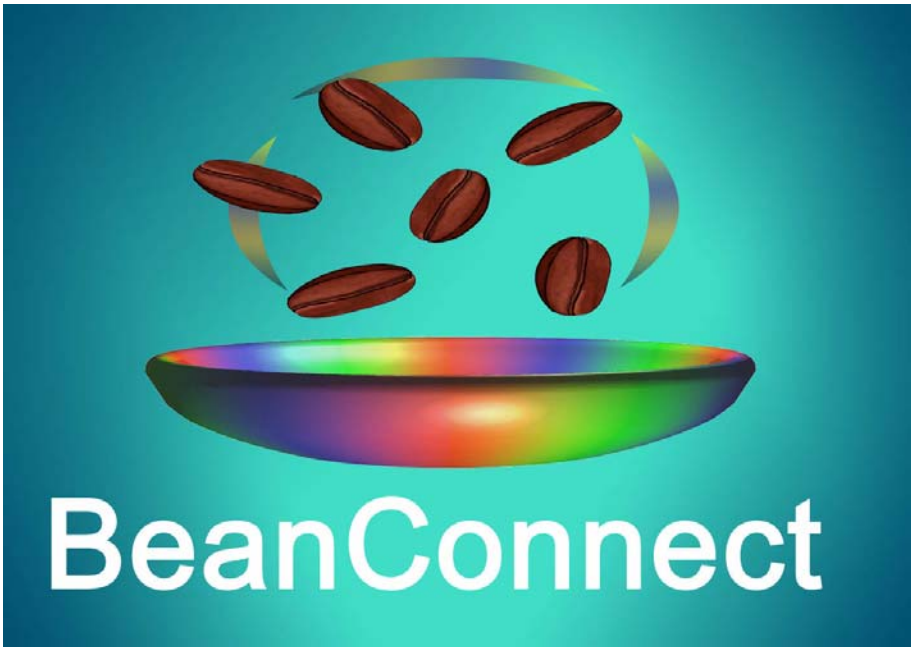
BeanConnect^{TM} V3.0



Edition August 2013

Comments… Suggestions… Corrections…

The User Documentation Department would like to know your opinion on this manual. Your feedback helps us to optimize our documentation to suit your individual needs.

Feel free to send us your comments by e-mail to:
manuals@ts.fujitsu.com

Certified documentation according to DIN EN ISO 9001:2008

To ensure a consistently high quality standard and user-friendliness, this documentation was created to meet the regulations of a quality management system which complies with the requirements of the standard DIN EN ISO 9001:2008.

cognitas. Gesellschaft für Technik-Dokumentation mbH
www.cognitas.de

Copyright and Trademarks

Contents

1 Preface . . . 17

1.1 Target group . . . 18

1.2 Structure of the BeanConnect documentation . . . 18

1.3 Structure of this manual . . . 19

1.4 Changes compared to the predecessor version . . . 20

1.5 Notes on third-party products and literature . . . 21

1.6 Notational conventions . . . 22

2 JCA adapter integration overview . . . 23

2.1 Die JCA adapter versions . . . 24
2.1.1 JCA adapter integration . . . 24
2.1.2 JCA 1.6 contracts . . . 25
2.1.3 SOA architecture . . . 26
2.1.4 JCA adapter integration in Oracle WebLogic Server . . . 27

2.2 BeanConnect architecture . . . 28
2.2.1 BeanConnect components . . . 28
2.2.1.1 BeanConnect resource adapter . . . 30
2.2.1.2 BeanConnect proxy . . . 30
2.2.1.3 BeanConnect Management Console . . . 32
2.2.1.4 BeanConnect tools . . . 33
2.2.2 Standard operation with one resource adapter and one proxy . . . 34
2.2.3 Multiple resource adapter mode . . . 35

2.3 BeanConnect as a JCA-compliant resource adapter . . . 36
2.3.1 Outbound and inbound communication . . . 36
2.3.1.1 Outbound communication . . . 37
2.3.1.2 Inbound communication . . . 37
2.3.2 Dialog and asynchronous communication . . . 39
2.3.2.1 Dialog communication . . . 39

2.3.2.2 Asynchronous communication 39
2.3.3 Transactional and non-transactional communication 39
2.3.3.1 Transactional communication 39
2.3.3.2 Non-transactional communication 40
2.3.4 Interfaces 40

2.4 BeanConnect in cluster operation 42

3 Installing BeanConnect 43

3.1 Installing BeanConnect under Solaris systems 44
3.1.1 Master installation 44
3.1.1.1 Installing PCMX 45
3.1.1.2 Installing openUTM 45
3.1.1.3 Installing the BeanConnect product files 46
3.1.1.4 Installing the openUTM-LU62 Gateway (for CICS partners) 46
3.1.1.5 Silent installation 46
3.1.2 Installing the BeanConnect proxy container and the Management Console 47

3.2 Installing BeanConnect under Linux systems 52
3.2.1 Master installation 52
3.2.1.1 Installing PCMX 53
3.2.1.2 Installing openUTM 54
3.2.1.3 Installing the BeanConnect product files 54
3.2.1.4 Installing the openUTM-LU62 Gateway (for CICS partners) 54
3.2.1.5 Silent installation 54
3.2.2 Installing the BeanConnect proxy container and the Management Console 55

3.3 Installing BeanConnect under Windows systems 60
3.3.1 Master installation 60
3.3.1.1 Installing PCMX 60
3.3.1.2 Installing openUTM 61
3.3.1.3 Installing BeanConnect 62
3.3.1.4 Installing the openUTM-LU62 Gateway (for CICS partners) 66
3.3.2 Installing the BeanConnect proxy container via the command line 67

3.4 Installing a BeanConnect resource adapter 69

3.5 Installing the BeanConnect tools 71

3.6 Update installation for the BeanConnect proxy container and Management Console 72
3.6.1 Update installation under Solaris systems 72
3.6.2 Update installation under Linux systems 74
3.6.3 Update installation under Windows systems 76

3.7 Uninstalling BeanConnect . . . 78
3.7.1 Uninstalling BeanConnect under Solaris systems . . . 78
3.7.2 Uninstalling BeanConnect under Linux . . . 79
3.7.3 Uninstalling BeanConnect under Windows . . . 80

3.8 Uninstalling the BeanConnect resource adapter . . . 82

3.9 Uninstalling the BeanConnect tools . . . 82

4 Configuration in the application server . . . 83

4.1 Overview . . . 84
4.1.1 Configuration files in the application server . . . 84
4.1.2 Configuration steps for outbound and inbound communication . . . 85

4.2 Configuring general properties for the resource adapter . . . 87
4.2.1 Defining general properties in ra.xml . . . 87
4.2.2 Defining the general properties of the resource adapter in weblogic-ra.xml . . . 93
4.2.3 Deploying and undeploying the resource adapter . . . 94
4.2.3.1 Deploying the resource adapter . . . 94
4.2.3.2 Update deployment of the resource adapter . . . 95
4.2.3.3 Undeploying the resource adapter . . . 96
4.2.4 Example of an ra.xml File . . . 96
4.2.5 Example of an weblogic-ra.xml file . . . 97

4.3 Setting configuration properties for outbound communication via OSI-TP / LU6.2 . . . 100
4.3.1 Defining general and connection-specific properties for OSI-TP / LU6.2 in weblogic-ra.xml . . . 101
4.3.1.1 Defining the resource for OSI-TP / LU6.2 . . . 101
4.3.1.2 Defining the JNDI name for OSI TP / LU6.2 . . . 102
4.3.1.3 Defining configuration properties for OSI-TP / LU6.2 . . . 103
4.3.1.4 Adapting connection pooling for OSI-TP / LU6.2 . . . 109
4.3.1.5 Defining security settings (managing sign-on) . . . 109
4.3.1.6 Example: weblogic-ra.xml . . . 111
4.3.2 Deploying an Enterprise JavaBean for OSI-TP / LU6.2 . . . 115

4.4 Configuring outbound communication via UPIC . . . 118
4.4.1 Defining general connection-specific properties for UPIC in weblogic-ra.xml . . . 119
4.4.1.1 Defining a resource for UPIC . . . 119
4.4.1.2 Defining the JNDI name for UPIC . . . 120
4.4.1.3 Setting the configuration properties for UPIC . . . 120
4.4.1.4 Adapting connection pooling for UPIC . . . 125
4.4.1.5 Defining transaction support for UPIC . . . 126
4.4.1.6 Example: weblogic-ra.xml (UPIC) . . . 126

4.4.2 Deploying an Enterprise JavaBean for UPIC 129

4.5 Setting configuration properties for inbound communication 132
4.5.1 Configuration properties in the ejb-jar.xml 132
4.5.2 Defining configuration properties for inbound communication in weblogic-ejb-jar.xml . 136
4.5.3 Examples for ejb-jar.xml and weblogic-ejb-jar.xml 136

4.6 Preparing resource adapter logging 141

4.7 Special characteristics of multiple resource adapter mode 142

4.8 Special characteristics in cluster operation 144

5 BeanConnect Management Console 149

5.1 Starting and shutting down the Management Console 150
5.1.1 Starting the Management Console 150
5.1.2 Starting the Management Console's online Help system 150
5.1.3 Shutting down the Management Console 151

5.2 User interface - Management Console window 152
5.2.1 Navigation area in the Management Console 153
5.2.2 Managed objects 154
5.2.3 Additional functions and information 156

5.3 Functions of the BeanConnect Management Console 157
5.3.1 Configuration functions 157
5.3.2 Configuration wizards 158
5.3.3 Starting and stopping proxies 160
5.3.4 Checking the availability of BeanConnect components and EIS partners 161
5.3.5 Diagnosis support 162
5.3.6 Todo topics 162
5.3.7 Cluster support 163
5.3.8 Management Console as a JMX client 164

5.4 Administrative data of the Management Console 166

6 Configuration of BeanConnect 167

6.1 Configuration steps 168

6.2 Adding a BeanConnect proxy to the Management Console 170
6.2.1 Adding a new proxy 170
6.2.2 Removing a proxy 172

6.3 Configuring the BeanConnect proxy 173
6.3.1 General information on the proxy 173
6.3.2 Proxy Components: CICS partners 177
6.3.3 Modifying the administration password 182
6.3.4 Configuration options in expert mode 183
6.3.4.1 Timer Settings 183
6.3.4.2 Performance Settings 183
6.3.4.3 Application Program Interface Mode (API Mode) 184
6.3.4.4 Container Application Process Title 184

6.4 Configuring a BeanConnect proxy cluster 185
6.4.1 Generating a proxy cluster 185
6.4.2 Adding a proxy to the proxy cluster 187
6.4.3 Removing a proxy from a cluster / removing a proxy cluster 188

6.5 Configuring the BeanConnect resource adapter 189
6.5.1 Adding a resource adapter (no cluster operation) 189
6.5.2 Adding a resource adapter in cluster operation 193
6.5.3 Resource adapter configuration file 196

6.6 Configuring the EIS partners 198
6.6.1 Configuring EIS partners of type openUTM 199
6.6.1.1 Adding EIS partners of the type UTM 199
6.6.1.2 Configuration files for EIS partners of type openUTM 207
6.6.2 Configuring EIS partners of type CICS 208
6.6.2.1 Adding EIS partners of the type CICS 208
6.6.2.2 Configuration files for the EIS partners of the type CICS 216
6.6.3 Configuring EIS partners of type XATMI 218
6.6.4 Removing an EIS partner 218

6.7 Configuring outbound communication 219
6.7.1 Configuring outbound services 219
6.7.2 Configuring outbound communication endpoints 222

6.8 Configuring inbound communication 224
6.8.1 Configuring inbound message endpoints 224
6.8.2 Configuring inbound services 229
6.8.3 Setting up users for access to inbound message endpoints 231
6.8.4 Configuring the error message prefix for inbound communication 232

6.9 Saving and activating the configuration of the BeanConnect proxy 233

6.10 Configuring the Management Console command handler (MC-CmdHandler) . . 235
6.10.1 Security and privileges 235
6.10.2 Administering the MC-CmdHandler 236
6.10.2.1 Starting the MC-CmdHandler 236
6.10.2.2 Shutting down the MC-CmdHandler 237

6.10.2.3 Configuring an MC-CmdHandler as a service 238

6.11 Configuring the Management Console as a JMX client 239
6.11.1 Defined resource adapter MBeans 239
6.11.2 Setting up the JMX client in the Management Console 242
6.11.2.1 Setting up a JMX client 243
6.11.2.2 Establishing and clearing a connection to the JMX server 245
6.11.2.3 Removing a JMX client 245

7 Adapting the configuration in the EIS partner 247

7.1 Adapting the configuration in EIS partners of type openUTM 248
7.1.1 Defining connections between BeanConnect and openUTM 248
7.1.1.1 Defining an OSI-TP connection between BeanConnect and openUTM 248
7.1.1.2 Defining a UPIC connection for outbound communication between the openUTM partner and BeanConnect 249
7.1.1.3 Defining a socket connection between the openUTM partner and BeanConnect . 250
7.1.1.4 Defining a BCMAP entry (only for BS2000/OSD partners) 250
7.1.2 Defining connections between BeanConnect and other EIS partners 251

7.2 Adapting the configuration in EIS partners of type CICS 252
7.2.1 Configuration in the CICS 252
7.2.2 Configuration of VTAM on an IBM mainframe 253

8 Administering BeanConnect 255

8.1 Administering a BeanConnect proxy via the Management Console 256
8.1.1 Starting a proxy 257
8.1.2 Restarting a proxy 259
8.1.3 Stopping a proxy 259
8.1.4 Special characteristics in cluster operation 260

8.2 Administering a BeanConnect proxy container on command level 262
8.2.1 Starting a proxy container 262
8.2.1.1 Starting via a script 262
8.2.1.2 Starting using the proxy container program group under Windows 262
8.2.1.3 Starting as a Windows service 263
8.2.1.4 Starting after abnormal termination of a proxy container run 263
8.2.2 Restarting a proxy container 264
8.2.2.1 Restarting using a script 264
8.2.2.2 Restarting using the proxy container program group under Windows 264
8.2.3 Stopping a proxy container 265
8.2.3.1 Stopping using a local script 265

8.2.3.2 Stopping using the proxy container program group under Windows 265
8.2.3.3 Stopping as a Windows service 265

8.3 Starting an MC-CmdHandler as a service on Windows systems 266

8.4 Administering the openUTM-LU62 Gateway 267
8.4.1 Starting the openUTM-LU62 Gateway 267
8.4.2 Stopping the openUTM-LU62 Gateway 268
8.4.3 Displaying status information on the openUTM-LU62 268

8.5 Administering the communication service 269
8.5.1 Starting and stopping the SNA daemon (Linux and Solaris systems) 269
8.5.2 Starting and stopping a communication service in a command line (Linux and Solaris systems) 269

8.6 Checking the availability of BeanConnect proxies 271
8.6.1 Checking the availability of a proxy 271
8.6.2 Checking the availability of a BeanConnect resource adapter 273
8.6.3 Checking the availability of an openUTM-LU62 Gateway and a communication service 275
8.6.4 Checking the availability of an MC-CmdHandler 276
8.6.4.1 Checking the availability of the MC-CmdHandler with the Management Console . 276
8.6.4.2 Checking the availability of the MC-CmdHandler in the command line 276
8.6.5 Checking the availability of an EIS partner 277

8.7 Monitoring the resource adapter with the Management Console 279
8.7.1 Establishing a connection to the MBean server 280
8.7.2 Displaying MBean object names 281
8.7.3 Displaying and modifying MBean attributes 282
8.7.3.1 Displaying MBean attributes 282
8.7.3.2 Modifying MBean attribute values 283
8.7.4 Collecting and displaying diagnostic values 284
8.7.4.1 Configuring, displaying and modifying statistics collectors 284
8.7.4.2 Displaying statistical values 285
8.7.5 Subscribing to and displaying MBean notifications 286
8.7.5.1 Subscribing to MBean notifications 287
8.7.5.2 Displaying MBean notifications 288
8.7.6 Displaying and executing MBean operations 289

9 Command Line Interface of the BeanConnect Management Console (MC-CLI) . . 291

9.1 Overview of MC-CLI 292

9.2 Creating and calling MC-CLI user scripts 295
9.2.1 Prerequisites when calling an MC-CLI user script 295
9.2.2 Preparing the configuration 296

9.2.3 Structure of the user script . . . 296
9.2.4 Specifying call parameters . . . 297

9.3 Java classes . . . 299
9.3.1 Class: BcDef . . . 299
9.3.2 Class: BcObjectType . . . 299
9.3.3 Class: BcObject . . . 300
9.3.3.1 getName() . . . 300
9.3.3.2 getObjectType() . . . 300
9.3.4 Exceptions . . . 301
9.3.4.1 Class: BcObjectException . . . 301
9.3.4.2 Class: BcParameterException . . . 302
9.3.4.3 Class: BcToolException . . . 303

9.4 Functions . . . 304
9.4.1 General . . . 304
9.4.1.1 Parameters . . . 304
9.4.1.2 Properties . . . 305
9.4.1.3 Messages . . . 306
9.4.1.4 Returns . . . 306
9.4.2 BcAdminAction . . . 306
9.4.2.1 getCheckResults() – Show results of check actions . . . 307
9.4.2.2 getResults() – Show results of all subactions of an action . . . 308
9.4.2.3 isFinishedSuccessfully() – Show success/failure of an action . . . 310
9.4.3 BcAdminEisPartner . . . 311
9.4.3.1 create() – Add EIS partner to the configuration . . . 311
9.4.3.2 getObject() – Read EIS partner object from the configuration . . . 312
9.4.3.3 getProperties() – Read properties of an EIS partner . . . 313
9.4.3.4 modifyProperties() – Modify properties of an EIS partner . . . 313
9.4.3.5 perform() – Start administrative actions . . . 314
9.4.3.6 remove() – Remove EIS partner . . . 315
9.4.3.7 Properties of an EIS partner . . . 315
9.4.4 BcAdminInboundMsgEndpoint . . . 318
9.4.4.1 create() – Add inbound message endpoint to the configuration . . . 318
9.4.4.2 getObject() – Read inbound message endpoint object from the configuration . . . 319
9.4.4.3 getProperties() – Read properties of an inbound message endpoint . . . 320
9.4.4.4 getProperties() – Modify properties of an inbound message endpoint . . . 320
9.4.4.5 remove() – Remove inbound message endpoint . . . 321
9.4.4.6 Properties of an inbound message endpoint . . . 322
9.4.5 BcAdminInboundService . . . 323
9.4.5.1 getObject() – Read inbound service object from the configuration . . . 323
9.4.5.2 getProperties() – Read properties of an inbound service . . . 324
9.4.5.3 modifyProperties() – Modify properties of an inbound service . . . 324
9.4.5.4 Properties of an inbound service . . . 325
9.4.6 BcAdminInboundUser . . . 326

9.4.6.1 create() – Add inbound user to the configuration . . . 326
9.4.6.2 getObject() – Read inbound user object from the configuration . . . 327
9.4.6.3 getProperties() – Read properties of an inbound user . . . 328
9.4.6.4 modifyProperties() – Modify properties of an inbound user . . . 328
9.4.6.5 remove() – Remove inbound user . . . 329
9.4.6.6 Properties of an inbound user . . . 330
9.4.7 BcAdminMain . . . 331
9.4.7.1 close() – Close Management Console session . . . 331
9.4.7.2 getList() – Output list of all configured objects of an object type . . . 332
9.4.7.3 getVersion() – Read Management Console version . . . 333
9.4.7.4 init() – Start Management Console session for MC-CLI . . . 333
9.4.8 BcAdminOutboundCommEndpoint . . . 334
9.4.8.1 create() – Add outbound communication endpoint to the configuration . . . 334
9.4.8.2 getObject() – Read outbound communication endpoint object from the configuration . . . 335
9.4.8.3 getProperties() – Read properties of an outbound configuration endpoint . . . 336
9.4.8.4 modifyProperties() – Modify properties of an outbound communication endpoint . . . 336
9.4.8.5 remove() – Remove outbound communication endpoint . . . 337
9.4.8.6 Properties of an outbound communication endpoint . . . 338
9.4.9 BcAdminOutboundService . . . 339
9.4.9.1 create() – Add outbound service to the configuration . . . 339
9.4.9.2 getObject() – Read outbound service object from the configuration . . . 340
9.4.9.3 getProperties() – Read properties of an outbound service . . . 341
9.4.9.4 modifyProperties() – Modify properties of an outbound service . . . 341
9.4.9.5 remove() – Remove outbound service . . . 342
9.4.9.6 Properties of an outbound service . . . 343
9.4.10 BcAdminProxy . . . 344
9.4.10.1 authenticate() – Authenticate for proxy . . . 344
9.4.10.2 getList() – List all objects of an object type present in a proxy . . . 345
9.4.10.3 getObject() – Read proxy object from the configuration . . . 346
9.4.10.4 getProperties() – Read properties of a proxy . . . 347
9.4.10.5 modifyProperties() – Modify properties of a proxy . . . 347
9.4.10.6 perform() – Start administrative actions for a proxy . . . 348
9.4.10.7 remove() – Remove proxy object from the configuration . . . 349
9.4.10.8 Properties of a proxy . . . 350
9.4.11 BcAdminProxyCluster . . . 353
9.4.11.1 addProxy() – Add proxy to the proxy cluster . . . 353
9.4.11.2 authenticate() – Authenticate for proxy cluster . . . 354
9.4.11.3 create() – Add proxy cluster to the configuration . . . 355
9.4.11.4 getList() – List all objects of a type in the proxy cluster . . . 356
9.4.11.5 getMasterProxy() – Read master proxy of a proxy cluster . . . 357
9.4.11.6 getObject() – Read proxy cluster object from the configuration . . . 357
9.4.11.7 getProperties() – Read properties of a proxy cluster . . . 358

