cified within the CEMT transaction. However, in some cases there are exceptions. These exceptions have been
marked bold in Table 17.

Another exception is the SP-type command CREATE used with the CEMT transaction. This command does
not exist for the transaction. It implies a CEDA INSTALL for which the user must have an ALTER access to the
resource. Hence, all those users can install resources to the CICS region if they may execute the CEDA transac-
tion. That are on the JEDI OS/390-server the user groups PRAKT, DIPLOM, and ADMIN. All these users may
create any resource using the transaction CEDA until this resource has been protected by its associated resource
security mechanism. When the several security mechanisms for the resources have been enabled, for example the
Started Transaction Security, the CICS Program Security, or the Journal Log Security mechanisms, and no secur-
ity profile for the started transactions or the CICS logs has been created, the user may not create any resource us-
ing CEDA. For the CICS programs it has to be enabled the parameter XPPT in the CICS region's SIT, for the
CICS started transactions the parameter XPCT, and for CICS DB2-Entries the parameter XDB2. In case, that
there exists a security profile for the resource and the user has an ALTER access to the resource profile, the user
can create it. Therefore, the creation of a resource does not depend on the permission the user has to the SP-type
commands, but it depends on the permission to the additional resource security profiles. However, they cannot
delete resp. discard the resource from the CSD because this procedure can only be done within the CEMT trans-
action and its SP-type command DISCARD. For using this command the user needs an ALTER access to the re-
sources subject to the SP-type command DISCARD, a lower access level blocks the creation.

Hence, the creation of a resource is always permitted until an explicit resource profile has been defined using
the resource security classes as described in chapter 6.4.1 “Decision about useful and necessary security mechan-
ism” on page 185. For further descriptions on how to activate the additional security mechanisms and create the
security profiles refer to [RSG03]; begin at chapter 6 “Resource security”.
<table>
<thead>
<tr>
<th>Resource to protect</th>
<th>EXEC CICS command</th>
<th>CEMT command</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATISTICS</td>
<td>COLLECT STATISTICS</td>
<td>----</td>
<td></td>
</tr>
<tr>
<td>AUTINSTMODEL</td>
<td>INQUIRE AUTINSTMODEL</td>
<td>INQUIRE AUTINSTMODEL</td>
<td></td>
</tr>
<tr>
<td>AUTOINSTALL</td>
<td>INQUIRE AUTOINSTALL</td>
<td>INQUIRE AUTOINSTALL</td>
<td></td>
</tr>
<tr>
<td>CFDTPOOL</td>
<td>INQUIRE CFDTPOOL</td>
<td>INQUIRE CFDTPOOL</td>
<td></td>
</tr>
<tr>
<td>CONNECTION</td>
<td>INQUIRE CONNECTION</td>
<td>INQUIRE CONNECTION</td>
<td></td>
</tr>
<tr>
<td>DB2CONN</td>
<td>INQUIRE DB2CONN</td>
<td>INQUIRE DB2CONN</td>
<td></td>
</tr>
<tr>
<td>DB2ENTRY</td>
<td>INQUIRE DB2ENTRY</td>
<td>INQUIRE DB2ENTRY</td>
<td></td>
</tr>
<tr>
<td>DB2TRAN</td>
<td>INQUIRE DB2TRAN</td>
<td>INQUIRE DB2TRAN</td>
<td></td>
</tr>
<tr>
<td>DELETESHIPPED</td>
<td>INQUIRE DELETESHIPPED</td>
<td>INQUIRE DELETESHIPPED</td>
<td></td>
</tr>
<tr>
<td>DOCTEMPATEL</td>
<td>INQUIRE DOCTEMPATEL</td>
<td>INQUIRE DOCTEMPATEL</td>
<td></td>
</tr>
<tr>
<td>DSNNAME</td>
<td>INQUIRE DSNNAME</td>
<td>INQUIRE DSNNAME</td>
<td></td>
</tr>
<tr>
<td>DUMPDS</td>
<td>INQUIRE DUMPDS</td>
<td>INQUIRE DUMPDS</td>
<td></td>
</tr>
<tr>
<td>ENQMODEL</td>
<td>INQUIRE ENQMODEL</td>
<td>INQUIRE ENQMODEL</td>
<td></td>
</tr>
<tr>
<td>EXITPROGRAM</td>
<td>INQUIRE EXITPROGRAM</td>
<td>----</td>
<td></td>
</tr>
<tr>
<td>FILE</td>
<td>INQUIRE FILE</td>
<td>INQUIRE FILE</td>
<td></td>
</tr>
<tr>
<td>IRBATCH</td>
<td>INQUIRE IRBATCH</td>
<td>INQUIRE IRBATCH</td>
<td>Both can be used</td>
</tr>
<tr>
<td>IRC</td>
<td>INQUIRE IRC</td>
<td>INQUIRE IRC</td>
<td></td>
</tr>
<tr>
<td>JOURNALMODEL</td>
<td>INQUIRE JOURNALMODEL</td>
<td>INQUIRE JMODEL</td>
<td>Spelling!</td>
</tr>
<tr>
<td>JOURNALNUM</td>
<td>INQUIRE JOURNALNAME</td>
<td>INQUIRE JOURNALNAME</td>
<td>Resource name!</td>
</tr>
<tr>
<td>LINE</td>
<td>----</td>
<td>INQUIRE LINE</td>
<td></td>
</tr>
<tr>
<td>MODENAME</td>
<td>INQUIRE MODENAME</td>
<td>INQUIRE MODENAME</td>
<td></td>
</tr>
<tr>
<td>MONITOR</td>
<td>INQUIRE MONITOR</td>
<td>INQUIRE MONITOR</td>
<td></td>
</tr>
<tr>
<td>PARTNER</td>
<td>INQUIRE PARTNER</td>
<td>INQUIRE PARTNER</td>
<td></td>
</tr>
<tr>
<td>PROCESSTYPE</td>
<td>INQUIRE PROCESSTYPE</td>
<td>INQUIRE PROCESSTYPE</td>
<td></td>
</tr>
<tr>
<td>PROFILE</td>
<td>INQUIRE PROFILE</td>
<td>INQUIRE PROFILE</td>
<td></td>
</tr>
<tr>
<td>PROGRAM</td>
<td>INQUIRE PROGRAM</td>
<td>INQUIRE PROGRAM</td>
<td></td>
</tr>
<tr>
<td>REQID</td>
<td>INQUIRE REQID</td>
<td>----</td>
<td></td>
</tr>
<tr>
<td>REQUESTMODEL</td>
<td>INQUIRE REQUESTMODEL</td>
<td>INQUIRE REQUESTMODEL</td>
<td></td>
</tr>
<tr>
<td>RRMS</td>
<td>INQUIRE RRMS</td>
<td>INQUIRE RRMS</td>
<td></td>
</tr>
<tr>
<td>STATISTICS</td>
<td>INQUIRE STATISTICS</td>
<td>INQUIRE STATISTICS</td>
<td></td>
</tr>
<tr>
<td>STORAGE</td>
<td>INQUIRE STORAGE</td>
<td>----</td>
<td></td>
</tr>
<tr>
<td>STREAMNAME</td>
<td>INQUIRE STREAMNAME</td>
<td>INQUIRE STREAMNAME</td>
<td></td>
</tr>
<tr>
<td>SUBPOOL</td>
<td>INQUIRE SUBPOOL</td>
<td>----</td>
<td></td>
</tr>
<tr>
<td>SYSDUMPCLASS</td>
<td>INQUIRE SYSDUMPCLASS</td>
<td>INQUIRE SYSDUMPCLASS</td>
<td>Spelling!</td>
</tr>
<tr>
<td>SYSTEM</td>
<td>----</td>
<td>INQUIRE DSAS</td>
<td>These options require access to the same resource</td>
</tr>
<tr>
<td>TASK</td>
<td>INQUIRE TASK</td>
<td>INQUIRE TASK</td>
<td>Both can be used</td>
</tr>
<tr>
<td>TASK LIST</td>
<td>----</td>
<td>----</td>
<td></td>
</tr>
<tr>
<td>TCLASS</td>
<td>INQUIRE TCLASS</td>
<td>INQUIRE TCLASS</td>
<td>Both can be used</td>
</tr>
<tr>
<td>TCPIPSERVICE</td>
<td>INQUIRE TCPIPSERVICE</td>
<td>INQUIRE TCPIPSERVICE</td>
<td></td>
</tr>
<tr>
<td>TDQUEUE</td>
<td>INQUIRE QUEUE</td>
<td>INQUIRE TDQUEUE</td>
<td>Both can be used</td>
</tr>
<tr>
<td>TERMINAL</td>
<td>INQUIRE NETNAME</td>
<td>INQUIRE NETNAME</td>
<td>These options require access to the same resource</td>
</tr>
<tr>
<td></td>
<td>INQUIRE TERMINAL</td>
<td>INQUIRE TERMINAL</td>
<td></td>
</tr>
</tbody>
</table>