9.4.11.8 modifyProperties() – Modify properties of a proxy cluster 358
9.4.11.9 perform() – Start administrative actions 359
9.4.11.10 remove() – Remove proxy cluster 360
9.4.11.11 removeProxy() – Remove proxy from proxy cluster 360
9.4.11.12 setMasterProxy() – Change master proxy of a proxy cluster 361
9.4.11.13 Properties of a proxy cluster 362
9.4.12 BcAdminRA 364
9.4.12.1 create() – Add resource adapter to the configuration 364
9.4.12.2 getObject() – Read resource adapter object from the configuration 365
9.4.12.3 getProperties() – Read properties of a resource adapter 366
9.4.12.4 modifyProperties() – Modify properties of a resource adapter 366
9.4.12.5 perform() – Start administrative actions 367
9.4.12.6 remove() – Remove resource adapter 368
9.4.12.7 Properties of a resource adapter 369
9.4.13 BcAdminTodo 370
9.4.13.1 getProperties() – Read properties of a todo topic 370
9.4.13.2 remove() – Delete todo topic 371
9.4.13.3 Properties of a todo topic 372

9.5 Application scenarios (examples) 373
9.5.1 Configuring outbound communication with an openUTM application 373
9.5.2 Configuring inbound communication with an openUTM application 375
9.5.3 Administer proxies 376
9.5.4 Jython sample scripts 377

10 Interfaces and programming 379

10.1 BeanConnect-specific interfaces and Common Client Interface (CCI) 380

10.2 Programming outbound communication 382
10.2.1 BeanConnect-specific interfaces for outbound communication 382
10.2.1.1 Connection factory interfaces 382
10.2.1.2 Connection interfaces (overview) 383
10.2.1.3 Communication using the connection interfaces 386
10.2.2 Common Client Interface (CCI) for outbound communication 394
10.2.3 Programming information on outbound communication 395
10.2.3.1 Addressing an EIS application 395
10.2.3.2 Placing BeanConnect calls in an EJB 395
10.2.3.3 Authentication (user ID and password) 396
10.2.3.4 Querying information on the conversation with the EIS application 397
10.2.3.5 Programming hints with respect to CICS applications 397
10.2.3.6 Support of DPL (Distributed Program Link) programs 398
10.2.4 Program framework for outbound communication 399
10.2.4.1 Program framework for BeanConnect-specific interfaces 399

10.2.4.2 Program framework for Common Client Interface (CCI) 400
10.2.5 Outbound communication with XATMI partners 405
10.2.6 Code samples for outbound communication 406

10.3 Programming inbound communication 410
10.3.1 OLTP message-driven beans 410
10.3.2 Inbound communication with openUTM partners 411
10.3.3 Inbound communication with CICS applications 412
10.3.4 Inbound communication with other EIS partners (openUTM) 414
10.3.5 Inbound communication with XATMI partners 414
10.3.6 BeanConnect-specific interfaces for inbound communication 415
10.3.6.1 Programming information on OLTP message-driven beans 415
10.3.6.2 Determining sender contexts in the OLTP message-driven bean 417
10.3.6.3 Program framework using the interfaces AsyncOltpMessageListener and OltpMessageListener 419
10.3.7 Common Client Interface (CCI) for inbound communication 421
10.3.7.1 Programming information on OLTP message-driven beans (CCI) 421
10.3.7.2 Program framework using the interface javax.resource.cci.MessageListener 421
10.3.8 Code samples for inbound communication 423

11 Encoding and national language support 429

11.1 Encoding 430
11.1.1 Standard conversion between EBCDIC code and Unicode for EIS partners of type openUTM 430
11.1.2 Standard conversion between EBCDIC code and Unicode for EIS partners of type CICS 433
11.1.3 Using other predefined code tables 433
11.1.4 Using custom charsets 441
11.1.4.1 Custom Charset Provider 441
11.1.4.2 Creating and using legacy code tables 441

11.2 National language support for message output 443

12 High availability and scalability 447

12.1 Shared memory of the proxy container 448
12.1.1 Adapting the shared memory 448

12.2 Number of proxy container processes 450
12.2.1 Displaying the workload of processes 450
12.2.2 Setting the number of processes 451

12.3 Page pool area and cache of the proxy container . . . 453

12.4 Number of parallel connections to the EIS partner . . . 454

12.5 Asynchronous processing in the proxy container . . . 455
12.5.1 Duration of asynchronous requests . . . 455
12.5.2 Inbound communication . . . 456

12.6 OSI-SCRATCH-AREA in the proxy container . . . 457

12.7 Number of semaphores in the proxy container . . . 458

13 Logging, diagnostics and troubleshooting . . . 459

13.1 Logging with Log4j . . . 460
13.1.1 Basic principles of Log4j . . . 460
13.1.1.1 Loggers . . . 460
13.1.1.2 Appenders . . . 462
13.1.1.3 How the rolling file appender works . . . 463

13.2 Logging with JDK logging . . . 465

13.3 Configuring logging with Log4j . . . 466
13.3.1 Configuring logging for BeanConnect resource adapter and proxy . . . 466
13.3.1.1 Configuring loggers . . . 467
13.3.1.2 Configuring appenders . . . 468
13.3.2 Editing the Log4j configuration file using the BeanConnect Management Console . . . 469
13.3.3 Configuring the BeanConnect Management Console as a Log4j socket reader . . . 470
13.3.4 Displaying the logging events in the BeanConnect Management Console . . . 472
13.3.5 Display the Log4j logging file using the BeanConnect Management Console . . . 475

13.4 LogWriter for connection factories . . . 476

13.5 Diagnosis of the BeanConnect resource adapter . . . 481
13.5.1 Overview of logging in the BeanConnect resource adapter . . . 481
13.5.2 Predefined logging configuration of a resource adapter . . . 483
13.5.3 Logging of user interface calls . . . 489

13.6 Diagnosis of the BeanConnect proxy container . . . 490
13.6.1 Predefined logging configuration of a proxy . . . 490
13.6.2 Log files of the BeanConnect proxy container . . . 493
13.6.2.1 stdout/stderr log . . . 493
13.6.2.2 System log file SYSLOG . . . 494
13.6.2.3 Dumps and diagnostic dumps . . . 495
13.6.2.4 Application log under Windows . . . 495
13.6.3 Traces of the BeanConnect proxy container . . . 496
13.6.3.1 OSS trace . . . 496

13.6.3.2 BCAM trace 497
13.6.3.3 CMX trace 498

13.7 Diagnosis of the BeanConnect Management Console 500

13.8 Diagnosing the BeanConnect tools 501

13.9 Diagnosis of the openUTM-LU62 Gateway 502
13.9.1 Traces and logs of the openUTM-LU62 Gateway 502
13.9.1.1 Activate/deactivate traces 502
13.9.1.2 Evaluating traces and logs 503
13.9.2 Diagnosis information for the openUTM-LU62 Gateway 505

13.10 Diagnosis of SNAP-IX for Solaris systems 507
13.10.1 Diagnosis with the Management Console 508

13.11 Diagnosis of the IBM Communications Server for Linux 509
13.11.1 Diagnosis with the Management Console 510

13.12 Diagnosis of the IBM Communications Server for Windows systems 511
13.12.1 Diagnosis with the Management Console 511

13.13 Collecting diagnostic information 512

13.14 Error messages of the BeanConnect proxy container 513
13.14.1 Configuration error messages 513
13.14.2 Runtime error messages 514
13.14.2.1 Types of messages 514
13.14.2.2 K messages 516
13.14.2.3 P messages 536
13.14.2.4 U messages 543

13.15 Error messages of the openUTM-LU62 Gateway 548
13.15.1 openUTM-LU62 Gateway error messages on start-up 548
13.15.2 openUTM-LU62 Gateway error messages at runtime 549
13.15.3 openUTM-LU62 Gateway error messages on status queries 558
13.15.4 openUTM-LU62 Gateway error messages during administration 559
13.15.5 openUTM-LU62 Gateway error messages during configuration 560

13.16 Error codes 561
13.16.1 Error codes during file processing (DMS error codes) 561
13.16.2 System error codes 562

14 Cobol2Java 563

14.1 Mapping COBOL data types to Java classes 564
14.1.1 System requirements 565
14.1.2 Installation 565

14.2 Converting COBOL data types . . . 567
14.2.1 Creating an XML description for a COBOL program in BS2000 . . . 567
14.2.1.1 Transferring the LMS library to BS2000/OSD . . . 567
14.2.1.2 Converting the data structures . . . 568
14.2.1.3 D.XMLPROG . . . 568
14.2.1.4 D.XMLCOPY . . . 569
14.2.1.5 Example call . . . 570
14.2.1.6 Generated files . . . 570
14.2.2 Generating Java classes on UNIX or Windows systems . . . 571
14.2.2.1 Generating Java classes with Ant . . . 571
14.2.2.2 Generating Java classes without Ant . . . 575

14.3 Programming reference . . . 577
14.3.1 Type assignment . . . 577
14.3.2 Naming conventions . . . 579
14.3.3 Accessing COBOL fields . . . 580
14.3.3.1 Writing a data field . . . 580
14.3.3.2 Reading a data field . . . 581
14.3.3.3 Replacement data type PicU . . . 581
14.3.3.4 Setting and reading the data for the entire structure (for sending and receiving) . . 582
14.3.4 Java/EBCDIC conversion . . . 583
14.3.5 Formatted mode support . . . 583

14.4 Example . . . 584
14.4.1 COBOL example program . . . 584
14.4.2 Creating the XML description . . . 585
14.4.3 Generating the Java classes . . . 586
14.4.4 Use of the generated classes . . . 587

14.5 Error messages and error handling . . . 590

Glossary . . . 591

Index . . . 609

List of Examples . . . 623

List of Figures . . . 625

1 Preface

The BeanConnect^{TM} adapter is part of the openSEAS product range (open Suite for Enterprise Application Server). BeanConnect implements the link between classical transaction monitors and modern application servers, thus allowing legacy applications to be efficiently integrated into modern Java applications.

This manual describes the product BeanConnect. BeanConnect provides a JCA 1.6 compliant adapter connecting openUTM and CICS^{TM} applications to applications based on Java^{TM} EE (Java Platform, Enterprise Edition), e.g. the Oracle^{TM} WebLogic Server. In this document, the EIS (Enterprise Information System) is understood to be the openUTM or CICS application. The term openUTM application (UTM application for short) is used to refer to both standalone UTM applications and UTM cluster applications.

BeanConnect supports different communication directions and models. BeanConnect allows outbound and inbound communication, transactional and non-transactional communication as well as dialog and asynchronous communication.

BeanConnect components

BeanConnect consists of the following components:

- The **BeanConnect resource adapter** implements the JCA 1.6 interfaces. Being a compliant JCA adapter, it is deployed in the application server and hence runs within the application server.
- The **BeanConnect proxy** provides the functionality of a protocol converter as well as functions for transaction control and transaction processing. It can be seen as an intelligent gateway. It communicates with the resource adapter running within the application server on the one hand and with the EIS on the other hand.
- The **BeanConnect Management Console** (MC) is a Java GUI used for configuration and administration of BeanConnect proxies. It can manage multiple proxies running on the same system or on other systems.
- The **Management Console Command Line Interface** (abbreviated to MC-CLI in the following) provides a set of Jython functions which allow you to start BeanConnect Management Console functions from a Jython script.
- The **BeanConnect tools** are tools which you require in many BeanConnect applications. They include Cobol2Java and the MC-CmdHandler.

1.1 Target group

This manual is intended for the following target groups:

- BeanConnect administrators
- Administrators of an application server, such as Oracle WebLogic Server
- Deployers
- Developers of Enterprise Java Beans (EJB)
- openUTM and CICS administrators

It is assumed that you are familiar with Java and with the JCA V1.6 specification.

1.2 Structure of the BeanConnect documentation

The documentation for BeanConnect comprises the following components:

- The **BeanConnect** manual (this document).
- The **Help System** for the Management Console, which provides quick and context-sensitive support on screen when configuring and administering BeanConnect proxies.
- The **JavaDoc** of BeanConnect, which is supplied with the resource adapter JAR file `BC30A00_RA.jar` and is available after the installation of the resource adapter.
- The **JavaDoc** of the BeanConnect Management Console Command Line Interface. This documentation is available following installation of the BeanConnect Management Console.

Manual extension file

Last-minute changes and additions to this manual can be found in the product-specific manual extension file (man file).

The man file is stored in the BeanConnect home directory

- under Solaris and Linux® in `/docs/English/man01-en.pdf`
- under Windows™ in `\docs\English\man01-en.pdf`

i For detailed information on the Oracle WebLogic Server and other software products mentioned in this manual, please refer to the relevant documents.

1.3 Structure of this manual

Chapter 2, "JCA adapter integration overview" provides an overview of the Oracle concept for the integration of adapters. It describes the features of BeanConnect and shows how this product is embedded in the Oracle WebLogic Server environment.

Chapter 3, "Installing BeanConnect" describes how to install, update and uninstall BeanConnect.

Chapter 4, "Configuration in the application server" provides information for configuring outbound and inbound communication via the OSI-TP / LU6.2 protocol and outbound communication via the UPIC protocol. It describes deployment of the resource adapter, deployment of an OLTP message-driven bean and deployment of Enterprise Java Beans.

Chapter 5, "BeanConnect Management Console" gives an overview of the BeanConnect Management Console.The Management Console is used to configure and administer one or more BeanConnect proxies.

Chapter 6, "Configuration of BeanConnect" describes the steps you must perform to configure a BeanConnect proxy and its components using the Management Console.

Chapter 7, "Adapting the configuration in the EIS partner" describes the configuration activities that are necessary in the EIS (Enterprise Information System) to establish communication between the application server and the EIS.

Chapter 8, "Administering BeanConnect" describes the administration tasks involved in operating the BeanConnect proxies.

Chapter 9, "Command Line Interface of the BeanConnect Management Console (MC-CLI)" describes the Management Console Command Line Interface which you can use to start BeanConnect Management Console functions via Jython scripts.

Chapter 10, "Interfaces and programming" describes how to program communication between the application server and the EIS.

Chapter 11, "Encoding and national language support" describes code conversion between the encoding used in the EIS partner and Unicode. Additionally, this chapter provides information on national language support for message output.

Chapter 12, "High availability and scalability" describes the modifications which could be necessary in the configuration of BeanConnect for heavy load operations.

Chapter 13, "Logging, diagnostics and troubleshooting" describes the variety of diagnosis utilities and trace functions.

Chapter 14, "Cobol2Java" describes the integration of BS2000/OSD COBOL applications and BeanConnect clients using Cobol2Java for mapping and converting COBOL data types to Java classes.

1.4 Changes compared to the predecessor version

BeanConnect V3.0 contains extensive changes compared to BeanConnect V2.1. The most important changes are listed below. For a complete description – in particular concerning the software configuration – please refer to the Release Notice:

New functions

- The BeanConnect resource adapter supports the JCA 1.6 specification and therefore the new generic work context and security work context contracts.
- BeanConnect V3.0 can work together with application servers that support the JCA 1.5 or JCA 1.6 specifications.

  For BeanConnect V3.0, the Oracle WebLogic Server was used as the standard application server. All the examples in this manual are based on this application server.
- Management Console Command Line Interface for the script-based administration and configuration of the BeanConnect components.

  BeanConnect provides a command line interface based on the Jython script language. It can be used to start the following administration functions from within a Jython script:

  – All proxy functions and all proxy cluster functions, i.e. the configuration and administration of resource adapters, EIS partners, inbound services, outbound services, inbound communication endpoints and outbound communication endpoints.

    You can create the objects listed here, read and modify the properties of the objects, and remove objects from the configuration. You can also perform administration functions such as checking availability or starting and stopping a BeanConnect proxy.

  – Processing of todo topics

    You can read the properties of the todo topics and delete todo topics.
- Extensions to the Management Console functionality

  – BeanConnect supports UTM cluster applications during outbound communication by making it possible to configure and address the node applications individually.

  – You can now modify the EIS partner directly in an outbound communication endpoint's properties sheet.

1.5 Notes on third-party products and literature

This manual refers to the following and other third-party products which can be used in combination with BeanConnect:

- Oracle WebLogic Server
- IBM^{TM} Communications Server for Windows systems and IBM Communications Server for Linux systems
- SNAP-IX^{TM} from Data Connection Ltd's for Solaris systems
- IBM's CICS^{TM} (Customer Information Control System)

In some cases, this manual makes concrete reference to parameters which have to be specified in the Oracle WebLogic Server. Oracle WebLogic Server 12 is a prerequisite for the current description, i.e. the specifications apply to precisely this version. If another version is used then different specifications may be required. For details, refer to the relevant Oracle documentation.

1.6 Notational conventions

This documentation uses the following notational conventions:

Convention	Meaning
. . .	Vertical ellipsis points in an example mean that information not directly related to the example has been omitted.
. . .	Horizontal ellipsis points in statements or commands mean that parts of the statement or command not directly related to the example have been omitted.
boldface text	Boldface type in text indicates all user interface items (menu items, field names, options, etc.).
`typewriter font`	Typewriter font indicates input for the system, system output and file names.
< >	Angle brackets enclose names which the user must supply. Angle brackets indicate XML statements in the examples with XML.
[ ]	Square brackets enclose optional clauses from which you can choose one or none.
{ }	Braces enclose alternative clauses from which you must choose exactly one.
|	A vertical line separates the alternatives or optional clauses.

ℹ This symbol indicates important notes and further information.

⚠ This symbol indicates a warning.

2 JCA adapter integration overview

This document describes the product BeanConnect. BeanConnect provides a JCA 1.6 compliant adapter connecting openUTM applications (Universal Transaction Monitor) and CICS applications (Customer Information Control System) to applications based on Java EE, for example the Oracle WebLogicServer.

This chapter provides an overview of JCA adapter integration in a Java EE application server environment. The Java EE Connector Architecture (JCA) is a software architecture that permits the integration of heterogeneous applications in the Java EE platform.

The chapter describes the features of BeanConnect and shows how this product is embedded in the Oracle WebLogicServer Servers environment. In detail this chapter contains the following information:

- Die JCA adapter versions
- BeanConnect architecture
- BeanConnect as a JCA-compliant resource adapter
- BeanConnect in cluster operation

2.1 Die JCA adapter versions

Java Specification Requests (JSRs) are developed within the Java Community Process (JCP) and are then agreed on in a formal, generally accessible process. The JSR documents are identified via a number and describe specifications and technologies that can be added to the Java platform. The document "JSR 322: Java EE Connector Architecture 1.6" defines standard Java interfaces for the simple integration of applications for communication with an EIS. In this document, compliance with this specification is referred to simply as " JCA 1.6 conformity". It was preceded by the document "JSR 112: J2EE^{TM} Connector Architecture 1.5". Compliance with this document is referred to as "JCA 1.5 conformity". Java EE 5 includes JCA 1.5 and Java EE 6 includes JCA 1.6.

2.1.1 JCA adapter integration

A resource adapter is specific to the Enterprise Information System (EIS) for which it was developed. It provides the system-level operations needed to communicate and operate with the EIS. A resource adapter which supports the JCA interfaces can be used with any application server which also supports these interfaces. The resource adapter reveals its capabilities to the application server through a JCA-defined API. Using the defined API, the application server can effectively incorporate the services of the resource adapter into its operations while isolating the applications themselves from the underlying implementation of the EIS. Important requirements for effective and scalable integration with EIS systems are services such as.

- connection management and pooling
- transaction management to support global transactions, i.e. transactions which involve both the application server and EIS.
- logging and tracing
- and a security framework enabling both container-managed and bean-managed sign-on.

2.1.2 JCA 1.6 contracts

All contracts of the JCA 1.6 specification are supported:

- Connection management contract

  The connection management contract enables application components to connect to an EIS and to exploit any connection pooling provided by the application server.

- Transaction management contract

  The transaction management contract enables an application server to use a transaction manager to manage transactions across multiple resource managers.

- Security contract

  The security management contract provides authentication, authorization, and secure communication between the Java EE server and the EIS.

- Lifecycle management contract

  The lifecycle management contract enables the application server to manage the lifecycle, i.e. the startup and shutdown functionality, of the resource adapter.

- Work management contract

  The work management contract enables the resource adapter to carry out work by submitting it to an application server for execution. Since the application server does the work for the resource adapter, the resource adapter need not worry about thread management. Instead, the application server manages this aspect efficiently and can use thread pooling if necessary. Although the work management contract is not required (the resource adapter can choose to manage its own thread for work), it is definitely recommended.

- Message inflow contract

  The message inflow contract allows a resource adapter to synchronously or asynchronously deliver messages to endpoints in the application server, irrespective of message style, semantics, and infrastructure.

- Generic work context contract (new in JCA 1.6)

  The generic work context contract enables a resource adapter to pass on, during inbound communication, context information that it has received from EIS to a work instance it has instructed the application server to execute.

- Transaction inflow contract

  The transaction inflow contract allows a resource adapter to propagate an imported transaction to an application server. It also makes it possible to transfer the termination of a transaction as well as calls initiated by an EIS.

- Security work context (new in JCA 1.6)

  The security work context contract enables a resource adapter to pass on security information that it has received from EIS to a work instance it has instructed the application server to execute. This functionality is referred to as "security inflow".

- Common Client Interface (CCI)

  The CCI describes a standard API client and is primarily intended to meet the requirements relating to the development of tools for application development and EAI frameworks (Enterprise Application Integration). The CCI possesses limited functionality compared to the BeanConnect-specific API

You can also find more detailed information in the JCA 1.6 specification.

2.1.3 SOA architecture

The JCA adapter supports the SOA concept (Service-Oriented Architecture).

SOA is a concept for a system architecture in which functions are implemented in the form of reusable, technically independent, loosely coupled **services**. Services can be called independently of the underlying implementations via interfaces whose specifications may be public and therefore reliable. Service interaction is performed via a specially provided communication infrastructure.

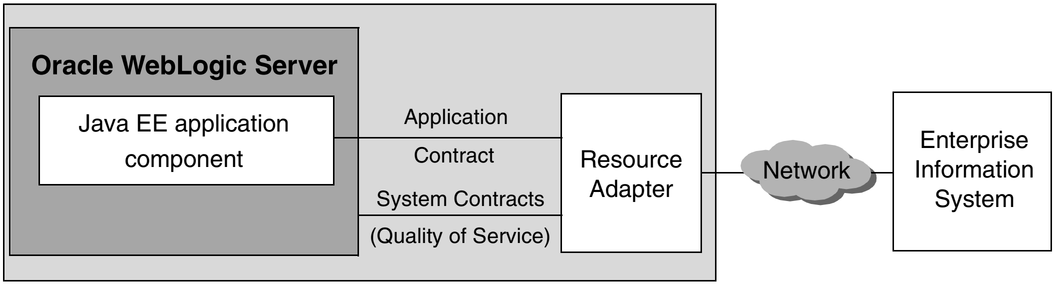
Using BeanConnect, it is possible to make components of openUTM and CICS applications available as services. At the same time, openUTM and CICS applications are able to address services in the application server.

i Further information can be found, for example, at `http://en.wikipedia.org/wiki/Service-oriented_architecture`

2.1.4 JCA adapter integration in Oracle WebLogic Server

Oracle WebLogic Server 12 is an implementation of the Java EE 6 specification and provides standardized APIs which make it possible, for example, to also use connections to remote Enterprise Information Systems (EIS). As a component of Java EE 6, the Oracle WebLogic Server supports the Java EE Connector Architecture 1.6 in accordance with the JSR322 specification. As a result, it is a simple task to deploy the BeanConnect JCA 1.6 resource adapter in the Oracle WebLogic Server 12.

Figure 1: Oracle WebLogic Server – Java EE Connection Architecture



The application components and the resource adapter are deployed using deployment descriptors which make it possible to integrate the resource adapter and the application components in the application server:

- Standard deployment descriptor `ejb-jar.xml` for application component in accordance with the Java EE specification.
- Standard deployment descriptor `ra.xml` for the resource adapter.
- Application server-specific deployment descriptors `weblogic-ra.xml` for the resource adapter and `weblogic-ejb-jar.xml` for the application component.

For a detailed description of these deployment descriptors, see Section 4.1.1, "Configuration files in the application server".

2.2 BeanConnect architecture

BeanConnect makes it possible to communicate with both openUTM and CICS Enterprise Information Systems (abbreviated to EIS partners). This chapter describes the components that are required for these partners as well as the functions that these components possess.

2.2.1 BeanConnect components

BeanConnect consists of the following components:

- The **BeanConnect resource adapter** implements the JCA 1.6 interfaces. Being a compliant JCA adapter, it is deployed in the application server and runs within the application server.

  To make it possible to continue to work with older application servers which do not yet support JCA 1.6, a JCA 1.5-compatible resource adapter providing only JCA 1.5 functionality is also supplied.

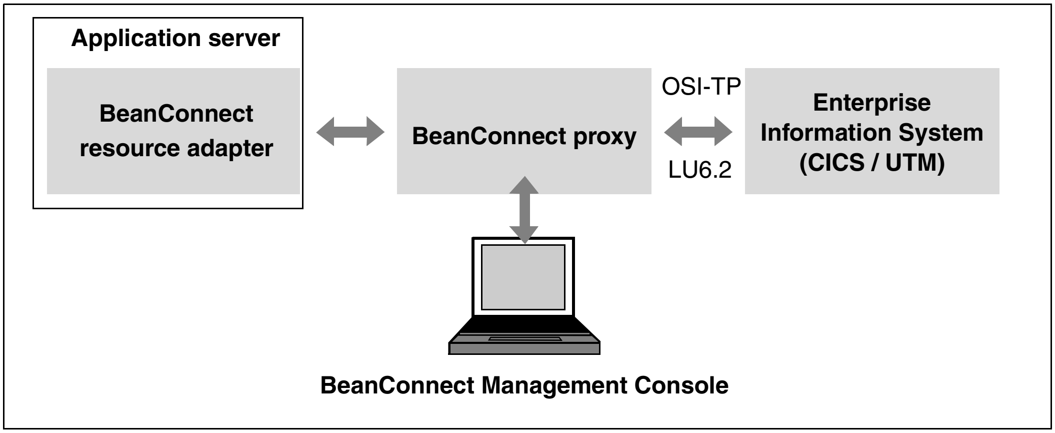
- The **BeanConnect proxy** provides the functionality of a protocol converter as well as functions for transaction control. It can be seen as an intelligent gateway. It communicates with the resource adapter running within the application server on the one hand and with the EIS on the other hand. It can be located on the same machine as the resource adapter or on a different one.

- The **BeanConnect Management Console** is a Java -based GUI used, for example, for the configuration and administration of proxies. The Management Console also provides a command line interface (MC-CLI).

  It can manage multiple proxies running on the same host as the Management Console or on remote hosts. The Management Console is not required for the configuration of outbound connections using the UPIC protocol

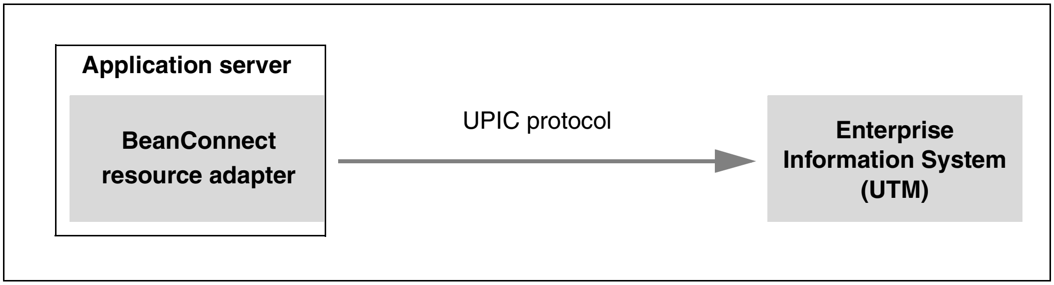
- The **BeanConnect tools** can be installed and used independently of the proxy container. These tools include the Management Console Command Handler (MC-CmdHandler) and Cobol2Java.

Figure 2: BeanConnect components


Application server
BeanConnect resource adapter
BeanConnect proxy
OSI-TP
LU6.2
Enterprise Information System (CICS / UTM)
BeanConnect Management Console

The BeanConnect proxy uses the appropriate protocol depending on the type of EIS partner, i.e. OSI-TP for openUTM partners or LU6.2 for CICS partners.

Figure 3: BeanConnect components for outbound communication with openUTM partners via UPIC protocol


Application server
BeanConnect resource adapter
UPIC protocol
Enterprise Information System (UTM)

In more complex scenarios it may be necessary to connect one or more application servers with several EISs. In this case, notice the following rules:

- BeanConnect may only be deployed once in an application server instance
- One resource adapter communicates with exactly one proxy.
- A proxy can communicate either with a single resource adapter (Section 2.2.2, "Standard operation with one resource adapter and one proxy") or with multiple resource adapters (Section 2.2.3, "Multiple resource adapter mode" or Section 2.4, "BeanConnect in cluster operation")
- One proxy can communicate with multiple EIS partners.
- One Management Console can handle multiple proxies (configuration and administration).

In addition, when BeanConnect is used in a cluster configuration, it can be operated with multiple resource adapters and multiple proxies, see Section 2.4, "BeanConnect in cluster operation".

2.2.1.1 BeanConnect resource adapter

BeanConnect is a connector resource adapter in compliance with the JCA 1.6 specification from Sun Microsystems^{TM}. It supplies standardized connectivity of openUTM and CICS applications to applications running in an application server based on the Java EE architecture and plays a fundamental role in the integration and connectivity between an EIS and an application server. It serves as the point of contact between application components, application servers and Enterprise Information Systems.

The resource adapter is provided as a RAR archive. This archive must be deployed in the application server by means of the application server.
Detailed information how to configure the resource adapter can be found in Chapter 4, "Configuration in the application server".

2.2.1.2 BeanConnect proxy

The proxy is the connecting link between the resource adapter on one side and the EIS on the other side.

It serves as container for configuration properties of the communication partners and holds all information on services, communication endpoints and message endpoints.

The proxy is responsible for transporting the messages and ensures that they are assigned to the corresponding partners and services. It provides functions for transactional security and for verification of access rights (user ID and password) on requests to and from openUTM or CICS applications. The proxy stores asynchronous messages (inbound and outbound) until they are sent to the EIS partner or the message endpoint application.

The bulk of the BeanConnect configuration is done by configuring the proxy.

The proxy contains the proxy container which is based on the openUTM transaction monitor openUTM V6.2.

You can use the Management Console to configure the proxy in three different ways

- As an openUTM proxy which communicates exclusively with EIS partners of type openUTM.

  An openUTM proxy consists only of the proxy container.

- As a CICS proxy which communicates exclusively with EIS partners of type CICS.

  A CICS proxy requires additional components to communicate with CICS applications. A separate license is required for CICS communication.

- As a combined proxy

  A combined proxy communicates both with EIS partners of type openUTM and with EIS partners of type CICS. A further, separate license is required for CICS communication.

Components of a CICS proxy

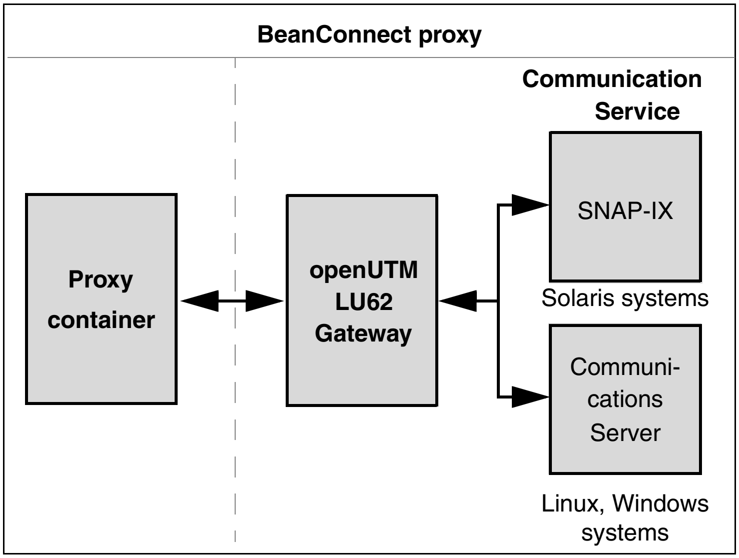
For communications with CICS applications, the proxy additionally consists of the following internal components:

- the openUTM-LU62 Gateway, implementing the LU6.2 protocol stack for interconnection with EIS partners supporting the SNA protocol LU6.2. Both transactional and non-transactional connections with CICS applications are supported.
- Communication service:
  This is a third-party product that implements the SNA stack

  This is Data Connection's SNAP-IX for Solaris Systems or IBM's Communications Server for Linux and Windows. These products are not included in the BeanConnect scope of delivery.

The following figure shows the components of the proxy for communications with CICS applications:

Figure 4: BeanConnect proxy components of a CICS proxy



The proxy and all the proxy components are managed and administered using the Management Console. The openUTM-LU62 Gateway and communication service proxy components may run either on the same computer as the proxy container or together on a separate computer as required.

If a component that is to be administered (proxy, gateway, communication service) runs on a different host from the Management Console, then an MCCmdHandler must be installed on this separate host, see also Section 2.2.1.4, "BeanConnect tools".

Please note that the communication service must be installed under a user ID which has SNA authorization.

ℹ The BeanConnect proxy is used for outbound communication via the OSI-TP and LU6.2 protocols as well as for inbound communication.

In the case of outbound communication with EIS partners of type openUTM via the UPIC protocol, the proxy is not used.

2.2.1.3 BeanConnect Management Console

The Management Console supports configuration and administration of the proxy and proxy components. It offers a Java GUI that facilitates the necessary configuration steps.

The Management Console is able to work with more than one proxy, local or remote.

- **Local** means that the proxy is located on the same computer as the Management Console and that the proxy is installed under the same user ID, or that the access rights to the user ID under which the proxy is installed have been set accordingly.
- **Remote** means that the proxy is installed on a different computer from the Management Console or that the proxy is installed on the same computer but under a different user ID which the Management Console is not able to access.

Remote access is carried out with the Management Console Command Handler (**MC-CmdHandler**). The MC-CmdHandler is a stand-alone Java program which allows the Management Console to manage remote proxies.

The administrator uses the Management Console to define all necessary information for the proxy configuration. The following information can be specified:

- General information on the proxy or the proxy cluster
- Management of the properties of resource adapters and access to the resource adapters' deployment descriptors
- Description of the EIS partners
- The services and the communication endpoints for outbound communication (see "Outbound communication" on page 37)

- The Services and the message endpoints for inbound communication (see "Inbound communication" on page 37)
- Connection to the JMX server for access to MBeans and the information and functions which these provide

Additionally for CICS partners:

- Connection to the openUTM-LU62 Gateway
- Connection between the Communication Service and the EIS partner

The configuration data is stored by the Management Console so that it is available in subsequent Management Console sessions and can be modified or completed. The Management Console creates the configuration files to be used by the proxy, the proxy components and the EIS.

You can also use the command line interface (MC-CLI) to read the configuration data of the BeanConnect configuration, see Chapter 9, "Command Line Interface of the BeanConnect Management Console (MC-CLI)".

2.2.1.4 BeanConnect tools

BeanConnect provides certain components in the form of tools which can be installed and used independently of the proxy container. These tools are not dependent on the type of EIS communication partner.

BeanConnect provides the following tools:

- MC-CmdHandler (Management Console Command Handler)

  Using separately installed MC-CmdHandlers, it is possible, for example, to administer remote resource adapters or the proxy components required for CICS partners via the Management Console without having to install the proxy container.

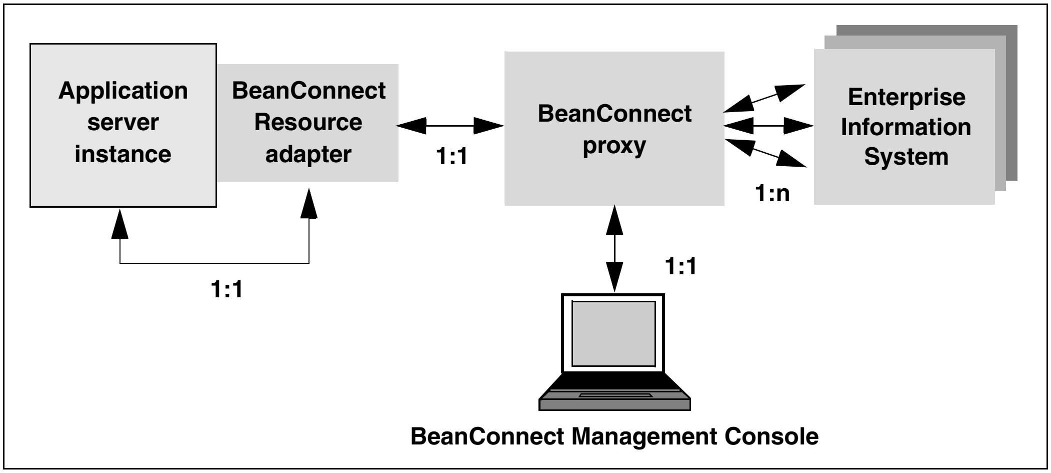
- Cobol2Java

  Cobol2Java permits the object-oriented mapping of COBOL data structures to Java classes.

2.2.2 Standard operation with one resource adapter and one proxy

During standard operation, one proxy works together with precisely one resource adapter. However, it is possible to address multiple EIS partners.

Figure 5: Relationships between the BeanConnect components in standard operation and without a cluster


Application server instance
BeanConnect Resource adapter
1:1
BeanConnect proxy
Enterprise Information System
1:n
1:1
1:1
BeanConnect Management Console

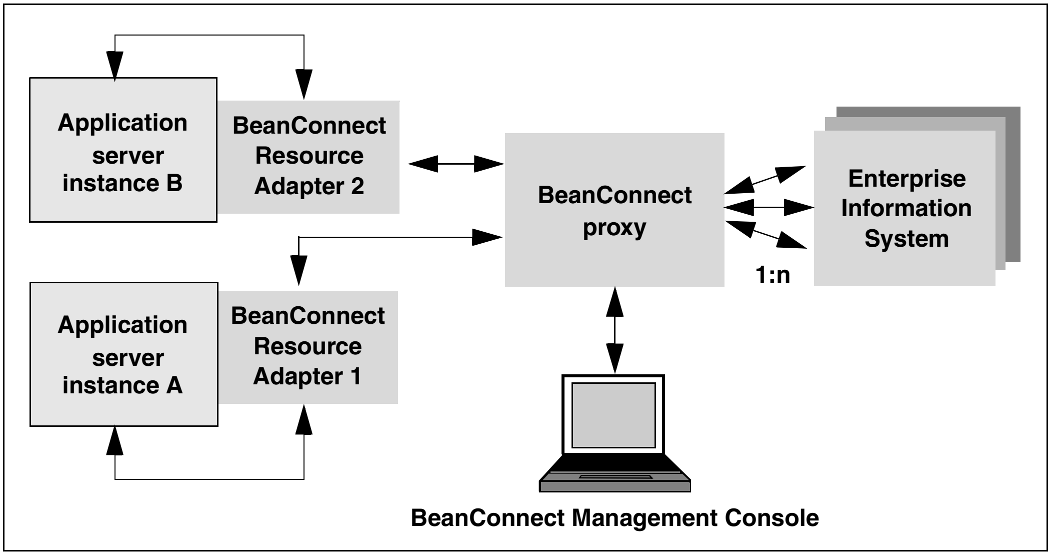
The BeanConnect resource adapter cannot be deployed more than once in the same application server instance.