Table 17: The CICS resources accessible by the SP-type commands (continued on next page)
<table>
<thead>
<tr>
<th>Resource to protect</th>
<th>EXEC CICS command</th>
<th>CEMT command</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>TERMINAL</td>
<td>INQUIRE NETNAME</td>
<td>INQUIRE NETNAME</td>
<td>These options require access to the same resource</td>
</tr>
<tr>
<td></td>
<td>INQUIRE TERMINAL</td>
<td>INQUIRE TERMINAL</td>
<td></td>
</tr>
<tr>
<td>TRACEDEST</td>
<td>----</td>
<td>INQUIRE AUXTRACE</td>
<td>These options require access to the same resource</td>
</tr>
<tr>
<td></td>
<td>----</td>
<td>INQUIRE GTFTRACE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INQUIRE TRACEDEST</td>
<td>----</td>
<td></td>
</tr>
<tr>
<td>TRACEFLAG</td>
<td>INQUIRE TRACEFLAG</td>
<td>----</td>
<td></td>
</tr>
<tr>
<td>TRACETYPE</td>
<td>INQUIRE TRACETYPE</td>
<td>----</td>
<td></td>
</tr>
<tr>
<td>TRANDUMPCODE</td>
<td>INQUIRE TRANDUMPCODE</td>
<td>INQUIRE TRDUMPCODE</td>
<td>Spelling!</td>
</tr>
<tr>
<td>TRANSACTION</td>
<td>INQUIRE TRANSACTION</td>
<td>INQUIRE TRANSACTION</td>
<td></td>
</tr>
<tr>
<td>TSMODEL</td>
<td>INQUIRE TSMODEL</td>
<td>INQUIRE TSMODEL</td>
<td></td>
</tr>
<tr>
<td>TSPPOOL</td>
<td>INQUIRE TSPPOOL</td>
<td>INQUIRE TSPPOOL</td>
<td></td>
</tr>
<tr>
<td>TSQUEUE</td>
<td>INQUIRE TSQNAME</td>
<td>INQUIRE TSQNAME</td>
<td>Both can be used</td>
</tr>
<tr>
<td></td>
<td>INQUIRE TSQUEUE</td>
<td>INQUIRE TSQUEUE</td>
<td></td>
</tr>
<tr>
<td>UOW</td>
<td>INQUIRE UOW</td>
<td>INQUIRE UOW</td>
<td></td>
</tr>
<tr>
<td>UOWDSNFAIL</td>
<td>INQUIRE UOWDSNFAIL</td>
<td>INQUIRE UOWDSNFAIL</td>
<td></td>
</tr>
<tr>
<td>UOWENQ</td>
<td>INQUIRE ENQ</td>
<td>INQUIRE ENQ</td>
<td>Both can be used</td>
</tr>
<tr>
<td>UOWLINK</td>
<td>INQUIRE UOWLINK</td>
<td>INQUIRE UOWLINK</td>
<td></td>
</tr>
<tr>
<td>VOLUME</td>
<td>INQUIRE VOLUME</td>
<td>----</td>
<td></td>
</tr>
<tr>
<td>VTAM</td>
<td>INQUIRE VTAM</td>
<td>INQUIRE VTAM</td>
<td></td>
</tr>
<tr>
<td>WEB</td>
<td>INQUIRE WEB</td>
<td>INQUIRE WEB</td>
<td></td>
</tr>
</tbody>
</table>

**ACCESS (UPDATE)**

| EXITPROGRAM         | ENABLE PROGRAM   | ---- | All options require access to the specified resource. |
|                     | DISABLE PROGRAM  | ---- | |
|                     | EXTRACT EXIT     | ---- | |
|                     | RESYNC ENTRYNAME | ---- | |
| CONNECTION          | PERFORM ENDAFFINITY NETNAME | PERFORM ENDAFFINITY NETNAME | |
| DELETESHIPPED       | PERFORM DELETESHIPPED | PERFORM DELETESHIPPED | |
| DUMP                | PERFORM DUMP     | ---- | Both can be used |
|                     | PERFORM SNAP     | ---- | |
| RESESTTIME          | PERFORM RESESTTIME | PERFORM RESET | |
| SECURITY            | PERFORM SECURITY | PERFORM SECURITY (REBUILD) | REBUILD in the CEMT transaction is the default - it can be omitted! |
| SHUTDOWN            | PERFORM SHUTDOWN | PERFORM SHUTDOWN | |
| STATISTICS          | PERFORM STATISTICS RECORD | PERFORM STATISTICS (RECORD) | RECORD in the CEMT transaction is the default - it can be omitted! |
| AUTOINSTALL         | SET AUTOINSTALL  | SET AUTOINSTALL | |
| CONNECTION          | SET CONNECTION   | SET CONNECTION | |
| DB2CONN             | SET DB2CONN      | SET DB2CONN | |
| DB2ENTRY            | SET DB2ENTRY     | SET DB2ENTRY | |
| DB2TRAN             | SET DB2TRAN      | SET DB2TRAN | |
| DELETESHIPPED       | SET DELETESHIPPED | SET DELETESHIPPED | |
| DSNNAME             | SET DSNNAME      | SET DSNNAME | |
| DUMPS               | SET DUMPS        | SET DUMPS | |
Table 17: The CICS resources accessible by the SP-type commands (continued on next page)

<table>
<thead>
<tr>
<th>Resource to protect</th>
<th>EXEC CICS command</th>
<th>CEMT command</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENQMODEL</td>
<td>SET ENQMODEL</td>
<td>SET ENQMODEL</td>
<td></td>
</tr>
<tr>
<td>FILE</td>
<td>SET FILE</td>
<td>SET FILE</td>
<td></td>
</tr>
<tr>
<td>IRC</td>
<td>SET IRC</td>
<td>SET IRC</td>
<td></td>
</tr>
<tr>
<td>JOURNALNUM</td>
<td>SET JOURNALNAME</td>
<td>SET JOURNALNAME</td>
<td>Spelling!</td>
</tr>
<tr>
<td>LINE</td>
<td>SET LINE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MODENAME</td>
<td>SET MODENAME</td>
<td>SET MODENAME</td>
<td></td>
</tr>
<tr>
<td>MONITOR</td>
<td>SET MONITOR</td>
<td>SET MONITOR</td>
<td></td>
</tr>
<tr>
<td>PROCESSTYPE</td>
<td>SET PROCESSTYPE</td>
<td>SET PROCESSTYPE</td>
<td></td>
</tr>
<tr>
<td>PROGRAM</td>
<td>SET PROGRAM</td>
<td>SET PROGRAM</td>
<td></td>
</tr>
<tr>
<td>STATISTICS</td>
<td>SET STATISTICS</td>
<td>SET STATISTICS</td>
<td></td>
</tr>
<tr>
<td>SYSDUMPCODE</td>
<td>SET SYSDUMPCODE</td>
<td>SET SYSDUMPCODE</td>
<td>Spelling!</td>
</tr>
<tr>
<td>SYSTEM</td>
<td>SET SYSTEM</td>
<td>SET SYSTEM</td>
<td></td>
</tr>
<tr>
<td>TASK</td>
<td>SET TASK</td>
<td>SET TASK</td>
<td></td>
</tr>
<tr>
<td>TCLASS</td>
<td>SET TCLASS</td>
<td>SET TCLASS</td>
<td></td>
</tr>
<tr>
<td>TCPPIP</td>
<td>SET TCPPIP</td>
<td>SET TCPPIP</td>
<td></td>
</tr>
<tr>
<td>TCPIPSERVICE</td>
<td>SET TCPIPSERVICE</td>
<td>SET TCPIPSERVICE</td>
<td></td>
</tr>
<tr>
<td>TDQUEUE</td>
<td>SET QUEUE</td>
<td>SET TDQUEUE</td>
<td></td>
</tr>
<tr>
<td>TERMINAL</td>
<td>SET NETNAME</td>
<td>SET NETNAME</td>
<td></td>
</tr>
<tr>
<td>TRANSACTION</td>
<td>SET TRANSACTION</td>
<td>SET TRANSACTION</td>
<td></td>
</tr>
<tr>
<td>TSQUEUE</td>
<td>SET TSQUEUE</td>
<td>SET TSQUEUE</td>
<td></td>
</tr>
<tr>
<td>TRACEDEST</td>
<td>SET AUXTRACE</td>
<td>SET GTFTRACE</td>
<td></td>
</tr>
<tr>
<td>TRACETYPE</td>
<td>SET TRACETYPE</td>
<td>SET INTRACE</td>
<td></td>
</tr>
<tr>
<td>TRANDUMPCODE</td>
<td>SET TRANDUMPCODE</td>
<td>SET TRDUMPCODE</td>
<td></td>
</tr>
<tr>
<td>TRANSACTION</td>
<td>SET TRANSACTION</td>
<td>SET TRANSACTION</td>
<td></td>
</tr>
<tr>
<td>TSQUEUE</td>
<td>SET TSQUEUE</td>
<td>SET TSQUEUE</td>
<td></td>
</tr>
<tr>
<td>UOW</td>
<td>SET UOW</td>
<td>SET UOW</td>
<td></td>
</tr>
<tr>
<td>UOWLINK</td>
<td>SET UOWLINK</td>
<td>SET UOWLINK</td>
<td></td>
</tr>
<tr>
<td>VTAM</td>
<td>SET VTAM</td>
<td>SET VTAM</td>
<td></td>
</tr>
<tr>
<td>WEB</td>
<td>SET WEB</td>
<td>SET WEB</td>
<td></td>
</tr>
</tbody>
</table>

**ACCESS (ALTER)**

| CONNECTION          | CREATE CONNECTION | CREATE CONNECTION |         |
| DB2CONN             | CREATE DB2CONN    | CREATE DB2CONN   |         |
| DB2ENTRY            | CREATE DB2ENTRY   | CREATE DB2ENTRY  |         |
| DB2TRAN             | CREATE DB2TRAN    | CREATE DB2TRAN   |         |
| DOCTEMPLATE         | CREATE DOCTEMPLATE | CREATE DOCTEMPLATE |         |
| ENQMODEL            | CREATE ENQMODEL   | CREATE ENQMODEL  |         |
| FILE                | CREATE FILE       | CREATE FILE      |         |
| JOURNALMODEL        | CREATE JOURNALMODEL | CREATE JOURNALMODEL |         |
| LSRPOOL             | CREATE LSRPOOL    | CREATE LSRPOOL   |         |
| MAPSET              | CREATE MAPSET     | CREATE MAPSET    |         |
Table 17: The CICS resources accessible by the SP-type commands (continued on next page)

<table>
<thead>
<tr>
<th>Resource to protect</th>
<th>EXEC CICS command</th>
<th>CEMT command</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARTITIONSET</td>
<td>CREATE PARTITIONSET</td>
<td>CREATE PARTITIONSET</td>
<td></td>
</tr>
<tr>
<td>PARTNER</td>
<td>CREATE PARTNER</td>
<td>CREATE PARTNER</td>
<td></td>
</tr>
<tr>
<td>PROCESSSTYPE</td>
<td>CREATE PROCESSSTYPE</td>
<td>CREATE PROCESSSTYPE</td>
<td></td>
</tr>
<tr>
<td>PROFILE</td>
<td>CREATE PROFILE</td>
<td>CREATE PROFILE</td>
<td></td>
</tr>
<tr>
<td>PROGRAM</td>
<td>CREATE PROGRAM</td>
<td>CREATE PROGRAM</td>
<td></td>
</tr>
<tr>
<td>REQUESTMODEL</td>
<td>CREATE REQUESTMODEL</td>
<td>CREATE REQUESTMODEL</td>
<td></td>
</tr>
<tr>
<td>TCPIPREFERENCE</td>
<td>CREATE TCPIPREFERENCE</td>
<td>CREATE TCPIPREFERENCE</td>
<td></td>
</tr>
<tr>
<td>TDQUEUE</td>
<td>CREATE TDQUEUE</td>
<td>CREATE TDQUEUE</td>
<td></td>
</tr>
<tr>
<td>TERMINAL</td>
<td>CREATE TERMINAL</td>
<td>CREATE TERMINAL</td>
<td></td>
</tr>
<tr>
<td>TRANCLASS</td>
<td>CREATE TRANCLASS</td>
<td>CREATE TRANCLASS</td>
<td></td>
</tr>
<tr>
<td>TRANSACTION</td>
<td>CREATE TRANSACTION</td>
<td>CREATE TRANSACTION</td>
<td></td>
</tr>
<tr>
<td>TSMODEL</td>
<td>CREATE TSMODEL</td>
<td>CREATE TSMODEL</td>
<td></td>
</tr>
<tr>
<td>TYPETERM</td>
<td>CREATE TYPETERM</td>
<td>CREATE TYPETERM</td>
<td></td>
</tr>
<tr>
<td>AUTINSTMODEL</td>
<td>DISCARD AUTINSTMODEL</td>
<td>DISCARD AUTINSTMODEL</td>
<td></td>
</tr>
<tr>
<td>CONNECTION</td>
<td>DISCARD CONNECTION</td>
<td>DISCARD CONNECTION</td>
<td></td>
</tr>
<tr>
<td>DB2CONN</td>
<td>DISCARD DB2CONN</td>
<td>DISCARD DB2CONN</td>
<td></td>
</tr>
<tr>
<td>DB2ENTRY</td>
<td>DISCARD DB2ENTRY</td>
<td>DISCARD DB2ENTRY</td>
<td></td>
</tr>
<tr>
<td>DB2TRAN</td>
<td>DISCARD DB2TRAN</td>
<td>DISCARD DB2TRAN</td>
<td></td>
</tr>
<tr>
<td>DOCTEMPLATE</td>
<td>DISCARD DOCTEMPLATE</td>
<td>DISCARD DOCTEMPLATE</td>
<td></td>
</tr>
<tr>
<td>ENQMODEL</td>
<td>DISCARD ENQMODEL</td>
<td>DISCARD ENQMODEL</td>
<td></td>
</tr>
<tr>
<td>FILE</td>
<td>DISCARD FILE</td>
<td>DISCARD FILE</td>
<td></td>
</tr>
<tr>
<td>JOURNALMODEL</td>
<td>DISCARD JOURNALMODEL</td>
<td>DISCARD JMODEL</td>
<td>Spelling!</td>
</tr>
<tr>
<td>JOURNALNUM</td>
<td>DISCARD JOURNALNAME</td>
<td>DISCARD JOURNALNAME</td>
<td>Spelling!</td>
</tr>
<tr>
<td>PARTNER</td>
<td>DISCARD PARTNER</td>
<td>DISCARD PARTNER</td>
<td></td>
</tr>
<tr>
<td>PROCESSSTYPE</td>
<td>DISCARD PROCESSSTYPE</td>
<td>DISCARD PROCESSSTYPE</td>
<td></td>
</tr>
<tr>
<td>PROFILE</td>
<td>DISCARD PROFILE</td>
<td>DISCARD PROFILE</td>
<td></td>
</tr>
<tr>
<td>PROGRAM</td>
<td>DISCARD PROGRAM</td>
<td>DISCARD PROGRAM</td>
<td></td>
</tr>
<tr>
<td>REQUESTMODEL</td>
<td>DISCARD REQUESTMODEL</td>
<td>DISCARD REQUESTMODEL</td>
<td></td>
</tr>
<tr>
<td>TCPIPREFERENCE</td>
<td>DISCARD TCPIPREFERENCE</td>
<td>DISCARD TCPIPREFERENCE</td>
<td></td>
</tr>
<tr>
<td>TDQUEUE</td>
<td>DISCARD QUEUE</td>
<td>DISCARD QUEUE</td>
<td>Both can be used</td>
</tr>
<tr>
<td>TERMINAL</td>
<td>DISCARD TERMINAL</td>
<td>DISCARD TERMINAL</td>
<td></td>
</tr>
<tr>
<td>TRANCLASS</td>
<td>DISCARD TRANCLASS</td>
<td>DISCARD TCLASS</td>
<td>Spelling!</td>
</tr>
<tr>
<td>TRANSACTION</td>
<td>DISCARD TRANSACTION</td>
<td>DISCARD TRANSACTION</td>
<td></td>
</tr>
<tr>
<td>TSMODEL</td>
<td>DISCARD TSMODEL</td>
<td>DISCARD TSMODEL</td>
<td></td>
</tr>
</tbody>
</table>

Table 17: The CICS resources accessible by the SP-type commands
(corrected version in contrast to [RSG03], ch. 8)
6.5 Authorising access to the CICS region

As an additional security option the CICS region must also be protected against its assigned application identifier. The access to the CICS region A06C001 is restricted by a security profile stored in the IBM-supplied RACF general resource class APPL. This profile must have the same name as the application identifier specified within the SIT-parameter APPLID. Each time the CICS region calls RACF to verify a sign-on, the CICS region's APPLID is passed to RACF. The security manager checks whether such a profile exists, and if yes, it is checked whether the user is permitted to access it. If the user may access the profile, RACF allows the sign-on to the CICS region.