2.2.3 Multiple resource adapter mode

In multiple resource adapter mode, one proxy works together with multiple resource adapters which act independently of one another and run on different application server instances. One and the same BeanConnect resource adapter cannot run multiple times on the same application server instance.

i Cluster operation and multiple resource adapter mode are mutually exclusive.

Figure 6: BeanConnect components in multiple resource adapter mode with 2 resource adapters and one proxy


Application server instance B
BeanConnect Resource Adapter 2
BeanConnect proxy
Enterprise Information System
1:n
Application server instance A
BeanConnect Resource Adapter 1
BeanConnect Management Console

A proxy can handle up to 32 resource adapters on different application server instances. Like standard operation, multiple resource adapter mode is configured and administered via the Management Console.

2.3 BeanConnect as a JCA-compliant resource adapter

BeanConnect supports different communication directions and models. BeanConnect allows outbound and inbound communication, dialog and asynchronous communication as well as transactional and non-transactional communication.

2.3.1 Outbound and inbound communication

During **outbound** communication, an EJB deployed in the application server communicates with a partner application on the EIS side of the connection.

During **inbound** communication, an EIS sends messages to a message-driven bean in a Java EE application server.

With openUTM partners BeanConnect supports the following communication variants:

- bidirectional (outbound and inbound) and transactional or non-transactional communication to OLTP applications (On-Line Transaction Processing) via the OSI-TP protocol.
- unidirectional (outbound only) and non-transactional access to openUTM applications via the UPIC protocol. This access is performed without the involvement of proxies.
- Non-transactional, asynchronous inbound communication via openUTM-Socket and the RFC1006 protocol (see Section 2.3.1.2, "Inbound communication").

With CICS partners BeanConnect supports the following communication variant:

- bidirectional (outbound and inbound) and transactional or non-transactional access to/from OLTP (On-Line Transaction Processing) applications via the LU6.2 protocol. LU6.2 is a Systems Network Architecture (SNA) protocol that supports both system-to-system communication and system-to-device communication.

In addition, BeanConnect supports non-transactional inbound communication via the proxy with any application which supports one of the protocols: UPIC, openUTM-Socket or RFC1006 (see Section 2.3.1.2, "Inbound communication").

2.3.1.1 Outbound communication

With outbound communication, an EJB deployed in the application server initiates communication with a partner application on EIS side. Outbound communication can take place as both dialog and asynchronous communication.

Figure 7: Outbound communication for openUTM partners

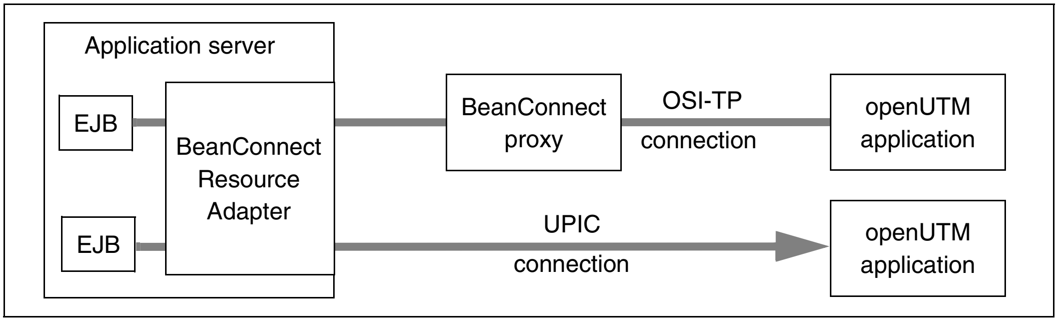
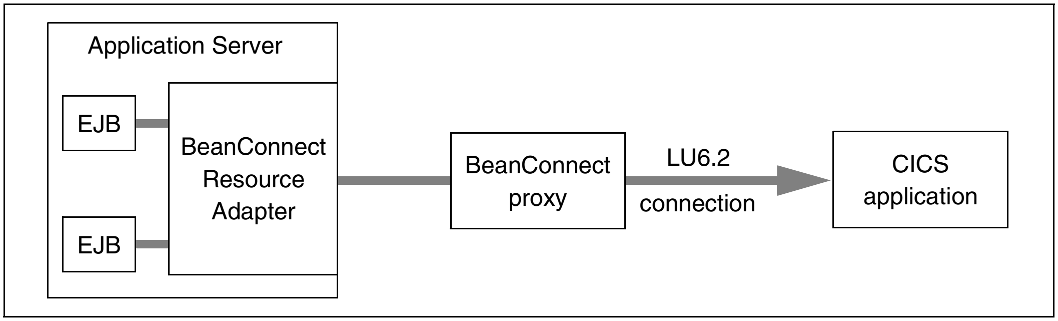


Figure 8: Outbound communication for CICS partners



2.3.1.2 Inbound communication

With inbound communication, an EIS sends messages to a message-driven bean application in a Java EE application server. The message-driven bean must implement a resource-adapter-specific message listener interface. BeanConnect supports the Service Endpoint Message Listener interface which forms part of the CCI (Common Client Interface) defined in the JCA specification and, additionally, its own message listener interface for dialog and asynchronous communication (see Section 2.3.4, "Interfaces".

BeanConnect supports the following communication types for inbound communication:

- EIS is an openUTM application
  - transactional and non-transactional communication via the OSI-TP protocol (dialog and asynchronous)
  - non-transactional communication via the openUTM socket protocol or RFC1006 protocol (only asynchronous)
- EIS is a CICS application
  - transactional and non-transactional communication via the LU6.2 protocol (asynchronous and dialog communication)
- EIS is another application
  - UPIC application: non-transactional communication via the UPIC protocol (dialog)
  - openUTM socket client or RFC1006 application: non-transactional communication via the openUTM socket protocol or RFC1006 protocol (dialog and asynchronous)

Figure 9: Inbound communication for openUTM partners

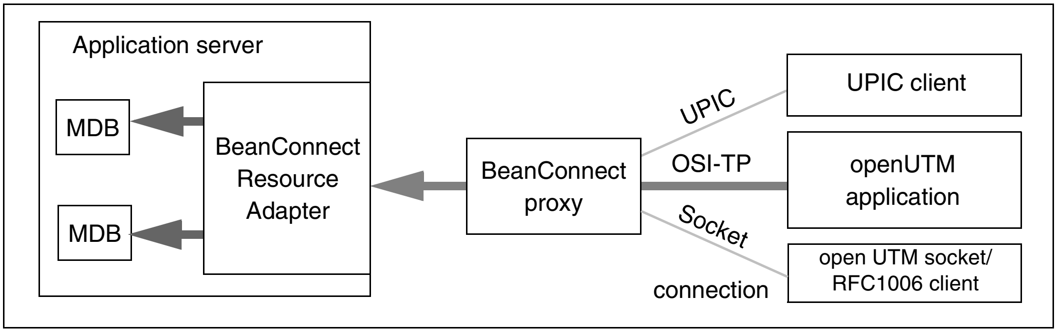
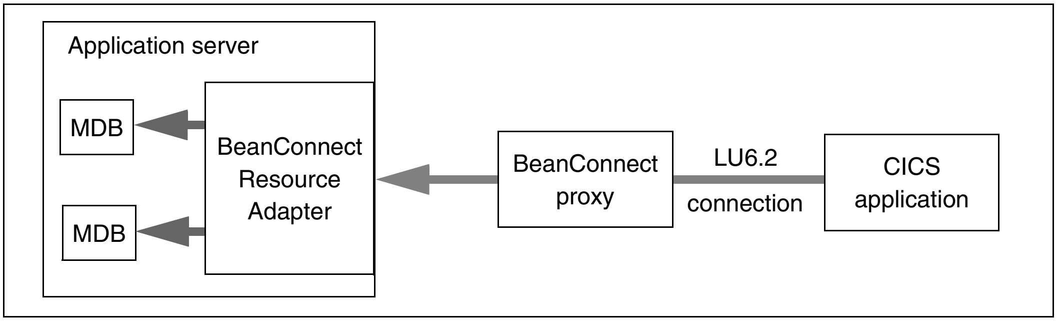


Figure 10: Inbound communication for CICS partners



2.3.2 Dialog and asynchronous communication

BeanConnect supports dialog as well as asynchronous communication for both inbound and outbound communication.

2.3.2.1 Dialog communication

In the case of dialog communication, one communication partner waits for a response from the other partner before continuing processing, i.e.:

- In the case of outbound dialog communication, the application (EJB) in the application server waits for the response from the EIS partner.
- In the case of inbound dialog communication, the EIS partner waits for the response from the message-driven bean application.

2.3.2.2 Asynchronous communication

In the case of asynchronous communication, a communication partner does not wait for a response from the other partner, i.e.:

- In the case of outbound asynchronous communication, the application (EJB) in the application server sends a message to the EIS partner without expecting a response,
- In the case of inbound asynchronous communication, the EIS partner sends a message to the message-driven bean application without expecting a response.

2.3.3 Transactional and non-transactional communication

BeanConnect supports transactional and non-transactional communication.

Transactional communication is only possible for communication with OLTP applications via the OSI-TP and the LU6.2 protocol.

2.3.3.1 Transactional communication

In the case of dialog communication, the processing in the remote application is included in the global transaction. This ensures that the distributed resources are consistent at all times, even across applications.

In the case of asynchronous communication, only transmission of the message is included in the transaction. Asynchronous jobs are transmitted exactly once in the case of transactional communication. This means that even in the event of network malfunctions or failure of an application, the asynchronous job is not lost, neither is the message duplicated.

2.3.3.2 Non-transactional communication

In the case of non-transactional communication, the processing in the remote application is independent of the local transaction. When two independent transactions interoperate, each application commits or rolls back its own transaction independently. In the event of communication failure, for instance, this can lead to inconsistent data in the different applications. This kind of communication does not ensure that asynchronous jobs are transmitted only once.

2.3.4 Interfaces

BeanConnect supports both BeanConnect-specific interfaces and standard interfaces in accordance with the JCA specification. This section presents an overview of the interfaces for outbound and inbound communication

Detailed information on the programming interfaces can be found in Chapter 10, "Interfaces and programming".

Interfaces for outbound communication

During outbound communication, an EJB deployed in the application server communicates with a partner application in the EIS system. This EJB can communicate with EIS partners via interfaces in the following packages.

- `net.fsc.jca.communication`

  The interfaces combined in the package `net.fsc.jca.communication` define proprietary BeanConnect-specific communication interfaces. These support different programming modes (like send/receive and call) and provide access to the functions supported by the underlying communication protocol.

- `net.fsc.jca.communication.cci`

  The Common Client Interface (CCI) is defined in the JCA specification. It describes a standard client API and primarily addresses the requirements of developing application development tools and EAI frameworks (Enterprise Application Integration). Compared with the BeanConnect-specific API, the CCI provides a restricted functional range.

Interfaces for inbound communication

With inbound communication, an EIS can send messages to a message-driven bean application in a Java EE application server.

The message-driven bean must implement a resource-adapter-specific message listener interface.

BeanConnect supports the following message listener interfaces:

- `net.fsc.jca.communication.AsyncOltpMessageListener`

  BeanConnect-specific interface for asynchronous communication

- `net.fsc.jca.communication.OltpMessageListener`

  BeanConnect-specific interface for dialog communication

- `javax.resource.cci.MessageListener`

  Common Client Interface (CCI) for dialog communication

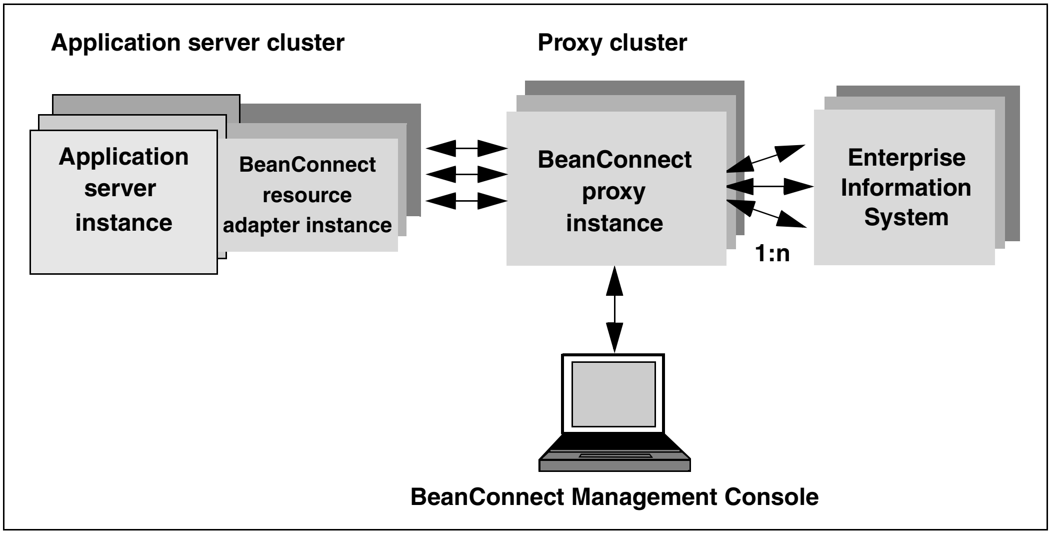
A message-driven bean which implements one of the first two message listener interfaces is called an OLTP message-driven bean.

2.4 BeanConnect in cluster operation

BeanConnect supports both clusters in the application server and proxy clusters. This means that n instances of the resource adapter can be assigned to m proxy instances.

Cluster operation is designed to increase reliability and balance the load between the instances.

Figure 11: BeanConnect components in cluster operation


Application server cluster
Proxy cluster
Application server instance
BeanConnect resource adapter instance
BeanConnect proxy instance
Enterprise Information System
1:n
BeanConnect Management Console

In the case of outbound communication, a resource adapter instance is always assigned to precisely one proxy cluster instance at any one time. If this instance fails, the resource adapter instance searches for a new proxy cluster instance. Load balancing is performed using the mechanisms present in the application server.

In the case of inbound communication, all the application server instances are always assigned to a single proxy instance. The proxy container is responsible for load balancing.

The proxy cluster is configured using the Management Console. One of the proxy instances is identified as the master instance and is used to synchronize the other proxy instances, for example when changes are made to the configuration data.

3 Installing BeanConnect

This chapter describes how to install, update and uninstall BeanConnect.

- Installing BeanConnect under Solaris systems
- Installing BeanConnect under Linux systems
- Installing BeanConnect under Windows systems
- Installing a BeanConnect resource adapter
- Installing the BeanConnect tools
- Update installation for the BeanConnect proxy container and Management Console
- Uninstalling BeanConnect
- Uninstalling the BeanConnect resource adapter
- Uninstalling the BeanConnect tools

ℹ Additional information can be found in the manual extension file `man01-en.pdf` (PDF format).

After installation, the manual extension file can be found in the BeanConnect home directory:
Solaris and Linux systems: `/Docs/English`
Windows systems: `\Docs\English`

3.1 Installing BeanConnect under Solaris systems

BeanConnect supports the Sun^{TM} Solaris operating system (SPARC).

BeanConnect contains the following product files:

- BeanConnect proxy
- openUTM
- openUTM-LU62 gateway (only for CICS partners)
- PCMX

Installing BeanConnect involves the following steps:

1. Master installation
2. Installing the BeanConnect proxy container and the Management Console

3.1.1 Master installation

Master installation must be performed under the root user ID.

You start master installation with the following command:

```
pkgadd -d MASTER_BC30A00.pkg
```

Master installation allows you to install the following packages:

- PCMX (Communications Manager UNIX OS)
- openUTM
- BeanConnect
- openUTM-LU62 gateway (for CICS partners)

Select the package(s) that you wish to install.

> To use BeanConnect, you must have installed all the packages of the master installation, but you should install only those packages that are not yet installed on your system. Please make sure that the versions of the package are identical.

If you want to install multiple packages, you do not always have to restart the master installation package. When you start the master package, the individual packages that you can install are listed in numerical order.

Example 1 Installing multiple packages

You want to install the BeanConnect (package 1) and openUTM (package 4) product files.

```
The following packages are available:
 1 BC30A00             BeanConnect
                       (sparc) 3.0A00
 2 SMAWpcmx            Communications Manager Unix OS
                       (sparc) 6.0B00
 3 SMAWutm6s           openUTM-LU62: openUTM LU6.2 comm. (using SNAP-IX)
                       (sparc) 5.1A40
 4 UTM62A00            openUTM:Universal Transaction Monitor
                       (sparc) 6.2A00

Select package(s) you wish to process (or 'all' to process all packages).
(default: all) [?,??,q]: 1 2 4
```

Packages 1, 2 and 4 will be installed one after the other.

3.1.1.1 Installing PCMX

The PCMX software must already be installed before a BeanConnect proxy can be installed on this host. If only the BeanConnect Management Console or other BeanConnect tools are installed on this host then it is not necessary to install PCMX.

1. Select the PCMX (Communications Manager UNIX OS) package from the master installation.
2. PCMX (Communications Manager UNIX OS) is installed automatically.

3.1.1.2 Installing openUTM

You must install openUTM before installing the BeanConnect proxy container. This means that it is not necessary to install openUTM if you only want to run the BeanConnect Management Console or other BeanConnect tools on this host.

1. Select the openUTM package from the master installation.
2. Specify the home directory in which openUTM is to be installed. Default: `/opt/lib`
3. Specify the base directory `<basedir>`. To do this, you must once again select the directory which you specified in step two.

3.1.1.3 Installing the BeanConnect product files

i Before you can install the BeanConnect container and/or the BeanConnect Management Console, you must first install the BeanConnect product files.

Proceed as follows:

1. Select the BeanConnect package from the master installation.
2. Specify the BeanConnect installation directory in which the BeanConnect product files are to be installed. Default: `/opt/lib`

3.1.1.4 Installing the openUTM-LU62 Gateway (for CICS partners)

1. Select the openUTM-LU62 gateway package from the master installation.
2. The openUTM-LU62 gateway is installed automatically.

3.1.1.5 Silent installation

In the case of the openUTM and BeanConnect subpackages, you need a so-called <response-file> in order to respond to the questions concerning location, owner and group. In addition, for both products you need a modified `<installation-administration-file>`, in order to deactivate the security query for root authorization.

The default file `default` is located under `/var/sadm/install/admin`.

1. Copy the default file.
2. Specify `action= nocheck`.
3. Specify the file as the `<installation-administration-file>`

On Solaris systems, you call silent installation as follows:

```
pkgadd -r <response-file> -a <installation-administration-file>
-d MASTER_BC30A00.pkg
<<EOF
<Space-separated subpackage number>
EOF
```

Example for `<response-file>`:

```
LOC="/opt/lib"
OWNER="root"
GROUP="root"
```

3.1.2 Installing the BeanConnect proxy container and the Management Console

Before you can install BeanConnect components, you must first install JDK, PCMX, openUTM, openUTM-LU62 gateway (for CICS partners) and the BeanConnect product files from the master installation.

The BeanConnect installation program can perform the following operations:

- Installing the BeanConnect proxy container and the MC-CmdHandler
- Installing the BeanConnect Management Console including the command line interface (MC-CLI)

Starting the installation procedure

After you have installed the product files, the second step is to perform a user-specific installation of the BeanConnect components.

Log in to the system using the user ID under which BeanConnect is to run. Root or administration authorization is not required for installation.

1. Switch to the directory in which you want to install the proxy container / Management Console.

   Example:

   Switch to the directory `<user_base_dir>` if you want to install the proxy container in `<user_base_dir>/<proxy_cont_name>` and the Management Console in `<user_base_dir>/<console_name>`.

2. Start the installation using the following command:

   ```
   <BC_home>/shsc/install.BeanConnect
   ```

   The default value for `<BC_home>` is `/opt/lib/bc30a00`.

3. The procedure displays a menu containing the components you can install.

   - **1 BeanConnect Proxy Container**

     This installs a BeanConnect proxy container and the MC-CmdHandler.

   - **2 Management Console (Administration Tool)**

     This installs the BeanConnect Management Console including the command line interface (MC-CLI).