However, the application identifier must be firstly defined in the CICS region's SIT definition script C001 within the APPLID parameter (cf. Listing 51, page 245):

```
Script_C001:
  08 APPLID=A06C001, APPLICATION NAME OF THE CICS REGION
```

As next, the APPL general resource profile A06C001 has to be defined to the resource class APPL using the RACF command RDEFINE:

```
RDEFINE APPL CICS_region_appl_id NOTIFY(user_id) OWNER(owner_id) +
UACC(NONE)

Example: RDEFINE APPL A06C001 NOTIFY(TBUSSE) OWNER(IBMUSER) +
UACC(NONE)
```

Following users/groups need an access to the APPL general resource profile A06C001 – ADMIN, DIPLOM, PRAKT, and GAST as groups and the user IDs STCCICS, C001DEF, PLTCICS, and additionally STCJGATE as the user ID for the CICS Transaction Gateway. If the CRU and DCU have not been added to the access list of the profile A06C001 it is impossible to start the CICS region. The RACF command PERMIT adds the users/groups to the access list of the APPL general resource profile. The users/groups need only a READ access to the profile:

```
PERMIT A06C001 Class(APPL) ID(ADMIN,DIPLOM,GAST,PRAKT, +
C001DEF,STCCICS,PLTCICS,STCJGATE) ACCESS(READ)
```

After the general resource profile has been added and the access list for it is created, the information of the profile can be listed with the RACF command RLIST:

```
RLIST class_name profile_name all
```

```
Example: RLIST APPL A06C001 all
```

Before the profile can be used it has to be activated if it was not done previously:
SETROPTS CLASSACT(APPL) or
SETROPTS CLASSACT(APPL) RACLIST(APPL)

When the profile A06C001 is saved to the main/virtual storage (already defined and activated), it is not necessary to activate the class again, it should only be refreshed:

SETROPTS RACLIST(APPL) REFRESH
6.6 Adjust the LOGIN terminal to pass capital letters to RACF

RACF recognises only upper case alphabetic characters. However, when a user wants to sign on to the CICS region, it could be possible that the data is passed to RACF as entered without converting the characters into upper case. The result is, that RACF denies the access. User IDs and passwords specified in the CESN transaction are only translated to upper case when the \textit{UCTRAN(YES)} attribute is specified in the PROFILE resource definition of the transaction CESN.

For converting the characters into upper case a new profile called \textit{AAACICST} has been created for the sign on transaction CESN. This profile has been copied from the original profile \textit{DFHCICST} that CESN used by then. To get the information which profile the transaction CESN uses the transaction definition is searched for the profile name using following command:

\texttt{CEDA DISP TRANS(CESN) GR(*)}

Figure 78 on the next page lists the transaction definition CESN stored to the group \textit{DFHSIGN} which is opened using the command \texttt{VIEW (shortcut: v)}. This panel is searched for the profile the transaction uses – \textit{DFHCICST} (Figure 79, next page). Both – CESN as a CICS supplied transaction stored in the group \textit{DFHSIGN} and \textit{DFHCICST} as a CICS supplied profile stored in the group \textit{DFHSTAND} – cannot be modified until they have been copied into a new group. CICS does not allow any modifying of resources beginning with the string DFH or stored to groups beginning with that string. After the profile name is known, the transaction CESN as CESN and the profile definition \textit{DFHCICST} as \textit{AAACICST} have been copied into a new CICS resource group, for example \textit{TYPENEU} (Figure 80 & Figure 81, both on page 209). If such a CICS resource group does not yet exist, CICS creates it new by taking the letters from the parameter \texttt{to(...)}.

When issuing \texttt{CEDA DISPLAY GR(TYPENEU)} the content of the new group \textit{TYPENEU} is displayed (Figure 82, page 210). The profile \textit{AAACICST} can be modified when the command \texttt{ALTER (shortcut: a)} is specified behind the resource name. This opens the alter panel to set the \textit{UCTRAN} attribute to \texttt{YES} (Figure 83, page 210). Afterwards, this profile name has to be specified to the transaction definition of CESN on its parameter \texttt{PROFILE} (Figure 84, page 211). Before the \textit{UCTRAN} parameter is activated, both resource definitions need to be installed to the CSD using the \texttt{CEDA INSTALL GROUP} command. CICS now converts all lower cases characters to upper case on the sign-on terminal.
Generation of a Java front end for a standalone CICS application accessed through MQSeries &
Securing CICS with RACF

---

**Figure 78: Search for the transaction CESN – CEDA TRANS(CESN) DISP GR(*)**

<table>
<thead>
<tr>
<th>NAME</th>
<th>TYPE</th>
<th>GROUP</th>
<th>DATE</th>
<th>TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>CESN</td>
<td>TRANSACTION</td>
<td>DFHSIGN</td>
<td>99.232</td>
<td>12.12.27</td>
</tr>
</tbody>
</table>

**OBJECT CHARACTERISTICS**

CEDA View Transaction( CESN )

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Group</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CESN</td>
<td>DFHSIGN</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PROGram</th>
<th>TWasize</th>
<th>PROFile</th>
<th>TASKDATALoc</th>
<th>TASKDATAKey</th>
<th>STorageclear</th>
<th>Runaway</th>
<th>Shutdown</th>
<th>Isolate</th>
<th>Brexit</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFHSNP</td>
<td>00000</td>
<td>DFHCICST</td>
<td>Below</td>
<td>Cics</td>
<td>No</td>
<td>System</td>
<td>Disabled</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 79: Display the transaction CEDA using DISP TRANS(CESN) GR(DFHSIGN)**
disp trans(CESN) gr(*)
ENTER COMMANDS
NAME TYPE GROUP DATE TIME
CESN TRANSACTION DFHSIGN c as(CESN) to(TYPENEU) 99.232 12.12.27

RESULTS: 1 TO 1 OF 1
TIME: 17.58.41 DATE: 03.265
PF 1 HELP 3 END 4 TOP 5 BOT 6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL

SYSID=C001 APPLID=A06C001

Figure 80: Copy the transaction definition to the new group TYPENEU

disp profile(DFHCICST) gr(*)
ENTER COMMANDS
NAME TYPE GROUP DATE TIME
DFHCICST PROFILE DFHSTAND c as(AAACICST) to(TYPENEU) 99.232 12.12.29

RESULTS: 1 TO 1 OF 1
TIME: 18.01.14 DATE: 03.265
PF 1 HELP 3 END 4 TOP 5 BOT 6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL

SYSID=C001 APPLID=A06C001

Figure 81: Copy the profile definition to the new group TYPENEU as AAACICST
**Figure 82: Display the contents of the group TYPENEU**

```
disp gr(TYPENEU)
ENTER COMMANDS
NAME     TYPE         GROUP                                   DATE   TIME
AAACICST PROFILE      TYPENEU  a                              03.265 18.01.16
CESN     TRANSACTION  TYPENEU                                 03.265 17.48.51

PF 1 HELP       3 END 4 TOP 5 BOT 6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
```

**Figure 83: Modify UCTRAN from NO to YES**

```
OBJECT CHARACTERISTICS                                   CICS RELEASE = 0530
CEDA  Al ter PRO file( AAACICST )
PROFILE        : AAACICST
Group          : TYPENEU
Description    : PROFILE FOR SIGN-ON SCREEN
Scrnsize       : Default            Default | Alternate
  Uctran        : Yes                No | Yes
MODename       :
Facilitylike   :
PRIntercomp    : No                 No | Yes
JOURNALLING    
  Journal       : No                  No | 1-99
  MSGJrnl      : No                  No | INPut | Output | INOut
PROTECTION     
  MSGInteg      : No                  No | Yes
  Onewte       : No                  No | Yes
  PROtect      : No                  No | Yes
  Chaincontrol : No                  No | Yes
  PROTOCOLS     

PF 1 HELP 2 COM 3 END 6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
```
Figure 84: Modify the profile name to the new one
7 SUMMARY AND FURTHER WORK

7.1 Summary

In this master thesis we have built two operational on-line business applications called NACT and MQNACT. Both represent a little clip of a customer account program how it could be used in a bank to create, read, update, and delete a customer account and its information (MQNACT only supports reading an account record). Additionally, the account records can be browsed by NACT for the names of the customers. All these customer accounts are saved to a database-like file on an S/390 architectural computer system. Like each application it consists of a business logic and a presentation logic part. Stored on the OS/390-server the programs of the business logic written in COBOL are accessed on-line using two different methods. This on-line accessing is handled by IBM's OLTP system CICS which runs on the OS/390-server. Therefore, the business logic programs consist of some special CICS COBOL commands. The first method uses the transaction monitor itself to display the customer information. The results are displayed by a CICS presentation logic program, also written in COBOL, on the CICS terminal screen using the 3270 interface. Within the second technique, presentation logic programs written in JAVA access the customer accounts on the OS/390-server using IBM's middleware product MQSeries. The results of this request can be displayed on each computer system that supports JAVA resp. has installed a JVM, here, the WINDOWS2000 environment has been used.

Because the programs for the CICS application NACT are provided by a CD – pure and compiled – they have been uploaded onto the OS/390-server using the FTP protocol. The text then describes both – the presentation and the business logic of the CICS application. The CICS presentation logic consists of the programs NACT01 and NACT03. NACT01 is used to display the results on the CICS terminal screen, whereas NACT03 can be used to print the customer information. However, the presentation logic not only consists of these two CICS COBOL programs. An additional file, a map set called NACTSET, is needed to create the design of the CICS application for the CICS terminal screen. This map set is written with BMS. When the required data is entered in the input fields of NACT on a CICS terminal it is sent by NACT01 to the business logic programs where the request is processed. As a result, an information appropriate to the request is sent back to NACT01 and is listed on the terminal screen using the “map set mask”. NACT03 also uses the map set to send the response data to a printer connected with the OS/390-server. The three programs of the business logic are distinguished into: NACT02 – the so-called CRUD program, that creates, reads, updates, and deletes an account; NACT05 – the browse program to search accounts by the customer's name. As part of the CICS business logic, the error handling program NACT04 takes up an exceptional position. This program is always called in case of a sudden occurred problem whether it is initi-
ated by the presentation logic programs \textit{NACT01} or \textit{NACT03} or by the business logic programs \textit{NACT02} and \textit{NACT05}.