Specify the number of the component(s) that you want to install. If you want to install both components, enter 1 2. If you enter `q`, the installation procedure is terminated.

Installing the BeanConnect proxy container

The installation procedure `<BC_home>/shsc/install.BeanConnect` requires the following input:

1. User base directory in which the BeanConnect proxy is to be installed

   The installation procedure sets the current directory (from which the procedure was started) as the BeanConnect user base directory `<user_base_dir>` in which the proxy container is to be installed.

   If you accept this directory with `y`, the installation procedure continues.
   If you specify `n`, the installation procedure is aborted.

   To install the proxy container in another directory, switch to this directory and start the installation again.

2. Name of the proxy container

   Specify the name of the proxy container.

   The name must be unique in your system.
   The name can have a maximum length of eight characters.
   The following characters are permitted: A,...,Z, a,...,z, 0,...,9
   Lowercase letters are converted to uppercase letters.

   The default is `BCCONT`.

   A subdirectory with the name of the proxy container is created in the BeanConnect user base directory. The files are installed in this subdirectory. The proxy container home directory is therefore `<user_base_dir>/<proxy_cont_name>` (e.g. `<user_base_dir>/BCCONT`).

   If you specify the name of a proxy container that is already installed and is located under `<user_base_dir>`, you are asked whether you want to overwrite the proxy container (new installation) or perform an update installation (see "Update installation for the BeanConnect proxy container and Management Console" on page 72).

3. openUTM home directory

   Specify the openUTM home directory. This environment variable is pre-assigned the value of the environment variable UTMPATH. The installation procedure also searches for appropriate openUTM versions that are installed on the computer and displays a list of the installed UTM versions as of a minimum version (openUTM V6.2). You can select the desired home directory from this list.

   If openUTM is not installed in the specified directory, the following error message is displayed:
   `openUTM not found!`
   In this case, check the specified home directory or the installation of openUTM (see Section 3.1.1.2, "Installing openUTM").

4. JAVA home directory

   Prerequisite: JDK must already be installed on your system before you can install the proxy.

   The installation procedure asks for the name of the JAVA home directory. You have to specify the directory explicitly. No name is suggested. You have to specify a fully qualified directory name. The procedure then checks your input.

   If JDK is not installed in the specified directory, the following error message is displayed:
   `JDK not found!`
   Check the specified home directory or install JDK first.

5. Acceptance

   The installation procedure displays the name of the archive from which the proxy is installed. Installation must be performed from the archive
   `<BC_home>/CPIO.BCCont`.

   The default value for `<BC_home>` is `/opt/lib/bc30a00`.

   Accept this archive.

6. Port number of the proxy container

   BeanConnect needs a range of one hundred port numbers. You are asked to enter the port number which specifies the beginning of this range.

   The range of port numbers must not be used by other proxy containers or other applications. You have to specify a different port number here.

   ```
   Start Value of the Port Number Range for BeanConnect
   ```

   Specify the start value of the port number range. The next hundred port numbers are reserved for BeanConnect. Values between 1025 and 32667 are permitted for the port number. Default: 31000.

7. Password for administration of the proxy container

   Specify the password. Default: admin

8. The proxy container is installed once this dialog has been completed.

i You can repeat the installation procedure several times if you want to install multiple proxy containers.

Installing the Management Console

The installation procedure requires you to make the following entries:

1. User base directory in which the Management Console is to be installed

   The installation procedure sets the current directory (from which the procedure was started) as the BeanConnect user base directory <`user_base_dir`> in which the Management Console is to be installed.

   If you accept the directory with `y`, the installation procedure continues. If you specify `n`, the installation procedure is aborted.

2. JAVA home directory

   Prerequisite: JDK must already be installed on your system before you can install the Management Console.

   The installation procedure asks for the name of the JAVA home directory. You have to specify a fully qualified directory name. The procedure then checks your input.

   If JDK is not installed in the specified directory, the following error message is displayed:
   `JDK not found!`
   Check the specified Java home directory or install JDK first.

3. Select the bit architecture

   The installation procedure asks you to specify the bit architecture under which the Management Console is to run. This allows you to force the Management Console to run under a different bit architecture from the JDK that you have specified in the Java home directory.

4. Directory for the Management Console configuration files

   The configuration files are stored in a subdirectory of the BeanConnect user base directory <`user_base_dir`>.

   Specify the subdirectory <`console_name`>. Default: `console`

   The Management Console home directory is consequently
   `<user_base_dir>/<console_name>.`

If you specify the name of a Management Console that is already installed in `<user_base_dir>`, you are asked whether you want to overwrite the Management Console (new installation) or perform an update installation (see "Update installation for the BeanConnect proxy container and Management Console" on page 72).

5. Oracle WebLogic Server configuration

   The following query is issued:

   ```
   Configuration for Oracle WebLogic Server?([y]|n)
   ```

   If you enter `n`, continue at point 6.

   If you enter `y`, you will see the prompt:

   ```
   Please enter product installation directory of WebLogic Server or
   enter <RETURN> for remote machine
   ```

   If you enter <RETURN>, you will see the message:

   ```
   Please copy "wlclient.jar" and  "wljmxclient.jar" to
   "<console-Installationsverzeichnis>/bin/weblogic" after this
   installation
   ```

6. Acceptance

   The installation procedure displays the name of the archive from which the Management Console is installed. Installation must be performed from the archive `<BC_home>/CPIO.console`.

   The default value for `<BC_home>` is `/opt/lib/bc30a00`.

   Accept this archive.

3.2 Installing BeanConnect under Linux systems

BeanConnect supports the Linux operating system.

BeanConnect contains the following product files:

- BeanConnect proxy
- openUTM
- openUTM-LU62 gateway (for CICS partners)
- CMX

Installation of BeanConnect involves the following steps:

1. Master installation
2. Installing the BeanConnect proxy container and the Management Console

3.2.1 Master installation

Master installation must be performed under the root user ID.

You start master installation with the following command:

```
rpm -i MASTER_BC30A00.rpm --ignorearch --prefix=<BC_install>
```

Specify the desired BeanConnect installation directory for `prefix=<BC_install>`. If `prefix` is not specified, the default BeanConnect installation directory `/opt/lib/` is used.

Master installation allows you to install the following packages:

- CMX (Communications Manager UNIX OS)
- openUTM
- BeanConnect
- openUTM-LU62 gateway (for CICS partners)

Select the package(s) that you wish to install.

ℹ To use BeanConnect, you must have installed all the packages of the master installation necessary for the desired functions. However, you should install only those packages that are not yet installed on your system. Please make sure that the versions of the package are identical.

If you want to install multiple packages, you do not always have to restart the master installation package. When you start the master package, the individual packages that you can install are listed in numerical order.

Example 2 Installing multiple packages

You want to install the openUTM (package 3) and BeanConnect (package 4) product files:

```
The following packages are available:
  1  PCMX         Communications Manager LINUX
                 6.0B00
  2  UTMLU62      gw4   openUTM-LU62: openUTM LU6.2 communication
                 5.1A40
  3  UTM62A00     openUTM:Universal Transaction Monitor
                 6.2A00
  4  BC30A00      BeanConnect
                 3.0A00

Select package(s) you wish to process (or 'all' to process all packages).
(default: all) [?,??,q]: 1 3 4
```

Packages 1, 3 and 4 will be installed one after the other.

> ⚠ The installation operation itself runs in the background. You must therefore wait for a short time for the termination messages after the shell prompt has been output to determine whether installation has been performed successfully. You should therefore start by logging off once the messages have been output.

3.2.1.1 Installing PCMX

The PCMX software must already be installed before a BeanConnect proxy can be installed on this host. If only the BeanConnect Management Console or other BeanConnect tools are installed on this host then it is not necessary to install PCMX.

1. Select the PCMX (Communications Manager UNIX OS) package from the master installation.
2. PCMX is installed automatically in the directory `/opt/lib/`.

3.2.1.2 Installing openUTM

You must install openUTM before installing the BeanConnect proxy container. This means that it is not necessary to install openUTM if you only want to run the BeanConnect Management Console or other BeanConnect tools on this host.

1. Select the openUTM package from the master installation.
2. openUTM is installed automatically in the directory which you specified in `prefix`.

3.2.1.3 Installing the BeanConnect product files

i Before you can install the BeanConnect container and/or the BeanConnect Management Console, you must first install the BeanConnect product files.

Porceed as follows:

1. Select the BeanConnect package from the master installation.
2. The product files are installed automatically in the directory which you specified in `prefix`.

3.2.1.4 Installing the openUTM-LU62 Gateway (for CICS partners)

1. Select the openUTM-LU62 gateway package from the master installation.
2. openUTM-LU62 gateway is installed automatically in the directory `/opt/lib/`.

3.2.1.5 Silent installation

You call silent installation on Linux systems in exactly the same way as the master installation. In addition, you must enter the numbers of the subpackages – separated by spaces – in the file `/response`.

Please note the following:
If you do not enter anything in the`/response` then **all** the subpackages are installed.

3.2.2 Installing the BeanConnect proxy container and the Management Console

Before you can install BeanConnect, you must first install JDK, PCMX, openUTM, openUTM-LU62 gateway (for CICS partners) and the BeanConnect product files from the master installation.

The BeanConnect installation program can perform the following functions:

- Installing the BeanConnect proxy container and the MC-CmdHandler
- Installing the BeanConnect Management Console (including the command line interface MC-CLI)

Starting the installation procedure

After you have installed the product files, the second step is to perform a user-specific installation of the BeanConnect components.

The Korn shell (`ksh`) must be used for installation.

Log in to the system using the user ID under which BeanConnect is to run. Root or administration authorization is not required for installation.

1. Switch to the directory in which you want to install the proxy container / Management Console.

   Example:

   Switch to the directory `<user_base_dir>` if you want to install the proxy container in`<user_base_dir>/<proxy_cont_name>` and the Management Console in `<user_base_dir>/<console_name>`.

2. Start the installation using the following command:

   ```
   <BC_home>/shsc/install.BeanConnect
   ```

   The default value for `<BC_home>` is `/opt/lib/bc30a00`.

3. The procedure displays a menu containing the components you can install.

   - `1 BeanConnect Proxy Container`

     This installs a BeanConnect proxy container and the MC-CmdHandler.

   - `2 Management Console (Administration Tool)`

     This installs the BeanConnect Management Console including the MC-CLI.

   Specify the number of the component(s) that you want to install. If you want to install both components, enter 1 2. If you enter `q`, the installation procedure is terminated.

Installing the BeanConnect proxy container

The installation procedure `<BC_home>/shsc/install`.BeanConnect requires the following input:

1. User base directory in which the BeanConnect proxy is to be installed

   The installation procedure sets the current directory (from which the procedure was started) as the BeanConnect user base directory `<user_base_dir>` in which the proxy container is to be installed.

   If you accept this directory with `y`, the installation procedure continues.
   If you specify `n`, the installation procedure is aborted.

   To install the proxy container in another directory, switch to this directory and start the installation again.

2. Name of the proxy container

   Specify the name of the proxy container.

   The name must be unique in your system.
   The name can have a maximum length of eight characters.
   The following characters are permitted: A,...,Z, a,...,z, 0,...,9
   Lowercase letters are converted to uppercase letters.

   The default is `BCCONT`.

i A subdirectory with the name of the proxy container is created in the BeanConnect user base directory. The files are installed in this subdirectory. The proxy container home directory is therefore
`<user_base_dir>/<proxy_cont_name>`
(e.g. `<user_base_dir>/BCCONT`).

If you specify the name of a proxy container that is already installed and located under `<user_base_dir>`, you are asked whether you want to overwrite the proxy container (new installation) or perform an update installation (see "Update installation for the BeanConnect proxy container and Management Console" on page 72).

3. openUTM home directory

   Specify the openUTM home directory. This environment variable is pre-assigned the value of the environment variable UTMPATH. The installation procedure also searches for appropriate openUTM versions that are installed on the computer and displays a list of the installed UTM versions as of a minimum version (openUTM V6.2). You can select the desired home directory from this list.

   If openUTM is not installed in the specified directory, the following error message is displayed:
   `openUTM not found!`
   In this case, check the specified home directory or the installation of openUTM (see Installing openUTM on page 54).

4. Java home directory

   Prerequisite: JDK must already be installed on your system before you can install the proxy.

   The installation procedure asks for the name of the JAVA home directory. You have to specify the directory explicitly. No name is suggested. You have to specify a fully qualified directory name. The procedure then checks your input.

   If JDK is not installed in the specified directory, the following error message is displayed:
   `JDK not found!`
   In this case, check the specified home directory or install JDK first.

   Please note that on Linux systems, JDK must have the same bit characteristics as the proxy!

5. Acceptance

   The installation procedure displays the name of the archive from which the proxy is installed. Installation must be performed from the archive
   `<BC_home>/CPIO.BCCont`.

   The default value for `<BC_home>` is `/opt/lib/bc30a00`.

   Accept this archive.

6. Port number of the proxy container

   BeanConnect needs a range of one hundred port numbers. You are asked to enter the port number which specifies the beginning of this range.

   The range of port numbers must not be used by other proxy containers or other applications.

   ```
   Start Value of the Port Number Range for BeanConnect
   ```

   Specify the start value of the port number range. The next hundred port numbers are reserved for BeanConnect. Values between 1025 and 32667 are permitted for the port number. Default: 31000.

7. Password for the administration of the proxy container

   Specify the password. Default: admin

8. The proxy container is installed once this dialog has been completed.

i You can repeat the installation procedure several times if you want to install multiple proxy containers.

Installing the Management Console

The installation procedure requires you to make the following entries:

1. User base directory in which the Management Console is to be installed

   The installation procedure sets the current directory (from which the procedure was started) as the BeanConnect user base directory <`user_base_dir`> in which the Management Console is to be installed.

   If you accept the directory with `y`, the installation procedure continues. If you specify `n`, the installation procedure is aborted.

2. JAVA home directory

   Prerequisite: JDK must already be installed on your system before you can install the Management Console.

   The installation procedure asks for the name of the JAVA home directory. You have to specify a fully qualified directory name. The procedure then checks your input.

   If JDK is not installed in the specified directory, the following error message is displayed:
   `JDK not found!`
   Check the specified Java home directory or install JDK first.

3. Directory for the Management Console configuration files

   The configuration files are stored in a subdirectory of the BeanConnect user base directory `<user_base_dir>`.

   Specify the subdirectory `<console_name>`. Default: `console`

   The Management Console home directory is consequently `<user_base_dir>/<console_name>`.

   If you specify the name of a Management Console that is already installed, you are asked whether you want to overwrite the Management Console (new installation) or perform an update installation (see "Update installation for the BeanConnect proxy container and Management Console" on page 72).

4. Oracle WebLogic Server configuration

   The following query is issued:

   ```
   Configuration for Oracle WebLogic Server?([y]|n)
   ```

   If you entern, continue with point 5.

   If you enter `y`, you will see the prompt:

   ```
   Please enter product installation directory of WebLogic Server or
   enter <RETURN> for remote machine
   ```

   If you enter <RETURN>, you will see the message:

   ```
   Please copy "wlclient.jar" and  "wljmxclient.jar" to
   "<Console-Installationsverzeichnis>/bin/weblogic" after this
   installation
   ```

5. Acceptance

   The installation procedure displays the name of the archive from which the Management Console is installed. Installation must be performed from the archive `<BC_home>/CPIO.console`.

   The default value for `<BC_home>` is `/opt/lib/bc30a00`.

   Accept this archive.

3.3 Installing BeanConnect under Windows systems

BeanConnect supports the Microsoft Windows operating system.

3.3.1 Master installation

You need administration authorization to perform master installation.

Insert the BeanConnect DVD in the drive. The DVD starts automatically and the setup menu appears.

> The file `%WIN32SYS%\drivers\etc\hosts` can be saved under the name `hosts.save` and modified during installation. In this case, a message is output. If network problems occur, you should copy the `hosts.save` file back.

You can install the following packages:

- PCMX32 V5.0A70
- openUTM V6.2A00
- BeanConnect V3.0A00
- openUTM-LU62 gateway V5.1A41 (for CICS partners)

Select the package(s) in the **Choose Products** dialog box that you wish to install.

> To use BeanConnect, you must have installed all the packages of the master installation that are necessary for the desired functions. However, you should install only those packages that are not yet installed on your system. Please make sure that the versions of the package are identical.

Click the **Next** button to start master installation.

3.3.1.1 Installing PCMX

You must install PCMX before installing openUTM and BeanConnect.

1. Select PCMX-32 installation in the **Choose Products** dialog box which is described above.
2. In the **PCMX-32 Installation** window, click **Next >** to open the following sequence of dialog boxes:

- In the **Information** dialog box, click **Readme** to open the readme file which contains detailed information. Click **License** to display the license agreement.
- In the **... choose your preferred Installation method** dialog box, check the **Standard Installation** box (default) and click **Next >**.
- The **Ready to install** dialog box then lists the installation directories. Click **Next >** to continue.
- In the **Installation completed** dialog box, click **Finish**.

3.3.1.2 Installing openUTM

You must install openUTM before installing BeanConnect.

1. Select openUTM installation in the **Choose Products** dialog box which is described above.
2. Follow the installation program instructions and select the appropriate options.

   openUTM checks the system requirements and makes sure that sufficient disk space is available. If the system fails to meet the requirements, installation is rejected.

   If an older version of openUTM exists on your PC and you decide to install openUTM in the same directory, the older version is automatically overwritten.

   If an older version of openUTM exists on your PC and you decide to install openUTM in another directory, the valid version is the one that was most recently installed.
3. Reboot the system if you are requested to do so by the installation routine.

3.3.1.3 Installing BeanConnect

> Before you can install BeanConnect, you must first install JDK, PCMX and openUTM.

The BeanConnect installation program performs the following tasks:

- It installs the BeanConnect proxy container and the MC-CmdHandler
- It installs the BeanConnect Management Console including the command line interface MC-CLI

> If you want to update an existing installation, you find detailed information in section "Update installation for the BeanConnect proxy container and Management Console" on page 72.

Installation procedure

To install BeanConnect, perform the steps described below:

1. Select **BeanConnect Proxy** in the **Choose Products** dialog (see "Master installation" on page 60).

   The **Installation** dialog opens. You can navigate through the sequence of dialog boxes that are now displayed by clicking **Next >** and **< Back**.

2. **Welcome** screen
   Click **Next** to start the installation.

3. **Information** dialog box
   Click **Readme** to view the BeanConnect readme file.

4. **Choose Language** dialog box
   Select the language for the Management Console online help system.

5. **Common Resources** dialog box
   Decide whether or not you want to install the common resources. These include archives, libraries and documents etc. When performing a new installation, you must always also install the common resources, see section "Installing common resources".

   If you do not want to install the common resources, the **Common Resources Directory** dialog box asks you to specify the directory containing the common resources that you intend to use.

   In the case of an update installation, you must ensure that the current common resources are installed.

6. **Installation Option** dialog box
   Select **Normal installation** to perform a new installation.

   Select **Update installation** to update an existing BeanConnect installation. For information on update installations, see "Update installation for the BeanConnect proxy container and Management Console" on page 72.

   Select the entry **Update installation without deinstallation** to update an existing BeanConnect installation without uninstalling the existing installation. For information on update installations, see Section 3.6, "Update installation for the BeanConnect proxy container and Management Console".

7. **Proxy Components** dialog box
   Check the appropriate box(es) to specify the component(s) which you want to install.

   You can select one or both of the components. The subsequent installation process depends on the components that you have selected for installation.

   - **BeanConnect Proxy Container**

     Installs and initiates a BeanConnect proxy container and the MC-CmdHandler.

     It first opens the dialog box **openUTM Directory** and then the dialog box **Java Directory**.

     open**UTM Directory** dialog box
     Specify the openUTM home directory which contains the `ex\libwork.dll` file.

     JAVA Directory dialog box
     Specify the JAVA home directory. This is the directory on your system that contains the JDK `lib\tools.jar` and `bin\java.exe` files.

   - **Management Console (Administration Tool)**

     Installs the BeanConnect Management Console with which you can configure and administer BeanConnect proxies.

     If you only activate the option **Management Console (Administration Tool)** then the dialog box **Java Directory** is opened.

     JAVA Directory dialog box
     Specify the JAVA home directory. This is the directory on your system that contains the JDK `lib\tools.jar` and `bin\java.exe` files.