For exchanging the data between the CICS COBOL programs there are used different storage areas of OS/390. The most important and interesting storage area that is used by the CICS COBOL programs is called COMMAREA. This scratchpad facility transfers not only data between programs of one transaction but also between programs of many transactions.

Instead using a database, there has been chosen to save all the customer information to a special OS/390 file type. Such a VSAM KSDS is predefined and delivered within OS/390 and stores information using the key-sequenced method. The customer account number has been chosen as the key under which the information is to be stored. The file that contains these customer account information has been called as the account file. From this file there has been created a browse file which arrange the customer accounts by their names. Each time a customer record in the account file is updated, a lock file is created to prevent the account file from concurrent updates. The first time, the account file has been filled by predefined customer data, also published within the CD.

Before NACT can be used on the CICS terminal a few CICS resource definitions have to be set up. These definitions have been added to the CSD and create entries for the five CICS COBOL programs, for the one map set, for the three file objects, and, of course, for the two transactions; one starts the screen application, the other the print procedure.

After the CICS application \textit{NACT} has been installed and runs properly on the CICS terminal, a new application called \textit{MQNACT} has been created that runs outside a CICS terminal. It uses message and queuing to transfer data between OS/390 and WINDOWS2000 resp. CICS and JVM. \textit{MQNACT} also accesses the customer accounts stored on the OS/390-server for a read process only. It consists of the CICS business logic originally created for the CICS application NACT and of a presentation logic component created for the JVM. This JAVA application, in turn, consists of a GUI logic that uses the SWING package and of the MQSeries communication logic that sends the request data to a specified WINDOWS2000-client queue manager and gets the response data back from it. The data is transferred to another QM residing on the MQSeries server on OS/390 called OS/390-server queue manager. For this transport procedure some QM objects, for instance, message queues and message channels, have to be created on the QMs. Each QM has got a transmission queue, a reply-to queue, a remote queue \textit{definition}, and a dead letter queue. These queue stores the data until transmission by message channels. Since each QM needs a channel sender to send data and an appropriate channel receiver, both have been created. Additionally, there has been added a server connection for the communication between the JAVA application and MQSeries to the WINDOWS2000-client queue manager. The JAVA application requests the CICS business logic program \textit{NACT02} which accesses the customer records. For transferring the data from the OS/390-server queue manager to the CICS COBOL program a specific connection, the MQSeries CICS Bridge, is needed and has been installed. For starting this bridge automatically each time CICS starts up, a script has been created, compiled, and added to the CICS start up jobs.
Within the last topic we have described how to secure the CICS region A06C001 using the ESM RACF. Each time CICS calls RACF for a security decision it compares the entries in its database to give only a secured access to the CICS region and its resources. CICS resources are subdivided into general resources and data sets. For each of both, there exist special RACF mechanisms to secure the resources. They are classified into Data Set Protection, CICS Transaction Security, and CICS Command Security. Other mechanisms as the Terminal User Security and Surrogate User Security are RACF-supplied mechanisms. Each of these mechanisms uses RACF security profiles stored into special resource classes. The Terminal User Security mechanism controls whether a user is defined to RACF resp. has an appropriate user profile stored to the RACF database. After the users, who require an access to CICS and its resources, were identified, some required user IDs for the CICS region have been created – the CRU, the DCU, and the PLTPIU. Because the CRU has to act as a surrogate user for the DCU and PLTPIU, the Surrogate User Security mechanism has been used to define appropriate surrogate user profiles. Before any user may access the CICS region an appropriate authority to the region's APPLID must be added to its general resource class.

The Data Set Protection mechanism secures the CICS region data sets using data set security profiles stored to the RACF database. As part of the CICS general resources the transactions and the SP-type commands are secured by security mechanisms only available for CICS. The CICS Transaction Security secures the transactions distinguished into three categories (CAT) – CAT1-, CAT2-, and CAT3-transactions. CAT1-transactions are called as CICS internal transactions to which only the CRU requires an access. Transactions that should not be executed by everybody are called CAT2-transactions, whereas CAT3-transactions are exempt from any security check. For each of these transactions, IBM-supplied or user-installed, there have been defined the security profiles within CLISTS. The CLISTS \texttt{CATxJEDI} (x=1,2,3) hold the member and group profiles for the CATx-transactions supplied by CICS. For defining the security profiles for the user-installed transactions there have been created own CLISTS called \texttt{MQSJEDI} and \texttt{USERJEDI}. A CICS transaction can be executed when the user has a minimum READ access, a higher access is not needed.

The CICS Command Security secures the SP-type commands that can only be executed within the special CICS-supplied transaction CEMT, CEMT-dependent transactions, or within CICS application programs that use these SP-type commands on the EXEC CICS statement. With the SP-type commands only predefined CICS resources can be accessed. It is only needed to define the required authority level to these CICS resources to get an access to them. These CICS resources and their access permission by the SP-type commands are secured within the CLIST \texttt{COM1JEDI}.

For managing the RACF database special commands are used. These RACF commands are distinguished into commands used for user profiles, user group profiles, data set profiles and general resource profiles. Some information required for RACF is stored within parameters in the CICS region SIT. Into this table some new security related entries for the region have been added, for example, the names of the DCU, and PLTPIU. The CICS RACF security mechanisms are also activated by SIT-parameters as same as own defined class names for the security profiles can be set on the associated parameters. Last but not least, the CICS login terminal has been suited...
properly to convert the characters entered on the sign on screen into upper case characters because RACF only recognises them.

Furthermore, this master thesis is introduced by two chapters explaining some facts of the Operating System /390 and of the OLTP system CICS. There have been also given some facts about the MOM system MQSeries.

### 7.2 Further Work

An interesting point for further work is to write the business logic in other programming languages as for example in JAVA, or using object oriented languages as C++, C#, or Object COBOL. This could result in a comparison of the performance of these new created business logic programs with the existing ones. Instead using the COMMAREA as a storage area for exchange data, the TWA could be used for programs executed within one transaction. Only for the transmission between transactions the COMMAREA should then be used.

As another extension, the customer accounts could be stored onto a real database managed by a database system, for example DB/2, to increase the information contents. Using this database system requires no more an additional browse file, browsing can be done within this one file. Some more customer informations could also be added to the database as same as using the bank application as a real one with adding money to an account, and withdraw money from an account.

Since the IMS/DB system is also a common transaction system used in industries worldwide, the business application could be implemented to use its transaction monitor. This implies also a comparison with the CICS transaction monitor.

It could be also interesting to implement a real credit card banking as a new feature to the application. A credit card double could be used on an ATM-like machine to demonstrate the hardware facilities.

The presentation logic could also be written in other languages, but a bit more interesting is to use other transfer mechanisms to work with the customer accounts stored on the OS/390-server. For example, the CICS Transaction Gateway, or EJBeans could be used. Afterwards, a comparison between these methods could be created.

As seen, the generic profile ** in class TCICSTRN should not be used. It is better to use for each transaction an own security profile or add it to an existing profile. Otherwise, all users have an access to all the transactions not secured by security profiles. Furthermore, transactions created by students should always begins with the letter P for a better security administration.
BIBLIOGRAPHY


Generation of a Java front end for a standalone CICS application accessed through MQSeries &
Securing CICS with RACF

http://www.yelavich.com/history/ev199203.htm - last referenced to: 31.07.2004CD:
pubs/lecpap/YEL03/ev199203.htm

http://www.yelavich.com/history/ev196801.htm - last referenced to: 31.07.2004CD:
pubs/lecpap/YEL01/ev196801.htm

IBM No. SC34-5604-04 CD: no pdf versionpubs/boooks/amtyak03.boo

[AMQ95] IBM CORPORATION (Publ.): MQSeries : An Introduction to Messaging and Queuing. 2nd

No. SC33-1687-40 CD: no pdf versionpubs/boooks/dfhjap3b.boo

No. SC33-0674-01 CD: no pdf versionpubs/boooks/dfhzp104.boo

IBM No. SC33-1688-40 CD: no pdf versionpubs/boooks/dfhjap4c.boo

2000. - IBM No. GC33-1681-35 CD: pubs/books/dfhjaa17.pdfpubs/books/dfhjaa17.boo

CD: pubs/books/csqzae05.pdfpubs/books/csqzae05.boo


to Enterprise Servers and Operating Systems. 2000. CD: http : / / w w w -
306.ibm.com/software/info/university/

SC33-1872-03 CD: pubs/books/csqaee03.pdfpubs/books/csqaee03.boo

1999. - IBM No. GC28-1725-06 CD: pubs/books/e0z1a120.pdfpubs/books/e0z1a120.boo

No. SC33-1369-13 CD: no pdf versionpubs/books/csqaaj05.boo

[MQI94] IBM CORPORATION (Publ.): MQSeries : Message Queue Interface Technical Reference. 3rd

GC28-1779-06 CD: pubs/books/iea1f110.pdfpubs/books/iea1f110.boo
<table>
<thead>
<tr>
<th>Author</th>
<th>Publisher (Publ.)</th>
<th>Title</th>
<th>Edition</th>
<th>ISBN</th>
<th>CD</th>
<th>Notes</th>
</tr>
</thead>
</table>
APPENDICES
APPENDIX A

The CICS Business Application NACT

A.1 Defining an SDS template using ISPF/PDF

An SDS template can also be created on OS/390 using the ISPF/PDF. This template has also to be set to the “fixed block” format with 80 blocks per record. One after another, all source files can be uploaded from the CD into this template. After each upload step, the transferred data are received into their designated data sets as same as described in chapter 4.2 “Uploading the files”.