Installing common resources

If you have chosen to install the common resources in the **Common Resources** dialog box then the following dialog boxes are displayed

1. **Target BeanConnect Common Resources Directory to install** dialog box

   Enter the home directory in which the BeanConnect common resources are to be installed. Default: `C:\BeanConnect\BC30A00`.

2. **BeanConnect Common Resources Program Group** dialog box

   Select the program group in which you want the installation program to save the BeanConnect icons for the common resources. You can either select an existing program group or create a new one.

   By default, the installation program creates the following program group:

   `BeanConnect V3.0A00`

Installing the BeanConnect proxy container

i If a proxy container of the same name already exists then this must be shut down correctly before you perform this installation.

If you select **BeanConnect Proxy Container** in the **Proxy Components** dialog box, the following installation program dialog opens:

1. **Proxy Container Options** dialog box
   Specify the name, the port number and the password of the proxy.

   - **Name for Proxy Container**

     Name of the proxy container. Default: `BCCont`

     The name must be unique in your system.
     The name can have a maximum length of eight characters.
     The following characters are permitted: A,...,Z, a,...,z, 0,...,9

   - **Begin of port interval for Proxy Container**

     BeanConnect needs a range of 100 port numbers. You are asked to enter the port number which specifies the beginning of this range.

     The range of port numbers must not be used by other proxy containers or other applications.

     `Start Value of the Port Number Range for BeanConnect`

     Specify the start value of the port number range. The next hundred port numbers are reserved for BeanConnect. Values between 1025 and 32667 are permitted for the port number. Default: 31000.

- **Password for Proxy Container**

  Specify the admin password. Default: admin

2. **Target Proxy Container Directory to install** dialog box
   Specify the proxy container home directory in which the proxy container is to be installed.

   By default, the proxy container is installed in the BeanConnect home directory `<BC_home>\<proxy_cont_name>`.

   Click the **Browse** button to select a different directory.

3. **Proxy Container Program Group** dialog box
   Select the program group in which you want the installation program to save the BeanConnect icons for the proxy container. You can either select a program group that is already present or create a new program group.

   By default, the installation program creates the program group:

   `BeanConnect V3.0A00\<proxy_cont_name>`

4. **Permit ports through the windows firewall** dialog box

   Yes (default value) causes the installation program to enable the ports used to operate the Management Console command handler (MC-CmdHandler) for the use of the firewall.

Installing the BeanConnect Management Console
If you select **Management Console (Administration Tool)** in the **Proxy Components** dialog box, the following installation program dialog opens:

1. **Target BeanConnect Management Console Directory to install** dialog box
   Specify the Management Console home directory in which the Management Console is to be installed.

   By default, the Management Console is installed in the Management Console home directory `<BC_home>\Console`.

   Click the **Browse** button to select a different directory.

2. **BeanConnect Management Console Program Group** dialog box
   Select the program group in which you want the installation program to save the BeanConnect icons for the Management Console. You can either select a program group that is already present or create a new program group.

   By default, the installation program creates the program group:

   `BeanConnect V3.0A00\Management Console`

Installing the BeanConnect proxy container and Management Console

If you select both **BeanConnect Proxy Container** and **Management Console (Administration Tool)** in the **Proxy Components** dialog box, the proxy container and the Management Console are installed in a single step.

See above for a detailed description of the dialog boxes.

In this case, the following dialog boxes are displayed only once:

- **JAVA Directory**
- **Target BeanConnect Common Resources Directory to install**
- **BeanConnect Proxy Program Group**

Concluding the installation

You have now made all the entries and settings required for installation.

1. **Ready to Install!** dialog box
   Click the **Install** button to start the installation.
2. **Installation Completed!** dialog box
   Click **Finish** to conclude installation.

Depending on the selected option(s), the installation program creates a proxy container including the "internal" MC-CmdHandler and/or the Management Console.

> i The BeanConnect product files are installed automatically during the installation process.

3.3.1.4 Installing the openUTM-LU62 Gateway (for CICS partners)

To install the openUTM-LU62 gateway:

1. Select openUTM-LU62 gateway installation in the **Choose Products** dialog box which is described above.
2. Follow the installation program instructions and select the appropriate options.
3. Select the home directory in which openUTM-LU62 gateway is to be installed.
4. In the **Installation completed** dialog box, click **Finish**.

3.3.2 Installing the BeanConnect proxy container via the command line

The BeanConnect installation program `setup.exe` also has a command line interface which is described below.

To install the product via the command line, open a DOS prompt window. Switch to the following directory:

```
<DVD_drive>\BEANCONNECT-PROXY\Windows
```

Then enter the following command:

```
Setup.exe [/S /M=paramfile] [/E=errorfile]
```

The `/S` option specifies that installation is to be performed without a GUI (Graphical User Interface). In this case, you must also specify the `/M=paramfile` parameter. The installation parameters are then taken from the file `paramfile` in which you can set the installation parameters and directories yourself. The file `paramfile` must have the format described below. If you want to use a parameter file, you should specify all the parameters in the file. In this case, there is no guarantee that the default values will be used.

If `/S` is not specified, installation is started via the GUI.

With `/E=errorfile`, you can specify a log file which records the parameters used, the progress of the installation process, and any error messages that occur. If you specify the `/S` option, you are also advised to use the `/E` option.
If you do not specify `/E`, any error messages that occur may be lost. If it does not already exist, the file `errorfile` is created by the installation program. If it does exist, the file is overwritten.

Format of the parameter file

An example of a parameter file is shown below. This file (`BeanConnect_Install.ini`) is provided. You can adapt it to suit your requirements.

```
;Online Help Language;
A: English;
B: German

OHELP_CFG=A                                                  Language for the online help system

;Installation Option                                          Update installation?
UPDATEINSTALL=N

;BeanConnect Proxy directory to update UPDBEANCONNECTDIR=

INST_PROXY=Y                                                 Create proxy container

INST_CONSOLE=Y                                               Create Management Console

; openUTM Directory                                          openUTM home directory
UTMPATH=c:\openUTM-Server
```

```
; JAVA Home Directory
```

Java home directory

```
JAVA_HOME=C:\jdk1.6

; BeanConnect Common Resources Directory
ROOTDIR=C:\BeanConnect
```

BeanConnect home directory

```
; ------------------------------------------------------
; Proxy parameters. Only evaluated if INST_PROXY=Y and
UPDATEINSTALL=N
; -----------------------------------------------------

PROXY_NAME=BCCont
```

Name of the proxy container

```
PORT_BEGIN=31000
```

Port for communication with the BeanConnect proxy container

```
PROXY_PASSWORD=admin
```

Administration password of the proxy container

```
; Target Proxy Directory

PROXYDIR=C:\BeanConnect\BCCont
```

Directory for installation of the proxy container

```
; ------------------------------------------------------
;Management Console. Only evaluated if INST_CONSOLE=Y
; ------------------------------------------------------

; Only for Update installation.
; Existing Console directory to update

UPDCONSOLEDIR=C:\BeanConnect\Console
```

Directory of the existing Management Console

```
; Target Console directory
CONSOLEDIR=C:\BeanConnect\Console
```

Directory for the installation of the Management Console

3.4 Installing a BeanConnect resource adapter

You install the BeanConnect resource adapter using the JAR file `BC30A00_RA.jar`.

This file contains the resources listed below:

- the RAR file `BC30A00_RA.rar` containing the resource adapter classes for JCA 1.6. For information on using the RAR file see Section 4.1, "Overview".
- the RAR file `BC30A004JCA15.rar` which contains the resource adapter classes for JCA 1.5. For more information on using the RAR file, see Section 4.1, "Overview".
- the file `BeanConnectDev.jar` for compiling the EJBs
- The file `BeanConnectVerifyer.jar`, which must be appended to the BeanConnect RAR file if you want to use the Sun JEE Verifier (part of the Sun Glassfish V2 JEE5 Application Server).
- The directory `config`: log4j configuration files for resource adapter logging
- The directory `Docs`: Documents
- The directory `encoding`: Encoding example
- The directory `JavaDoc`: JavaDoc for the user API.
- The directory `scid`: Diagnostic tool

Installing a resource adapter in systems with a graphical user interface
In systems with a graphical user interface, proceed as follows:

1. In the relevant system, open a window for command entry, e.g. Shell or DOS prompt
2. Switch to the directory in which the JAR file is located
3. Unpack the JAR file using the command
   ```
   java -jar BC30A00_RA.jar
   ```
4. Follow the instructions output by the graphical installation program and define the installation directory for the resource adapter

Installing a resource adapter in systems without a graphical user interface

The file `RA-auto.xml` is supplied to permit installation on systems without a graphical user interface. Proceed as follows:

1. Open the file `RA-auto.xml` with a text editor and enter the required installation path for the resource adapter in the `<installpath>` tag
2. In the relevant system, open a window for command entry, e.g. Shell or DOS prompt.
3. Switch to the directory in which the JAR file is located
4. Unpack the JAR file using the command
   ```
   java -jar BC30A00_RA.jar RA-auto.xml
   ```
   This installs the resource adapter automatically. The files are unpacked in the current directory without modifying `RA-auto.xml`.

3.5 Installing the BeanConnect tools

BeanConnect provides a number of tools in addition to the proxy and the resource adapter. These tools can be installed separately and are present in the form of JAR archives. The tools are available in the following archives:

MC-CmdHandler	`BC30A00_MCCmdHandler.jar`
Cobol2Java	`BC30A00_Cobol2Java.jar`

Installing the BeanConnect tools in systems with a graphical user interface

In systems with a graphical user interface, proceed as follows:

1. In the relevant system, open a window for command entry, e.g. Shell or DOS prompt.
2. Switch to the directory in which the JAR file is located.
3. Unpack the JAR file using the following command
   `java -jar <jar-archive>`

   `<jar-archive>` is the name of the relevant JAR archive displayed at the top of the list.
4. Follow the instructions output by the graphical installation program and define the installation directory for the tool, possibly together with additional parameters.

Installing the BeanConnect tools in systems without a graphical user interface

An Auto.xml file is supplied for each tool to permit installation on systems without a graphical user interface. This file has the name `<tool>-auto.xml`, where `<tool>` is the name of the tool to be installed `MC-CmdHandler`, `Cobol2Java`).

Proceed as follows:

1. Open the Auto-xml- file with a text editor and enter the required installation path for the tool in the `<installpath>` tag. In the case of some tools, you must also enter configuration data such as port number or password for additional tags that have been commented out.
2. In the relevant system, open a window for command entry, e.g. Shell or DOS prompt.
3. Switch to the directory in which the JAR file is located.
4. Unpack the JAR file using the following command
   `java -jar <jar-archive> <tool>-auto.xml`

   This installs the tool automatically.
5. In the case of the tool MC-CmdHandler, you must enter the path to JDK in the file `javaenv.cmd` (Windows systems) or `javaenv.sh` (Linux, Solaris systems). You do this by editing the file with a text editor.

3.6 Update installation for the BeanConnect proxy container and Management Console

BeanConnect allows you to perform update installations. An update installation enables you to install a new BeanConnect version or a new patch while retaining the configuration data of the installed proxy container and Management Console.

This section provides an overview of:

- Update installation under Solaris systems
- Update installation under Linux systems
- Update installation under Windows systems

3.6.1 Update installation under Solaris systems

This section describes how you can update the proxy container and Management Console installation under Solaris systems. If you obtain new software, then start the master installation and install the new products (see Section 3.1, "Installing BeanConnect under Solaris systems").

Starting the update installation

The update installation is performed in the same way as a new installation.

Start the installation procedure with the following command:

```
<BC_home>/shsc/install.BeanConnect
```

The default value for `<BC_home>` is `/opt/lib/bc30a00`.

As in a new installation, the procedure displays a menu listing the two components that can be installed. You select the components for which you want to install an update:

```
Configuration Options

1 BeanConnect Proxy Container
2 Management Console (Administration Tool)
```

The installation procedure recognizes when you specify a home directory that already contains one of the components and checks whether the version is updatable. If it is, you are asked whether you want to perform an update installation or overwrite the existing version (new installation).

An update installation is performed for the selected BeanConnect components.

BeanConnect proxy container

1. The proxy container home directory is copied and saved in the directory `<BC_home>/<proxy_cont_name>.save`. This directory is not changed by the update installation. All the old configuration data is retained.
2. If required, the existing proxy container can be uninstalled. The following applies depending on whether or not the old and new directories are identical:
   - If the old and new directories are identical then the old proxy container is renamed, a new proxy container is installed and the data is transferred from the old proxy container.
   - If the old and new directories are different then a new proxy container is installed. The data is then transferred from the old proxy container.
3. The old configuration is moved to the new proxy container.

Management Console

1. The Management Console home directory is copied and saved in the directory `<BC_home>/<console_name>.save`. This directory is not changed by the update installation. All the old configuration data is retained.
2. The following applies depending on whether or not the old and new directories are identical:
   - If the old and new directories are identical then the old Management Console is renamed, a new Management Console is installed and the data is transferred from the old Management Console.
   - If the old and new directories are different then a new Management Console is installed. The data is then transferred from the old Management Console.
3. The old configuration files are read from `<console_name>.save` and copied to the new Management Console home directory.

3.6.2 Update installation under Linux systems

This section describes how you can update the proxy container and Management Console installation under Linux. If you obtain new software then start the master installation and install the new products (see Table 3.2, "Installing BeanConnect under Linux systems").

Starting the update installation

The update installation is performed in the same way as a new installation.

Start the installation procedure with the following command:

```
<BC_home>/shsc/install.BeanConnect
```

The default value for `<BC_home>` is `/opt/lib/bc30a00`.

As in a new installation, the procedure displays a menu listing the two components that can be installed. You select the components for which you want to install an update:

```
Configuration Options

1 BeanConnect Proxy Container
2 Management Console (Administration Tool)
```

The installation procedure recognizes when you specify a home directory that already contains one of the components and checks whether the version is updatable. If it is, you are asked whether you want to perform an update installation or overwrite the existing version (new installation).

An update installation is performed for the selected BeanConnect components.

BeanConnect proxy container

1. The proxy container home directory is copied and saved in the directory `<BC_home>/<proxy_cont_name>.save`. This directory is not changed by the update installation. All the old configuration data is retained.
2. If required, the existing proxy container can be uninstalled. The following applies depending on whether or not the old and new directories are identical:
   - If the old and new directories are identical then the old proxy container is renamed, a new proxy container is installed and the data is transferred from the old proxy container.
   - If the old and new directories are different then a new proxy container is installed. The data is then transferred from the old proxy container.
3. The old configuration is moved to the new proxy container.

Management Console

1. The Management Console home directory is copied and saved in the directory `<BC_home>/<console_name>.save`. This directory is not changed by the update installation. All the old configuration data is retained.
2. The following applies depending on whether or not the old and new directories are identical:
   – If the old and new directories are identical then the old Management Console is renamed, a new Management Console is installed and the data is transferred from the old Management Console.

   – If the old and new directories are different then a new Management Console is installed. The data is then transferred from the old Management Console.
3. The new Management Console is installed in the Management Console home directory
4. The old configuration files are read from `<console_name>.save` and copied to the new Management Console home directory.

3.6.3 Update installation under Windows systems

This section describes how you can update the proxy container and Management Console installation under Windows. If you obtain new software then start the master installation and install the new products (see Section 3.3, "Installing BeanConnect under Windows systems").

Starting the update installation

You start the installation in the usual way (see Section 3.3, "Installing BeanConnect under Windows systems"). You should then click the **BeanConnect Proxy** item in the setup menu.

Only the dialog boxes that differ from those displayed during a new installation are described here. You work with all the other dialog boxes in the same way as for a new installation (see Section 3.3, "Installing BeanConnect under Windows systems").

- **Installation Option** dialog box

  If you want to update an existing BeanConnect installation, select the option **Update installation**.

  If you want to update an existing BeanConnect installation without uninstalling the existing installation, select **Update installation without deinstallation**.

- **Existing Proxy Container Directory to update** dialog box

  Specify the proxy coantainer home directory that is to be updated.

- **Existing BeanConnect Management Console directory to update** dialog box

  Specify the Management Console home directory that is to be updated.

  For the subsequent procedure, see Section 3.3.1.3, "Installing BeanConnect".

The update installation is performed for the BeanConnect components selected in the **Configuration Options** dialog box (see Section , "Installation procedure").

BeanConnect proxy container update installation

1. The proxy container home directory is copied and saved in the directory `<BC_home>\<proxy_cont_name>.save`. This directory is not changed by the update installation. All the old configuration data is retained.
2. The existing proxy container is uninstalled on demand.
3. A new proxy container is installed in the proxy container home directory.
4. The old configuration is moved to the new proxy container.

Management Console update installation

1. The Management Console home directory is copied and saved in the directory `<BC_home>\<console_name>.save`. This directory is not changed by the update installation. All the old configuration data is retained.
2. The existing Management Console is uninstalled on demand.
3. The new Management Console is installed in the Management Console home directory.
4. The old configuration files are read from `<console_name>.save` and copied to the new Management Console home directory.

3.7 Uninstalling BeanConnect

This section describes how to uninstall BeanConnect:

- Uninstalling BeanConnect under Solaris systems
- Uninstalling BeanConnect under Linux
- Uninstalling BeanConnect under Windows

3.7.1 Uninstalling BeanConnect under Solaris systems

This section describes how to uninstall BeanConnect under Solaris systems.

Uninstalling the BeanConnect proxy container

To uninstall the proxy container you simply have to delete the home directory of the proxy container.

Uninstalling the BeanConnect Management Console

To uninstall the Management Console you simply have to delete the home directory of the Management Console.

Uninstalling the BeanConnect product files

You use the following command to uninstall the BeanConnect product files:

```
pkgrm BC30A00
```

Uninstalling PCMX

You use the following command to uninstall the supplied PCMX:

```
pkgrm SMAWpcmx
```

Uninstalling openUTM

You use the following command to uninstall openUTM:

```
pkgrm UTM<version>
```

Uninstalling openUTM-LU62 Gateway (for CICS partners)

You use the following command to uninstall the openUTM-LU62 gateway:

```
pkgrm SMAWutm6s
```

3.7.2 Uninstalling BeanConnect under Linux

This section describes how to uninstall BeanConnect under Linux.

Uninstalling the BeanConnect proxy container

To uninstall the proxy container you simply have to delete the home directory of the proxy container.

Uninstalling the BeanConnect Management Console

To uninstall the Management Console you simply have to delete the home directory of the Management Console.

Uninstalling the BeanConnect product files

You use the following command to uninstall the BeanConnect product files:

```
rpm -e BC30A00
```

Uninstalling PCMX

You use the following command to uninstall PCMX:

```
rpm -e PCMX-<version>
```

To obtain the exact package name, enter `rpm -qa | grep PCMX` .

Uninstalling openUTM

You use the following command to uninstall openUTM:

```
rpm -e UTM<version>
```

Uninstalling openUTM-LU62 Gateway

You use the following command to uninstall the openUTM-LU62 gateway:

```
rpm -e UTMLU62-<version>
```

3.7.3 Uninstalling BeanConnect under Windows

This section describes how to uninstall BeanConnect under Windows.

Uninstalling the BeanConnect proxy container
To uninstall the proxy container select:

Start - Programs - BeanConnect V3.0A00 - Proxy <proxy_cont_name> - Uninstall

Alternatively, you can use the following commands:

1. Start the uninstall program by clicking **Start - Settings - Control Panel - Software**.
2. Select **BeanConnect V3.0A00 <proxy_cont_name>** and click the **Remove** button.

The uninstallation of the proxy container comprises:

- Deletion of the directories: `<BC_home>\<proxy_cont_name>`, `<BC_home>\<proxy_cont_name>.SAVE`
- Deletion of `<BC_home>\MCINFO` if the directory is empty
- The product files will **not** be deleted (`<BC_home>\lib`, `<BC_home>\Docs`)

Uninstalling the BeanConnect Management Console
To uninstall the Management Console select:

Start - Programs - BeanConnect V3.0A00 - Management Console - Uninstall

Alternatively, you can use the following commands:

1. Start the uninstall program by clicking **Start - Settings - Control Panel - Software**.
2. Select **BeanConnect V3.0A00 Management Console** and click the **Remove** button.

The uninstallation of the Management Console comprises:

- Deletion of the directories: `<BC_home>\<console_name>`, `<BC_home>\<console_name>.SAVE`
- The product files will **not** be deleted (`<BC_home>\lib`, `<BC_home>\Docs`)

Uninstalling the BeanConnect common resources

To uninstall the BeanConnect common resources, select:

Start - Programs - BeanConnect V3.0A00 - Uninstall Common Resources

Alternatively, you can use the following commands:

1. Start the uninstall program by clicking **Start - Settings - Control Panel - Software**.
2. Select **BeanConnect V3.0A00 Common Resources** and click the **Remove** button.