After log on to TSO by entering a user ID with its associated password the CMAM appears (Figure 85). The ISPF/PDF program is started by entering the shortcut P on the OPTION line. This opens the ISPF PRIMARY OPTION MENU. From there, the option UTILITY (shortcut 3) opens the UTILITY SELECTION PANEL. The DATA SET option (shortcut 2) opens the DATA SET UTILITY (DSU). The short form P.3.2 entered on the option line of the CMAM leads directly to the DSU (Figure 85). After entering the DSU the SDS template is allocated by entering TBUSSE.CICSADP.NEWSEQ in the field Data Set Name and typing an A on the command line (Figure 86). The next screen ALLOCATE NEW DATA SET appears (Figure 87). An SDS is created, if the Directory blocks-field is set to 0. The record size of the data set is set to 2 MB as Primary quantity with an additional 1 megabyte as Secondary quantity. In the Record format-field FB is entered to fix the record length and use the block format. The Record length is fixed as 80 and the Data set name type is left blank. After confirming these adjustments by pressing the enter key the message “Data set allocated” appears in the upper right corner of the DSU screen. After that, the created template is ready to get data from the CD using FTP.

After a successful log on at the FTP client the command dir is executed. This displays a list of OS/390 data sets containing the newly created SDS-template TBUSSE.CICSADP.NEWSEQ. The column LRECL reports that the SDS has a fixed record length of 80 blocks (Figure 88). As next step, the important command bin is executed. By using this command binary source files are sent as binary files to OS/390. The files are transferred on to OS/390 using the command put, for instance (Figure 89):

```
put cicsadp.loa cicsadp.newseq
```

The file is can be received into its associated data set.
Generation of a Java front end for a standalone CICS application accessed through MQSeries & Securing CICS with RACF

CUSTOMPAC MASTER APPLICATION MENU

OPTION ===> P.3.2 SCROLL ===> PAGE

IS ISMF - Interactive Storage Management Facility
P PDF - ISPF/Program Development Facility
ATC ATC - Application Testing Collection
ART ARTT - Automated Regression Testing Tool
DB2 DB2 - Perform DATABASE 2 interactive functions
QMF QMF - QMF Query Management Facility
C CPSM - CICSPlex/SM
M MQ - MQSeries
MA MQAPPS - MQSeries Utilities
IP IPCS - Interactive Problem Control Facility
OS SUPPORT - OS/390 ISPF System Support Options
OU USER - OS/390 ISPF User Options
SM SMP/E - SMP/E Dialogs
SD SDSF - System Display and Search Facility
R RACF - Resource Access Control Facility
DI DITTO - Data Interfile Transfer, Testing and Operations
HC HCD - Hardware Configuration Definition
S SORT - DF/SORT Dialogs

F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE

Figure 85: CUSTOMPAC MASTER APPLICATION MENU on the JEDI OS/390-server

Menu RefList Utilities Help

Data Set Utility

A Allocate new data set                 C Catalog data set
R Rename entire data set                U Uncatalog data set
D Delete entire data set                S Data set information (short)
blank Data set information                  M Allocate new data set
Data Set Name . . . 'TBUSSE.CICSADP.NEWSEQ'
Volume Serial . . .  (If not cataloged, required for option "C")
Data Set Password  . .           (If password protected)

ISPF Library:
Project . . . ___
Group . . . ___
Type . . __________

Other Partitioned, Sequential or VSAM Data Set:

Option ===> a ________________________________________________________________
F1=Help      F3=Exit     F10=Actions  F12=Cancel

Figure 86: Data Set Utility screen in ISPF/PDF
Appendix A – The CICS Business Application NACT 225

Figure 87: “Allocate New Data Set” screen in ISPF/PDF

Figure 88: FTP session – Listing the own files on OS/390
Generation of a Java front end for a standalone CICS application accessed through MQSeries & Securing CICS with RACF

Figure 89: FTP-session – Copying one source file to OS/390
A.2 Listings referenced to in chapter 4

Listing 11: The physical map of the map set NACTSET
OS/390 – TBUSSE.CICSADP.COBSRCE(NACTSET)
CD – listings/chapter4/os390/cobol/tbusse/cicsadp/cobsrce/nactset

Listing 12: The symbolic description map of the map set NACTSET
OS/390 – TBUSSE.CICSADP.COBCOPY(NACTSET)
CD – listings/chapter4/os390/cobol/tbusse/cicsadp/cobcopy/nactset

Listing 13: The 3270 presentation logic of the NACT application – NACT01
OS/390 – TBUSSE.CICSADP.COBSRCE(NACT01)
CD – listings/chapter4/os390/cobol/tbusse/cicsadp/cobsrce/nact01

Listing 14: The CRUD business logic of the NACT application – NACT02
OS/390 – TBUSSE.CICSADP.COBSRCE(NACT02)
CD – listings/chapter4/os390/cobol/tbusse/cicsadp/cobsrce/nact02

Listing 15: The Browse business logic of the NACT application – NACT05
OS/390 – TBUSSE.CICSADP.COBSRCE(NACT05)
CD – listings/chapter4/os390/cobol/tbusse/cicsadp/cobsrce/nact05

Listing 16: The Error Handling business logic of the NACT application – NACT04
OS/390 – TBUSSE.CICSADP.COBSRCE(NACT04)
CD – listings/chapter4/os390/cobol/tbusse/cicsadp/cobsrce/nact04
Listing 17: The COBOL copybook NACWCRUD
OS/390 – TBUSSE.CICSADP.COBCOPY(NACWCRUD)
CD – listings/chapter4/os390/cobol/tbusse/cicsadp/cobcopy/nacwcrud

Listing 18: The COBOL copybook NACCTREC
OS/390 – TBUSSE.CICSADP.COBCOPY(NACCTREC)
CD – listings/chapter4/os390/cobol/tbusse/cicsadp/cobcopy/nacctrec

Listing 19: The COBOL copybook NACCCRUD
OS/390 – TBUSSE.CICSADP.COBCOPY(NACCCRUD)
CD – listings/chapter4/os390/cobol/tbusse/cicsadp/cobcopy/nacccrud

Listing 20: Storing the data of the NACT application – Installing the account, locking, and name files on the OS/390-server
OS/390 – TBUSSE.CICSADP.JCLLIB(VSAM)
CD – listings/chapter4/os390/cobol/tbusse/cicsadp/cobsrce/jcllib/vsam

Listing 21: The CICS resource definitions for the NACT application
OS/390 – TBUSSE.CICSADP.CSDDEFS(CICSCSD)
CD – listings/chapter4/cobol/tbusse/cicsadp/cobsrce/csddefs/cicscsd

Listing 22: The additional CICS SIT definition script C001 (besides COMMON and END)
OS/390 – CICS.COMMON.SYIN(C001)
CD – listings/chapter4/scripts/cics/common/sysin/c001
Listing 23: The CICS startup script CICSC001

OS/390 – SYS1.PROCLIB(CICSC001)

CD – listings/chapter4/scripts/sys1/proclib/cicsc001
APPENDIX B

The MQSeries CICS Business Application MQNACT

B.1 Reaching the log MSGUSR of the CICS region

Entering sd.da on the command line of the CMAM opens the DA-Panel (Figure 90). This panel displays active users in the sysplex. As next, the cursor has to be placed in the first column NP before the name of the started task procedure CICSC001 and a question mark has to be entered (Figure 91). Pressing the enter key leads to the next panel – the logs of the CICS region A06C001. Into the log MSGUSR the MQSeries CICS Bridge writes its comments; to open the log, the character S has to be again entered into the column NP before the string MSGUSR (Figure 92). The log opens at the first message, but the interesting message is anywhere in the log. To search for messages about the MQSeries CICS Bridge queue the string f bridge or find bridge can be entered (Figure 93). Immediately, the result is presented (Figure 94).
Generation of a Java front end for a standalone CICS application accessed through MQSeries & Securing CICS with RACF

Figure 91: Placing a question mark in the column NP to display the logs

Figure 92: Placing the character "S" in the column NP to display the MSGUSR log
Appendix B – The MQSeries CICS Business Application MQNACT

Figure 93: Searching for the MQSeries CICS Bridge message

Figure 94: The MQSeries CICS Bridge has been successfully started
B.2 Listings referenced to in chapter 5

Listing 24: The script IEFSSN00 for the OS/390 subsystem name table
OS/390 – SYS1.PARMLIB(IEFSSN00)
CD – listings/chapter5/os390/scripts/sys1/parmlib/iefssn00

Listing 25: The start script MQA1MSTR for the queue manager MQA1
OS/390 – SYS1.PROCLIB(MQA1MSTR)
CD – listings/chapter5/os390/scripts/sys1/proclib/mqa1mstr

Listing 26: Non-recoverable objects for the queue manager MQA1 defined within the script CSQ4INP1
OS/390 – MQM.MQA1.SCSQPROC(CSQ4INP1)
CD – listings/chapter5/os390/scripts/mqm/mqa1/scsqproc/csq4inp1

Listing 27: Must have system objects for the queue manager MQA1 defined within the Script CSQ4INSG
OS/390 – MQM.MQA1.SCSQPROC(CSQ4INSG)
CD – listings/chapter5/os390/scripts/mqm/mqa1/scsqproc/csq4insg

Listing 28: System objects for distributed queuing and clustering (not CICS) for the queue manager MQA1 defined within the script CSQ4INSX
OS/390 – MQM.MQA1.SCSQPROC(CSQ4INSX)
CD – listings/chapter5/os390/scripts/mqm/mqa1/scsqproc/csq4insx

Listing 29: The script CSQ4STRT starts the Channel Initiator and the Channel Listener
OS/390 – MQM.MQA1.SCSQPROC(CSQ4STRT)
CD – listings/chapter5/os390/scripts/mqm/mqa1/scsqproc/csq4strt
Listing 30: CICS objects for the MQSeries CICS adapter defined within the script CSQ4B100
OS/390 – CICS.COMMON.CSDDEFS(CSQ4B100)
CD – listings/chapter5/os390/scripts/cics/common/csddefs/csq4b100

Listing 31: The script DFHCSD01 to update the CSD
OS/390 – CICS.COMMON.CSDDEFS(DHCSD01)
CD – listings/chapter5/os390/cics/common/csddefs/dfhcsd01

Listing 32: The script CSQ4INYG defines additional general objects for the queue manager MQA1
OS/390 – MQM.MQA1.SCSQPROC(CSQ4INYG)
CD – listings/chapter5/os390/scripts/mqm/mqa1/scsqproc/csq4inyg

Listing 33: The updated CICS SIT definition script C001 (besides COMMON and END)
OS/390 – CICS.COMMON.SYSIN(C001)
CD – listings/chapter5/os390/scripts/cics/common/sysin/c001

Listing 34: The script CSQ4CKBM defines the MQSeries CICS Bridge queue and its trigger process
OS/390 – MQM.MQA1.SCSQPROC(CSQ4CKBM)
CD – listings/chapter5/os390/scripts/mqm/mqa1/scsqproc/csq4ckbm

Listing 35: CICS objects for the MQSeries CICS Bridge defined within the script CSQ4CKBC
OS/390 – CICS.COMMON.CSDDEFS(CSQ4CKBC)
CD – listings/chapter5/os390/scripts/cics/common/csddefs/csq4ckbc
Listing 36: The MQSeries CICS Bridge – The script STRTCKBR to start the bridge automatically during CICS start up
OS/390 – CICS.COMMON.CICSSRC(STRTCKBR)
CD – listings/chapter5/os390/scripts/cics/common/cicssrc/strtckbr

Listing 37: The MQSeries CICS Bridge – The CICS COBOL compiling script DFHYITVL
OS/390 – CICSTS13.CICS.SDFHPROC(DFHYITVL)
CD – listings/chapter5/os390/scripts/cicsts13/cics/sdfhproc/dfhyitvl

Listing 38: The MQSeries CICS Bridge – The PLT for programs loaded during CICS startup
OS/390 – CICS.COMMON.TABSRC(PLTIT)
CD – listings/chapter5/os390/scripts/cics/common/tabsrc/pltit

Listing 39: The MQSeries CICS Bridge – The DFHAUPLE script that compiles PLT scripts
OS/390 – CICS.COMMON.TABSRC(DFHAUPLE)
CD – listings/chapter5/os390/scripts/cics/common/tabsrc/dfhauple

Listing 40: The MQSC file “mqadmvs.tst” defines the objects for the WINDOWS2000-client queue manager TBUSSE.NACT
CD – listings/chapter5/windows/mqseries/mqadmvs.tst

Listing 41: The batch file loads the MQSC script “mqadmvs.txt”
CD – listings/chapter5/windows/mqseries/def.cmd
Listing 42: The MQSeries communication logic of the JAVA application – MQCommunicator.java

CD – listings/chapter5/windows/java/MQCommunicator.java

Listing 43: The presentation logic of the JAVA application – MQClient.java

CD – listings/chapter5/windows/java/MQClient.java
APPENDIX C

Securing CICS with RACF

C.1 Restarting the CICS region

If a SIT should be rebuild the CICS region has to be restarted. There is no other way to create a new SIT for a CICS region. The CICS region needs also to be restarted in a few more situations. CICS can be shut downed by using the transaction CEMT within the command `PERFORM SHUTDOWN` that is typed on the CICS terminal:

```
CEMT PERFORM SHUTDOWN
```

If an immediate shutdown is required the transaction command is:

```
CEMT PERFORM SHUTDOWN Immediate
```

Before the CICS security was active these resp. every transaction could be executed by every user logged on to CICS. Performing a CICS shutdown after activation of the CICS security is only allowed to users which have permission to execute the transaction CEMT and the command `SHUTDOWN`. This message is displayed on the CICS terminal screen during a shutdown of CICS, if no security is activate:

```
A06C001  CICS is being quiesced by user ID STCCICS in transaction CEMT at netname SC0TCP02.
```

After activating CICS security this message shows "... user ID TBUSSE ..." instead of "... user ID STCCICS ...". Hence, CICS lists the user who initiated the shutdown.

The shutdown process of CICS can also be started from SDSF’s LOG- or DApanel by purging (`P`) the CICS jobname in case of a normal shutdown or by cancelling (`C`) the jobname from the command line. Sometimes, if the CICS terminal cannot be reached, the CICS region must be shut downed from SDSF. The appendant OS/390 commands are:

```
/P CICSC001
/C CICSC001
```
The restart of the CICS region can be performed by entering the OS/390 START command (\$) together with the region start script CICSC001 on the command line in the SDSF LOG- or DApanel:

\$/S CICSC001

It is essential to have the RACF system SPECIAL or OPERATIONS attribute to execute these commands.

### C.2 Correction of a CICS system log failure after an OS/390-server IPL and a CICS restart

After the JEDI OS/390-server has been shut downed and re-IPLed, a failure with the primary CICS system log DFHLOG occurred. As the CICS system log initialisation has been started during the CICS region restart due to the IPL, the primary CICS log stream called $A06C001.DFHLOG$ stored to the VSAM KSDS $CICS.A06C001.DFHLOG.DAVPLEX$ should be accessed by the procedure IEESYSAS. This procedure is stored in the data set SYS1.PROCLIB. For the access the procedure needs an UPDATE authority but the job has not had the required authority. The data set profile $CICS.A06C001.**$ existed not yet at this time (see chapter 6.3.6, “The CICS region data set protection”, page 178), only the generic data set profile $CICS.**$.

Due to this behaviour an error message has been written to the OS/390 system log (Listing 44, next page). As listed in lines 03-06 of the Listing 44, the job IEESYSAS requires the UPDATE access to the generic data set profile $CICS.**$. This could be specified with the RACF command PERMIT or ALTDSD:

*Example 1:* PERMIT 'CICS.**' ID(IEESYSAS) ACCESS(UPDATE)

*Example 2:* ALTDSD 'CICS.**' UACC(UPDATE)

Unfortunately, giving the job the required authority after the first impossible CICS region restart after an IPL is a pointless venture. CICS has intermediately deleted the VSAM KSDS due to the access failure during the region initialisation. When the region is then restarted again, CICS tries to create a new VSAM KSDS for the primary CICS system log using IEESYSAS, which is impossible, because IEESYSAS has only an UPDATE access. It now requires an ALTER access to create the data set. This error stops the CICS region initialisation again and shut downs the CICS region. It is waited until the VSAM KSDS for the log stream is created. However, firstly it is checked, whether the log stream is still defined to the LOGR couple data set with the JCL-script LISTLOG, or not (Listing 45, next page). In the first line of Listing 45 is specified the job name LILOG01 and some job parameters. The program IXCMIAPI, the administrative data utility to display and update the LOGR couple data set\(^{18}\), is defined in line 6. This program is stored in the data set SYS1.MIGLIB and therefore, it is set on the parameter SYSLIB in the next line. The SYSIN parameter introduces the commands to display the CICS system logs $A06C001.DFHLOG$ and $A06C001.DFHPSPUNT$ (lines 9-11). DATA TYPE(LOGR) specifies that

---

\(^{18}\) For information about the LOGR couple data set and storing log streams to it refer to [MSS99].
the data which follows is for the LOGR couple data set. **REPORT(YES)** writes a complete report of the LOGR policies to the output.