The uninstallation of the product files comprises:

- Deletion of the directories: `<BC_home>\lib`, `<BC_home>\Docs`
- Deletion of `<BC_home>\MCINFO` if the directory is empty

⚠ If you uninstall the BeanConnect common resources, the proxy container and the Management Console that are installed in the same <BC_home> directory will no longer function.

Uninstalling PCMX

1. Start the uninstall program by clicking **Start - Settings - Control Panel - Software**.
2. Select **PCMX-32 <version>** and click the **Remove** button.

Uninstalling openUTM

To uninstall openUTM, select:

Start - Programs - openUTM-Server - Uninstall openUTM-Server

Alternatively, you can use the following commands:

1. Start the uninstall program by clicking **Start - Settings - Control Panel - Software**.
2. Select **openUTM-Server <version>** and click the **Remove** button.

Uninstalling openUTM-LU62 Gateway (for CICS partners)

To uninstall the openUTM-LU62 gateway, select:

Start - Programs - openUTM-LU62 - Uninstall openUTM-LU62

Alternatively, you can use the following commands:

1. Start the uninstall program by clicking **Start - Settings - Control Panel - Software**.
2. Select **openUTM-LU62 <version>** and click the **Remove** button.

3.8 Uninstalling the BeanConnect resource adapter

To uninstall the resource adapter you simply have to delete the copied or extracted files and directories that you unpacked from the JAR file when installing the resource adapter.

3.9 Uninstalling the BeanConnect tools

You uninstall a BeanConnect tool by deleting the tool's installation directory.

The only special case is the MC-CmdHandler tool. If the MC-CmdHandler is configured as a service (see Section 6.10.2.3, "Configuring an MC-CmdHandler as a service"), then you must remove this first. To do this, proceed as follows:

- Windows systems:

  Call the script `MCCmdHandler_UnInstSrv.cmd`.

- Linux and Solaris systems

  If you want to uninstall an individual service, delete the corresponding line from the file `/etc/init.d/bcmccmdhandler.dat`.

  If you want to uninstall the entire service, then you must delete the files `/etc/init.d/bcmccmdhandler.sh` (incl. symbolic links) and `/etc/init.d/bcmccmdhandler.dat`.

  You need system administrator authorizations to perform these activities. You may therefore need to ask the system administrator to perform this task.

4 Configuration in the application server

It is necessary to configure settings for BeanConnect both when deploying the BeanConnect resource adapter in the application server and when deploying Enterprise Java Beans (EJB) in the application server.

This chapter contains information on configuring outbound communication via the various protocols (OSI-TP for openUTM partners, LU6.2 protocol for CICS partners, UPIC protocol for openUTM partners) as well as on configuring inbound communication.

This chapter also describes how you prepare logging in the resource adapter and the issues that you have to take into account when operating in multiple resource adapter or cluster mode.

It contains information on the following topics:

- Deploying and undeploying the resource adapter
- Configuring general properties for the resource adapter
- Deploying an Enterprise JavaBean for OSI-TP / LU6.2
- Configuring outbound communication via UPIC
- Setting configuration properties for inbound communication
- Preparing resource adapter logging
- Special characteristics of multiple resource adapter mode
- Special characteristics in cluster operation

4.1 Overview

The resource adapter is supplied as a so-called BeanConnect RAR archive.

Only one BeanConnect resource adapter may be deployed per instance in the application server.

The deployment of more than one resource adapter may result in unpredictable errors.

The BeanConnect RAR archive for JCA 1.6 is named `BC30A00.rar, the` BeanConnect RAR archive for JCA 1.5 is named `BC30A004JCA15.rar`. Both archives contain various items including the deployment descriptor `ra.xml`.

4.1.1 Configuration files in the application server

The resource adapter and EJB configuration properties are defined in the following files:

- `ra.xml`

  Default deployment descriptor for the resource adapter.

  This file is present in the BeanConnect RAR archive and defines the resource adapter's general properties.

- `weblogic-ra.xml`

  WebLogic-specific deployment descriptor for the resource adapter.

  This file describes the WebLogic-specific settings for the resource adapter. Some of the entries in `weblogic-ra.xml` refer to entries in `ra.xml`.

- `ejb-jar.xml`

  Default deployment descriptor for EJBs.

  This file describes various items including the EJB properties that are relevant for communications. The entries refer to entries in `weblogic-ra-jar.xml` and `weblogic-ejb-jar.xml`.

- `weblogic-ejb-jar.xml`

  WebLogic-specific deployment descriptor for EJBs.

  You need this file during inbound communication in order to assign a message-driven bean to a resource adapter and during outbound communication in order to assign a JNDI name (JNDI = Java Naming and Directory Interface) to the employed connection factories.

The entries refer to entries in `weblogic-ra.xml`.

ℹ Some of the properties of the EJBs can be described in annotations instead of in deployment descriptor files. Below, only the description of the properties of the EJBs with deployment descriptor is presented.

The section below describes the procedure you may adopt dueing configuration.

4.1.2 Configuration steps for outbound and inbound communication

The procedure depends on whether you want to operate outbound communication via OSI-TP/ LU6.2, outbound communication via UPIC, and/or inbound communication. Each of these possibilities is presented separately below. However, different rules apply for multiple resource adapter and cluster operation.

In the case of outbound communication (OSI-TP / LU6.2 and UPIC), you must define resource references in ejb-jar.xml.

Procedure for outbound communication via OSI-TP / LU6.2

If you want to use BeanConnect for outbound communication via OSI-TP / LU6.2 in default mode (one proxy, one resource adapter, no cluster) then you must perform the following activities:

- Defining general properties in ra.xml
- Defining the general properties of the resource adapter in weblogic-ra.xml and Defining general and connection-specific properties for OSI-TP / LU6.2 in weblogic-ra.xml.
- Pack the file `weblogic-ra.xml` under the META-INF subdirectory of the BeanConnect RAR archive before deployment.
- Deploying the resource adapter
- Deploying an Enterprise JavaBean for OSI-TP / LU6.2
- Preparing resource adapter logging

Procedure for outbound communication via UPIC

If you want to use BeanConnect for communications via UPIC then you must perform the following activities:

- Defining the general properties of the resource adapter in weblogic-ra.xml and Defining general and connection-specific properties for OSI-TP / LU6.2 in weblogic-ra.xml.
- Before deployment, insert the file `weblogic-ra.xml` in the subdirectory META-INF of the BeanConnect RAR archive.
- Deploying the resource adapter
- Deploying an Enterprise JavaBean for UPIC
- Preparing resource adapter logging

Procedure for inbound communication

If you want to use BeanConnect for inbound communication in default mode (one proxy, one resource adapter, no cluster) then you must perform the following activities:

- Defining general properties in ra.xml
- Defining the general properties of the resource adapter in weblogic-ra.xml
- Before deployment, insert the file `weblogic-ra.xml` in the subdirectory META-INF of the BeanConnect RAR archive
- Deploying the resource adapter
- Configuration steps for outbound and inbound communication
- Preparing resource adapter logging

Procedure with multiple resource adapters and in cluster operation

When working with multiple resource adapters or in cluster operation, additional settings must be made in the file `ra.xml`. For further details, see:

- Special characteristics of multiple resource adapter mode
- Special characteristics in cluster operation

4.2 Configuring general properties for the resource adapter

The general properties are described in the standard deployment descriptor `ra.xml` and in the WebLogic-specific deployment descriptor `weblogic-ra.xml`. The resource adapter's standard deployment descriptor `ra.xml` is present in the BeanConnect RAR archive. The file `ra.xml` contains:

- The general configuration properties for the connection between the resource adapter and the proxy
- The definition of the connection factories supported by BeanConnect
- The properties supported by these connection factories together with their default values

It can be thought of as a template for configuring the connection.

The file `weblogic-ra.xml` contains:

- the WebLogic-specific configuration properties for the resource adapter
- the WebLogic-specific configuration properties for the connection factories

Before you deploy the BeanConnect RAR archive, you must adapt the general configuration properrties in the files `ra.xml` and `weblogic-ra.xml` as required.

4.2.1 Defining general properties in ra.xml

The general configuration properties are relevant for outbound communication via OSI-TP / LU6.2 and for inbound communication.

Alternatively you can deploy the resource adapter with the predefined settings. Then specify the values by using the application server's GUI. These settings can be lost while undeploying.

Adapting the ra.xml file

You can adapt `ra.xml` in two ways

- with the BeanConnect Management Console provided that the BeanConnect RAR archive and the Management Console are located on the same host, or an MC-CmdHandler is installed on the same computer as the BeanConnect RAR archive and is configured in the Management Console.

When you do this, the deployment descriptor is modified directly in the RAR archive. You can either edit the values of the properties (**Edit ra.xml of BeanConnect Resource Adapter RAR…** command) or update them on the basis of the values that have been configured for a defined proxy (**Update ra.xml of BeanConnect Resource Adapter RAR** command). Updating has the advantage that the Management Console determines the correct value for the property `proxyURL`.

For more information, see Section 6.5, "Configuring the BeanConnect resource adapter" on page 189).

- Manually using a text editor.

1. Extract the file `ra.xml` from the BeanConnect RAR archive:

   ```
   jar xf BC30A00.rar META-INF/ra.xml
   ```

2. Use a text editor of your choice to modify the following general configuration properties in the file `ra.xml`:

   proxyURL (outbound and availability check of the proxy)

   transactionLogging (outbound only)

   transactionLogDir (outbound only)

   inboundListenerPort (inbound and availability check of the resource adapter)

   RevisionNumber

3. Insert the file `ra.xml` in the BeanConnect RAR archive in the subdirectory `META-INF` again:

   ```
   jar uf BC30A00.rar META-INF/ra.xml
   ```

In multiple resource adapter mode or in cluster operation, it is necessary to make additional settings, see Section 4.7, "Special characteristics of multiple resource adapter mode" and Section 4.8, "Special characteristics in cluster operation".

proxyURL

The `proxyURL` defines the way the resource adapter assigns the proxy.

`proxyURL` is defined globally for all connections.

Definition: `oltp://<host>:<port>/<name>`

Explanation:

	`<host>`	Host on which the proxy container is installed `<host>` can be specified as a symbolic name or as an IPv4 address.
	`<port>`	Port number of the proxy container + 4
	`<name>`	Application name of the proxy container (BCU<port>)

Default: `oltp://localhost:31004/BCU31004`

Example:

```
<config-property>
  <description>BeanConnect Proxy URL for OLTP outbound
         communication</description>
  <config-property-name>proxyURL</config-property-name>
  <config-property-type>java.lang.String
  </config-property-type>
  <config-property-value>oltp://proxyhost:31004/BCU31004
  </config-property-value>
</config-property>
```

If you use the Management Console function **Update ra.xml of BeanConnect Resource Adapter RAR** then the correct values are set for `proxyURL`.

If you edit `ra.xml`, you must determine the value for `<port>` from the value specified during the installation of the proxy (port +4). You must enter a value for `<name>` that consists of the prefix BCU and `<port>`.

The Management Console displays the proxy URL to be used in the properties dialog of the resource adapter. The proxy URL cannot be modified after installation.

For information on configuration for cluster operation, see Section 4.8, "Special characteristics in cluster operation".

transactionLogging

This attribute defines whether or not BeanConnect is to write persistent transaction logs for transactions with EIS partners during outbound communication.

Definition: `[NONE | FILE]`

Explanation: Activates or deactivates persistent transaction logging

`NONE`: No persistent transaction logs are written.

`FILE`: Persistent transaction logs are written. The directory to which the logs are written is specified in the attribute `transactionLogDir`.

Default: `NONE`

Example:

```
<config-property>
  <description>BeanConnect transaction logging:
      possible values are NONE | FILE
  </description>
  <config-property-name>transactionLogging
  </config-property-name>
  <config-property-type>java.lang.String
  </config-property-type>
  <config-property-value>FILE
  </config-property-value>
</config-property>
```

During transaction recovery operations following a program or system crash, persistent transaction logs make it possible for BeanConnect to provide information about the status of the transactions that were being processed at the time of the crash. Activating this option has an impact on performance since two file access operations are required for every transaction which is terminated with two-phase commit.

If `FILE` is specified then it is also necessary to enter a value for the attribute `transactionLogDir`.

If transaction logging is configured then the resource adapter writes a separate transaction log for each transaction. The file name consists of the prefix `tx.` and a number.

A transaction log file is written on Prepare and deleted on Commit or Rollback, i.e. it is normally temporary. However, situations exist in which it is retained:

- If the resource adapter is terminated between Prepare and Rollback or Commit. This type of log file is retained until the recovery has been completed for this transaction.
- If a heuristic decision has been made for a transaction. This type of log file is retained for an indefinite period.

All the transaction log files are read in when the resource adapter is started. New transaction log files are written for transactions which have the status Prepared after start-up or for which a heuristic decision has been made.

i For transactions associated with heuristic decisions, the transaction log files are retained for an indefinite period. They should therefore be deleted from time to time. The creation time of the log files may serve as a criterion when deciding which log files are to be deleted. Files whose creation date and time correspond to the last resource adapter start-up contain heuristic logs and the application server has not called forget() for these transactions; these files can be deleted. To identify the application start time, BeanConnect writes a file during the start phase after processing the transaction log files. This file has the name `tx.startup-complete.<date>.<time>` and is written in the transaction log directory.

BeanConnect deletes old `startup-complete` files on the next start-up.

transactionLogDir

This attribute defines the directory to which BeanConnect is to write the persistent transaction logs. A value must be specified for this attribute if the attribute `transactionLogging` has been assigned the value `FILE`.

Definition: Name of a directory

Default: `persistence\BeanConnect`

Example:

```
<config-property>
  <description>Directory where transaction log files are to be
               stored (only needed if transactionLogging=FILE)
  </description>
  <config-property-name>transactionLogDir
  </config-property-name>
  <config-property-type>java.lang.String
  </config-property-type>
  <config-property-value>persistence/BeanConnect
  </config-property-value>
</config-property>
```

The directory can be specified with an absolute or relative path name. Any relative path specification is understood to be relative to the application server's home directory.

If the directory specified for `transactionLogDir` does not exist, BeanConnect creates it.

inboundListenerPort

For inbound communication, the resource adapter listens at a socket port for connection requests initiated by the proxy. You must specify the port number of this socket port with the global configuration property `inboundListenerPort`. This value must be adapted before deploying the resource adapter.

Definition: <port number>

Explanation: Port number of the socket port on which the resource adapter listens for inbound communication requests.

0 means that no inbound communication is possible.

Default: 31099

Example:

```
<config-property>
  <description>Resource Adapter Listener Port for Inbound
     Communication
  </description>
  <config-property-name>inboundListenerPort
  </config-property-name>
  <config-property-type>java.lang.String
  </config-property-type>
  <config-property-value>31099
  </config-property-value>
</config-property>
```

Even if no inbound functionality is used, you should specify a value here if you want to check the availability of the resource adapter as seen by the proxy (e.g. using the Management Console). You should only enter the value 0 if you also do not require this functionality.

The port number that is defined with `inboundListenerPort` must match the listener port number that you specified via the Management Console when adding the resource adapter to a proxy.

RevisionNumber

This attribute indicates the revision level of the file `ra.xml`. You should increase the value of the attribute `revisionNumber` every time you modify the `ra.xml` file in order to make it easier to identify possible inconsistencies in the configuration.

Example:

```
<config-property>
  <description>Revision number of the ra.xml. This number
    should be incremented with each change of the resource
    adapter properties.
  </description>
  <config-property-name>revisionNumber
  </config-property-name>
  <config-property-type>java.lang.String
  </config-property-type>
  <config-property-value>2
  </config-property-value>
</config-property>
```

4.2.2 Defining the general properties of the resource adapter in weblogic-ra.xml

The WebLogic-specific properties of the resource adapter are described in the deployment descriptor file `weblogic-ra.xml`.

The following general configuration properties are relevant:

- The JNDI name of the resource adapter is defined in the element `<jndi-name>` in the form in which it must be specified during inbound communication in the element `resource-adapter-jndi-name` of the file `<weblogic-ejb-jar.xml>` for each OLTP message-driven bean.
- The property `<enable-global-access-to-classes>` must be set to `true` in order to prevent errors during the deployment of the applications.
- In the element `<connector-work-manager>`, the tag `<max-concurrent-long-running-requests>` specifies the number of long-running work instances. The default value is 10. .

  Set this number to one higher than the number of work processes for the BeanConnect proxy container. Note that in a cluster configuration, you must take account of the work processes of all the proxy containers in the cluster.
- In the `<security-work-context>` element of the `<security>` block, you can use the tag `<inbound-mapping-required>` to specify whether you want to use case 1 or case 2 of the security inflow types described in the JCA specification. .

  The default value is `false` and stands for case 1. In this case, each user ID that is to be propagated by EIS in the application server must also be configured in the application server. The passwords that have to be defined for the user IDs in the application server are not validated for inbound communication.

  In case 2, you must configure a mapping of the EIS user ID to a user ID in the application server in the element `<security-work-context>`.

  See also „Configurations- properties in the weblogic-ra.xml file“ auf Seite 97.

4.2.3 Deploying and undeploying the resource adapter

Prerequisite

You should not deploy the resource adapter until you have entered the general configuration settings in the file `ra.xml` and the WebLogic-specific configuration settings in the file `weblogic-ra.xml`, see Section 4.2.1, "Defining general properties in ra.xml", Section 4.2.2, "Defining the general properties of the resource adapter in weblogic-ra.xml", Section 4.3.1, "Defining general and connection-specific properties for OSI-TP / LU6.2 in weblogic-ra.xml" and Section 4.4.1, "Defining general connection-specific properties for UPIC in weblogic-ra.xml".

4.2.3.1 Deploying the resource adapter

When deploying applications under Oracle WebLogic Server, you can use the following methods depending on the server startup mode:

- Production mode as startup mode

  Automatic deployment is globally inactive.

  The following three deployment possibilities are available:

  – Deployment in the browser using the WebLogic Server Administration Console

  – Deployment using the `weblogic.Deployer` tool

  – Deployment with the Oracle WebLogic Scripting Tool

- Development mode as startup mode

  The WebLogic Server instances can deploy and update applications automatically if these files are present in the directory `domain_name/autodeploy`. Oracle recommends only using this method for applications in "single-server" configurations.

  This method is not generally recommended for the BeanConnect resource adapter.

For the BeanConnect resource adapter, it is advisable to use one of the deployment methods that are also possible in production mode in development mode, see below.

Example:

Proceed as follows to deploy the resource adapter in the browser using the WebLogic Server Administration Console:

1. Create the file `weblogic-ra.xml`.
2. Add `weblogic-ra.xml`to the `META-INF` subdirectory of the BeanConnect RAR archive: `jar uf BC30A00.rar META-INF/ weblogic-ra.xml`
3. Start the web browser, log in at the WebLogic Server Administration Console and perform the following steps:

   a) Navigate to `<domainName>>Deployments` and click `install`

   b) Navigate to the directory containing the BeanConnect RAR archive, select the archive and click `<Next>`.

      The WebLogic Server Administration Console now performs other operations.

   c) Recommendation: Choose a value less than 100 (default value) for the deployment order, e.g. `98`.
      This means that applications that are dependent on the resource adapter can be added subsequently via autodeployment.

For the other two deployment methods (weblogic.Deployer tool and WebLogic Scripting Tool), please note the comments in the documentation for Oracle WebLogic Server under "Deploying Applications and Modules with weblogic.Deployer" or "Oracle WebLogic Scripting Tool". ol".

4.2.3.2 Update deployment of the resource adapter

Before perming an update deployment or new deployment, all the OLTP message-driven bean applications that reference the resource adapter must be stopped.

- To perform the update deployment, navigate to `<domainName>>Deployments` in the WebLogic Server Administration Console, select the installed BeanConnect resource adapter and click `<update>`.
- Once the update deployment is complete, you can start the OLTP message-driven bean applications again.

4.2.3.3 Undeploying the resource adapter

Before an undeployment, it is necessary to stop all the OLTP message-driven bean applications that reference the resource adapter.

To perform the undeployment, navigate to `<domainName>>Deployments` in the WebLogic Server Administration Console, select the installed BeanConnect resource adapter and click `<delete>`.