---

Listing 44: Error messages explain that the primary CICS system log cannot be accessed

```plaintext
01 //LILOG01 JOB ,'TBUSSE',MSGCLASS=H,MSGLEVEL=(1,1),CLASS=A,
02 //         NOTIFY=&SYSUID,REGION=6M
03 //**************************
04 //* DISPLAY THE LOGSTREAMS *
05 //**************************
06 //STEP1 EXEC PGM=IXCMIAPU
07 //SYSLIB DD DSN=SYS1.MIGLIB,DISP=SHR
08 //SYSPRINT DD SYSOUT=* 
09 //SYSIN DD *
10   DATA TYPE(LOGR) REPORT(YES)
11     LIST LOGSTREAM NAME(A06C001.DFH*) DETAIL(YES)
12 /*
```

Listing 45: The script LISTLOG displays information about the log stream and its definition

```plaintext
01 //LILOG01 JOB ,'TBUSSE',MSGCLASS=H,MSGLEVEL=(1,1),CLASS=A,
02 //         NOTIFY=&SYSUID,REGION=6M
03 //**************************
04 //* DISPLAY THE LOGSTREAMS *
05 //**************************
06 //STEP1 EXEC PGM=IXCMIAPU
07 //SYSLIB DD DSN=SYS1.MIGLIB,DISP=SHR
08 //SYSPRINT DD SYSOUT=* 
09 //SYSIN DD *
10   DATA TYPE(LOGR) REPORT(YES)
11     LIST LOGSTREAM NAME(A06C001.DFH*) DETAIL(YES)
12 /*
```
LIST LOGSTREAM is the request to display LOGR policies specified within the NAME parameter. Detailed information about the named log stream(s) data and definition provides DETAIL(YES). After the JCL script LISTLOG has been submitted the output is written to the job's log (Listing 46).

After it has been confirmed that the log stream A06C001.DFHLOG is still existing in the LOGR couple data set, it has been deleted to initialise it new to the LOGR couple data set. For the deletion a similar JCL-script as for the log stream listing has been used. The job name has been changed to DELLOG1 and the LOGR request parameter changes to DELETE LOGSTREAM. The required name is specified on the NAME parameter (Listing 47, next page).
Listing 47: The script DELCLOG deletes the log stream entry from the LOGR policy

Listing 48: The script DEFCLOG deletes the log stream entry from the LOGR policy

When the log stream is deleted it can be new created to the LOGR couple facility. For this procedure the JCL-script DEFCLOG is used (Listing 48). The log stream is created with the command DEFINE LOGSTREAM and gets the name A06C001.DFHLOG using the parameter NAME. DASDONLY(YES) specifies that the log stream is for DASD-only and not for a coupling facility\(^\text{19}\). STG_SIZE(3000) is the size, as a number of 4 K blocks, of the staging data set for the log stream. HLQ(CICS) defines the high level qualifier which prefixes the name of the log stream data set. If the HLQ attribute is omitted “the log stream will have a high level qualifier of IXGLOGR.” ([MSS99], chapter Appendix B) The both attributes LOWOFFLOAD and HIGHOFFLOAD specifies the range in percentage where the MVS system logger starts and stops offloading data to the DASD log stream data sets. MAXBUFSIZE sets the size of the largest block that can be written to the log stream. The MVS system logger can automatically delete log data from the log stream using the attribute AUTODELETE after a retention period has been specified within RETPD. Both parameters must not be used for CICS system logs. Before the script DEFCLOG is to be executed an ALTER permission is required to create the VSAM KSDS. It is recommended to

\(^{19}\) Coupling facility or DASD-only? Please refer to chapter 1.20.2.1 in [CIG00].
specify **UACC(ALTER)** for a generic data set profile called *CICS.A06C001.* that secures the VSAM KSDSs for the primary and secondary CICS system logs *DFHLOG* and *DFHSHUNT* on the CICS region A06C001. The data set profile is created with the RACF command **ADDSDF** (Example 1) and stored in the **DATASET** resource class, which has then to be refreshed using the **SETROPTS** command (Example 1)

**Example 1:**
```
ADDSDF 'CICS.A06C001.**' UACC(UPDATE)
```

**Example 2:**
```
SETROPTS GENERIC(DATASET) REFRESH
```

After the class **DATASET** has been refreshed the log stream can be created to the LOGR policy. Afterwards, the universal access to the data set profile *CICS.A06C001.* can/should be set to **UPDATE**.

If you specify **REPORT(YES)** (or accept the default for the report keyword) with a **DATA TYPE** of **LOGR**, the requestor must have SAF read access to the **MVSADMIN.LOGR** resource to successfully obtain a report.

To define and delete log structures using *IXCMIAPU*, you need ALTER access to the LOGR resource profile named **MVSADMIN.LOGR** in the **FACILITY** general resource class. For example, use the following RACF command:

```
PERMIT MVSADMIN.LOGR CLASS(FACILITY) ACCESS(ALTER) ID(your_userid)
```

To define, delete, and update log streams (including log stream models) that are defined in coupling facility structures, you need:

- ALTER access to the appropriate log stream profile defined in the **LOGSTRM** general resource class
- UPDATE access to the coupling facility structure (**IXLSTR**) profile defined in the **FACILITY** general resource class (in this case, # profile names are prefixed with **IXLSTR**).

For example, if the log stream and structure resource profiles are defined to RACF with the following commands:

```
RDEFINE LOGSTRM log_stream_profile UACC(NONE) [NOTIFY]
RDEFINE FACILITY IXLSTR.structure_name_a UACC(NONE) [NOTIFY]
```

use the following RACF commands to give your userid the required authorizations to these two profiles:

```
PERMIT log_stream_profile CLASS(LOGSTRM) ACCESS(ALTER) ID (your_userid)
PERMIT IXLSTR.structure_name_a CLASS(FACILITY) ACCESS(UPDATE) ID (your_userid)
```

If SAF is not available or if there is no **CLASS(LOGSTRM)** or **CLASS(FACILITY)** class defined for the log stream or structure, no security checking is performed.
C.3 Listings referenced to in chapter 6

Listing 49: The default CICS region's SIT script COMMON
OS/390 – CICS.COMMON.SYSIN(COMMON)
CD – listings/chapter6/os390/scripts/cics/common/sysin/common

Listing 50: The CICS region's 2nd SIT script C001 – temporary version
CD – listings/chapter6/os390/scripts/cics/common/sysin/c001t

Listing 51: The CICS region's 2nd SIT script C001 – final version
OS/390 – CICS.COMMON.SYSIN(C001)
CD – listings/chapter6/os390/scripts/cics/common/sysin/c001

Listing 52: The CLIST CAT1JEDI secures Category-1 transactions
OS/390 – CICS.COMMON.RACF(CAT1JEDI)
CD – listings/chapter6/os390/scripts/cics/common/racf/cat1jedi

Listing 53: The CLIST CAT2JEDI secures Category-2 transactions
OS/390 – CICS.COMMON.RACF(CAT2JEDI)
CD – listings/chapter6/os390/scripts/cics/common/racf/cat2jedi
Listing 54: The CLIST CAT3JEDI secures Category-3 transactions
OS/390 – CICS.COMMON.RACF(CAT3JEDI)
CD – listings/chapter6/os390/scripts/cics/common/racf/cat3jedi

Listing 55: The CLIST MQSJEDI secures MQSeries CICS transactions
OS/390 – CICS.COMMON.RACF(MQSJEDI)
CD – listings/chapter6/os390/scripts/cics/common/racf/mqsjedi

Listing 56: The CLIST USERJEDI secures CICS User-transactions
OS/390 – CICS.COMMON.RACF(USERJEDI)
CD – listings/chapter6/os390/scripts/cics/common/racf/userjedi

Listing 57: The CLIST COM1JEDI secures CICS Resources subject to SP-type commands
OS/390 – CICS.COMMON.RACF(COM1JEDI)
CD – listings/chapter6/os390/scripts/cics/common/racf/com1jedi
Statement:

I declare that this master thesis was composed by myself and that the work contained herein is my own except where explicitly stated otherwise in the text. This work has not been submitted for any other degree or professional qualification except as specified.

Erklärung:

Ich, Tobias Busse, versichere, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe. Diese Arbeit wurde nicht für den Erhalt eines anderen wissenschaftlichen Grad veröffentlicht außer dem angegebenen.

Place/Ort Date/Datum Signature/Unterschrift

(Tobias Busse)