4.2.4 Example of an ra.xml File

Example 3 shows the definition of the general configuration properties in the file `ra.xml`.

Example 3 Configuration properties in the file ra.xml

The section with the general configuration properties in the file `ra.xml` has the following layout:

```
...
<resourceadapter>

<resourceadapter-class>
  net.fsc.jca.BeanConnect.ResourceAdapterJBImpl
</resourceadapter-class>
<config-property>
  <description>Resource Adapter Listener Port for Inbound Communication
  </description>
  <config-property-name>inboundListenerPort
  </config-property-name>
  <config-property-type>java.lang.String</config-property-type>
  <config-property-value>31099</config-property-value>
</config-property>
<config-property>
  <description>BeanConnect Proxy URL for OLTP Outbound   Communication</description>
  <config-property-name>proxyURL</config-property-name>
  <config-property-type>java.lang.String</config-property-type>
  <config-property-value>oltp://localhost:31004/BCU31004</config-property-value>
</config-property>
<config-property>
  <description>BeanConnect transaction logging: possible values are NONE |
FILE</description>
<!-- If NONE is set, no persistent transaction logging will be performed.
If FILE is set, transaction log files will be written to the directory specified in
property
transactionLogDir. -->
<config-property-name>transactionLogging</config-property-name>
<config-property-type>java.lang.String</config-property-type>
```

```
<config-property-value>FILE</config-property-value>
</config-property>
<config-property>
<description>Directory where transaction log files are to be stored (only needed if
transactionLogging=FILE)</description>
<!-- The path name of the dircetory may be specified as absolute or relative path.
A relative path is relative to the home directory of the application server. -->
<config-property-name>transactionLogDir</config-property-name>
<config-property-type>java.lang.String</config-property-type>
<config-property-value>persistence/BeanConnect</config-property-value>
</config-property>
<config-property>
  <description>Revision number of the ra.xml. This number should be incremented with each
    change of the resource adapter properties.
  </description>
  <config-property-name>revisionNumber
  </config-property-name>
  <config-property-type>java.lang.String
  </config-property-type>
  <config-property-value>2
  </config-property-value>
</config-property>
```

4.2.5 Example of an weblogic-ra.xml file

Example 4 shows the definition of the general configuration properties in the fileweblogic-ra.xml.

Example 4 Configurations- properties in the weblogic-ra.xml file

The section containing the general configuration properties in the file weblogic-ra.xml has the following structure:

```
<weblogic-connector
   xmlns="http://xmlns.oracle.com/weblogic/weblogic-connector"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://xmlns.oracle.com/weblogic/weblogic-connector
   http://www.oracle.com/technology/weblogic/weblogic-connector/1.3/
   weblogic-connector.xsd">
   <jndi-name>BeanConnect</jndi-name>
   <enable-global-access-to-classes>true</enable-global-access-to-classes>             (1)
   <work-manager>
     <name>BeanConnect_WorkManager</name>
   </work-manager>
```

```
 <connector-work-manager>
     <!-- For inbound communication the BeanConnect resource adapter starts a long-running
          work instance, which listens at the inbound listener port.
          Each BeanConnect proxy work process establishs a
          connection to the BeanConnect resource adapter. For each
          connection the BeanConnect resource adapter starts a long-running
          work instance.
          The maximum number of concurrent long-running Work instances will
          be specified with the following property.

         Default value: 10
     -->
     <max-concurrent-long-running-requests>12
     </max-concurrent-long-running-requests>
   </connector-work-manager>
<security>
   <security-work-context>
       <!-- Two choices related to establishing the caller identity for a
            work instance are described in
            JSR 322: Java EE Connector Architecture 1.6
            The default value is false, which means Case 1.
            All caller-principal-mapping and group-principal-mapping
            subelements are ignored.
            If set to true, it means Case 2.
            All caller-principal-mapping and group-principal-mapping
            elements are used to determine the correct mapping from the
            EIS security domain to the WebLogic domain.
       -->
       <inbound-mapping-required>false</inbound-mapping-required>
     <caller-principal-default-mapped>
        <!-- Used for Inbound Message Endpoint invocation if inbound mapping
             required and a user is inflowed who has no explicit mapped
             caller principal  -->
        <!--
          <use-anonymous-identity>true</use-anonymous-identity>
        -->
        <principal-name>BcDefaultMappedPrincipal1</principal-name>
      </caller-principal-default-mapped>
      <caller-principal-mapping>
        <eis-caller-principal>EISUSER1</eis-caller-principal>
        <mapped-caller-principal>
          <!--
            <use-anonymous-identity>true</use-anonymous-identity>
          -->
          <principal-name>AsPrincipal1</principal-name>
        </mapped-caller-principal>
      </caller-principal-mapping>
      <caller-principal-mapping>
        <eis-caller-principal>EISUSER2</eis-caller-principal>
        <mapped-caller-principal>
          <principal-name>AsPrincipal2</principal-name>
        </mapped-caller-principal>
      </caller-principal-mapping>
      <caller-principal-mapping>
        <eis-caller-principal>EISUSER3</eis-caller-principal>
        <mapped-caller-principal>
          <principal-name>AsPrincipal3</principal-name>
        </mapped-caller-principal>
      </caller-principal-mapping>
      <caller-principal-mapping>
        <eis-caller-principal>EISUSER4</eis-caller-principal>
        <mapped-caller-principal>
          <principal-name>AsPrincipal4</principal-name>
        </mapped-caller-principal>
      </caller-principal-mapping>
   </security-work-context>
</security>
</weblogic-connector>
```

(1) This property is absolutely essential for correct operation!

4.3 Setting configuration properties for outbound communication via OSI-TP / LU6.2

The following configuration properties must be set for outbound communication:

- The general configuration properties in the file `ra.xml` (see Section 4.2.1) uand the file `weblogic-ra.xml` (see Section 4.2.2).
- The connecttion-specific properties for OSI TP / LU6.2 in the file `weblogic-ra.xml` (see Section 4.2.2).

Connection factories

Outbound communication is performed via so-called connection factories.

You must configure at least one connection factory for each EIS partner. However, you may also configure multiple connection factories with different properties for one and the same EIS partner.

You make the following specifications for each connection factory:

- JNDI name: This name is also required when deploying the EJB and makes it possible to address the connection factory in the EJB.
- Configuration properties: These are the parameters that apply to the connection to the EIS partners.
- Connection pooling: Connection pooling is an application server functionality. It helps improve performance on frequently used connections.
- Security settings: This is the data required when signing on at the EIS partner, e.g. user ID and password. If this data is encoded directly in the EJB then the corresponding entry can be omitted in the deployment descriptor.
- In the Management Console, an outbound communication endpoint is assigned to every connection factory.

4.3.1 Defining general and connection-specific properties for OSI-TP / LU6.2 in weblogic-ra.xml

The file `weblogic-ra.xml` is the WebLogic-specific deployment descriptor for the resource adapter. This deployment descriptor specifies the settings that are required for outbound communication with EIS partners. The communications are conducted via so-called connection factories.

General settings for outbound communication

- The settings for outbound communication are defined in the element `<outbound-resource-adapter>`.
- The properties for connection factories can be deterùmined using the sub-elements`<default-connection-properties>` and `<connection-definition-group>`.
- For each `<connection-definition>` that is present in the `ra.xml`, a `<connection-definition-group>` must be defined in the `weblogic-ra.xml`.
- Default settings that are to apply to all connection factories can be defined in the sub-element `<default-connection-properties>` of the element `<outbound-resource-adapter>`.
- Default settings that are to apply to the connection factories for a `<connection-definition-group>` can be defined in the sub-element `<default-connection-properties>` of the element `<connection-definition-group>`.
- For each connection factory that is to be used by the EJB, a `<connection-instance>` must be defined in the corresponding `<connection-definition-group>`.

4.3.1.1 Defining the resource for OSI-TP / LU6.2

The resource type of a connection factory is defined in the tag `<connection-factory-interface>`.

Communication via OSI-TP / LU6.2 involves the following resource types:

- `net.fsc.jca.communication.cci.BCOltpConnectionFactory`
- `net.fsc.jca.communication.EISOltpConnectionFactory`

Precisely this type must be specified for the EJB which uses this connection factory in the deployment descriptor `ejb-jar.xml`, see Section 4.3.2, "Deploying an Enterprise JavaBean for OSI-TP / LU6.2".

You will find an example of the file `weblogic-ra.xml` in Section 4.3.1.6, "Example: weblogic-ra.xml".

4.3.1.2 Defining the JNDI name for OSI TP / LU6.2

The connection factory's JNDI name is defined in the `<connector-factory>` tag by means of the `location` attribute.

An Enterprise Java Bean (EJB) in the application server addresses a connection factory via JNDI. The connection factory assigns the configured properties to the connection.

You will find an example in Section 4.3.1.6, "Example: weblogic-ra.xml".

4.3.1.3 Defining configuration properties for OSI-TP / LU6.2

For outbound communication, BeanConnect supports the following connection-specific configuration properties:

- `bufferedIO`
- `connectionURL`
- `displayName`
- `encoding`
- `encodingActive`
- `logLevel`
- `timeout`
- `transactional`

The connection which you obtain with a `ConnectionFactory.getConnection()` call is preinitialized with these configuration properties.

bufferedIO

The `bufferedIO` configuration property is used to define whether I/O buffering is carried out between the resource adapter and the proxy. If set, interaction between resource adapter and proxy is minimized. To reach maximum performance within a production environment you should set this property to `true`. During deployment or operation, you can set this value to `false` by means of the connection factory MBean to detect user errors as early as possible during the test.

Definition: `[true | false]`

Explanation: Switches IO buffering between resource adapter and proxy on or off.

`true`: IO buffering is used.

`false`: IO buffering is not used.

Default: `true`

Example:

```
<property>
   <name>bufferedIO</name>
   <value>true</value>
</property>
```

connectionURL

For communication via OSI-TP or LU6.2, the `connectionURL` configuration property specifies the name of the outbound communication endpoint which stands for a connection to an EIS partner and was defined in the proxy using the Management Console. The name begins with a prefix which describes the type of EIS partner, e.g. `utm` or `cics`. The name used in BeanConnect V2.0 and earlier - `oltp` - continues to be supported for reasons of compatibility and is used as a synonym for `utm`.

The name of the outbound communication endpoints is a freely definable string. It must be the same as the corresponding name of the outbound communication endpoint configured in the proxy, see Section 6.7, "Configuring outbound communication".

Definition: `<type>://<name>`

Explanation: `<type>` Type of the EIS partner, possible values:

`utm`	The EIS partner is of type openUTM.
`cics`	The EIS partner is of type CICS.
`xatmi-rr`	The EIS partner is of type XATMI and communication is conducted using the Request/Reply paradigm.
`xatmi-cv`	The EIS partner is of type XATMI and communication is conducted using the Conversational paradigm.

`<name>` Name of an outbound communication endpoint as it was defined using the Management Console

Default: `utm://outboundCommunicationEndpoint`

Example:

```
<property>
   <name>connectionURL</name>
   <value>utm://HELLO</value>
</property>
```

displayName

This attribute allows you to define a name for a managed connection factory. This name is then used by BeanConnect when outputting information about this managed connection factory, e.g. during the output of MBeans and LogWriter records.

Definition: `[<name>]`

Explanation: Freely definable name of a managed connection factory as it is to be used, for example in MBean and LogWriter output.

Default: No default value.

If you do not specify a name then the internal name of the managed connection factory is used. This consists of the prefix "MCF" and a 5-digit number.

Example:

```
<property>
   <name>displayName</name>
   <value>sample application/test</value>
</property>
```

encoding

The `encoding` configuration property defines a code table for converting byte code (for example EBCDIC) to Unicode. These code tables are used for converting byte streams to strings and vice versa. These conversions are always called implicitly when interactions (`sndString()`, `rcvString()` for example) are executed which contain strings as I/O parameters.

The code table that is defined with the `encoding` configuration property is used as default for the corresponding connection. The bean programmer may determine that a different code table is to be used for the connection by explicitly calling the `setEncoding(Encoding)` method of the `EISConnection` interface or of the `OltpMessageContext` interface (see Section 11.1, "Encoding").

Code conversion using this code table is only carried out if `encodingActive` is actually activated. You can select this with the `encodingActive` configuration property or by calling the method `setEncodingActive(true)`.

There are different definitions for openUTM-P partners and CICS partners.

For UTM partners apply:

Definition:

```
[<builtin_encoding_table> |
builtin:<builtin_encoding_table> |
jdk:<jdk_encoding_table> |
custom:<encoding_table>]
```

Explanation: Name of a code table to be used for code conversion.

If no prefix is specified or if the prefix `builtin:` is specified, you must specify the name of a built-in code table provided by BeanConnect.

The following built-in code tables are provided:

`OSD_EBCDIC_DF03_IRV`, `OSD_EBCDIC_DF04_1`, `OSD_EBCDIC_DF04_15`, `OSD_EBCDIC_DF04_DRV`

Use the prefix `jdk:` to specify a code table contained in the JDK.

Use the prefix `custom:` to assign your own code table. Here you must specify the fully qualified class name of the code table. For further details on using your own code tables, refer to the JavaDoc for BeanConnect.

Default: For openUTM partners, this value is set to "`OSD_EBCDIC_DF04_DRV`".

Example:

```
<property>
   <name>encoding</name>
   <value>OSD_EBCDIC_DF03_IRV</value>
</property>
```

For CICS partners apply:

Definition: `[jdk:<jdk_encoding_table> | custom:<encoding_table>]`

Explanation: Name of a code table to be used for code conversion.

Use the prefix `jdk:` to specify a code table contained in the JDK.

Use the prefix `custom:` to assign your own code table. Here you must specify the fully qualified class name of the code table. For further details on using your own code tables, refer to the JavaDoc for BeanConnect.

Default: `jdk:Cp1047`

Example:

```
<property>
   <name>encoding</name>
   <value>jdk:Cp1250</value>
</property>
```

encodingActive

The `encodingActive` configuration property specifies whether code conversion is to be activated.

Definition: `[true | false]`

Explanation: Flag specifying whether code conversion is to be activated.

`true`: Code conversion according to the settings of the `encoding` configuration property is activated.

`false`: The default code table of the JDK is used to convert byte streams to strings.

Default: `false`

Example:

```
<property>
   <name>encodingActive</name>
   <value>true</value>
</property>
```

The deployment settings can be overwritten using the `setEncodingActive()` method defined in the `EISConnection` interface.

logLevel

This attribute can be used to set the level for the output of log records to a LogWriter for a connection factory. LogWriters for connection factories are configured in different ways depending on the employed application server. For more detailed information on LogWriter configuration and output, see Section 13.4, "LogWriter for connection factories".

Definition: `[NONE | ERROR | INFO | ALL]`

Explanation:

`NONE` No output is written to the LogWriter.

`ERROR` Only information relating to exceptions and transaction rollbacks is written to the LogWriter.

`INFO` In addition to the information listed for `ERROR`, all transaction-related events are logged, e.g. the beginning or committing of transactions.

`ALL` In addition to the information listed for `INFO`, all events relating to connection lifecycles are logged. These include, for example, the requesting or releasing of connection handles by the application or events which affect pooling.

Default: `NONE`

Example:

```
<property>
   <name>logLevel</name>
   <value>INFO</value>
</property>
```

timeout

The `timeout` configuration property specifies the maximum time the resource adapter waits for the proxy to answer.

The value specified here must be greater than the maximum time that the EIS system needs to process a call. If the timer expires, an exception is thrown to the application and the connection between the resource adapter and the proxy is reinitialized. This generally causes the transaction to be reset at the EIS partner

Definition: Time in millisecods.

Explanation:

:> 0 Maximum time in milliseconds the resource adapter waits.

:0 The resource adapter waits indefinitely.

Default: 300000 (corresponds to 5 minutes)

Example:

```
<property>
   <name>timeout</name>
   <value>30000</value>
</property>
```

transactional

The `transactional` configuration property specifies whether the communication between the application server and the EIS should be transactional. In this case the transaction of the EIS is included in the transaction of the application server.

Definition: [true | false]

Explanation: `true`: Participation in the application server transaction is activated.

`false`: Participation in the application server transaction is deactivated.

Default: false

Example:

```
<property>
   <name>transactional</name>
   <value>true</value>
</property>
```

4.3.1.4 Adapting connection pooling for OSI-TP / LU6.2

For each connection factory in the file `weblogic-ra.xml`, you can specify how connection pooling is carried out.

Connection pooling is activated in order to increase the performance. Connections which are used frequently and by many clients should be defined with large values for `max-capacity`. For rarely used connections you need not define connection pooling at all. For more information on connection pooling, refer to the application server documentation.

A connection factory definition with sample settings is shown in Example 5:

Example 5 Connection pooling

```
<connection-instance>
   <jndi-name>eis/my_EIS</jndi-name>
   <connection-properties>
      <pool-params>
         <initial-capacity>2</initial-capacity>
         <max-capacity>10</max-capacity>
      </pool-params>
       ...
   </connection-properties>
</connection-instance>
```

Refer to the schema description for the file `weblogic-ra.xml` for further pool parameters.

4.3.1.5 Defining security settings (managing sign-on)

If an EJB requests a connection to the EIS with a `ConnectionFactory.getConnection()` call, this connection is set up in the security context of BeanConnect. In particular, the authentication data (user name and password) required for the EJB to access the EIS is assigned when the connection is set up.

EJBs can authenticate themselves to the EIS in two ways:

- Application-managed authentication
- Container-managed authentication

It is recommended that container-managed authentication is used.

The basic procedure for application- and container-managed authentication is explained below.

Application-managed authentication

In this case, the authentication data must be provided in the program code of the EJB (see Chapter 10, "Interfaces and programming"). For EJBs which perform authentication themselves, the `<res-auth>` tag of the associated EJB deployment descriptor must be specified as follows:

```
<res-auth>Application</res-auth>
```

Example of setting by EJB:

```
getConnection(new PasswordCredential(user, password));
```

Container-managed authentication

In this case, the application server regulates the transfer of authentication data. For EJBs which allow the application server to perform authentication, the `<res-auth>` tag of the associated EJB deployment descriptor must be specified as follows:

```
<res-auth>Container</res-auth>
```

The configuration for container-managed authentication is specific to the different type of application server.

The following applies to Oracle WebLogic Server:

- The container-managed sign-on procedure in Oracle WebLogic Server is based on **outbound credential mapping** during which the WebLogic credentials (normally the user name and password) are mapped to the user name and password of the EIS partner.
- In general terms, it is important to note that Oracle WebLogic Server only supports outbound credential mapping for the default security realm (normally "myrealm").
- In this case, a WebLogic user name can be mapped to the user name of the EIS system either specifically for an individual managed connection factory or for the entire resource adapter.
- Oracle WebLogic Server also makes it possible to define a default mapping for user names for which no explicit mapping has been specified and to define a user name for non-authenticated users (anonymous mapping).

During outbound credential mapping, Oracle WebLogic Server checks the following items in the specified order:

1. Has a mapping been defined for the current managed connection factory for the user name or, in the case of a non-authenticated user, has an anonymous mapping been defined?
2. Has a mapping been defined for the resource adapter for the user name or, in the case of a non-authenticated user, has an anonymous mapping been defined?

3. Has a default mapping been defined for the current managed connection factory?
4. Has a default mapping been defined for the resource adapter?

The user name is mapped as defined for the first condition that is satisfied.

If none of the conditions is satisfied, Oracle WebLogic Server does not pass any authentication data to the resource adapter. In this case, BeanConnect performs application-managed authentication.

Under Oracle WebLogic Server, you use the WebLogic Server Administration Console to perform outbound credential mapping as follows:

1. On the left side of the WebLogic Server Administration Console output screen, choose the option **Deployments**.
2. In the **Deployments**page click the name of the resource adapter.
3. In the resource adapter's **Settings** page, choose the **Security** and **Outbound Credential Mapping** tabs one after the other.
4. Perform the required mappings.

For details on outbound credential mapping in Oracle WebLogic Server , refer to the section "Outbound Credential Mappings" in the application server documentation.

4.3.1.6 Example: weblogic-ra.xml

Example 6 shows the definition of the connection-specific configuration properties in `weblogic-ra.xml`.

Example 6 Configuration properties in the file weblogic-ra.xml

The section with the connection-specific configuration properties in the file `weblogic-ra.xml` has the following layout:

```
<weblogic-connector
   xmlns="http://xmlns.oracle.com/weblogic/weblogic-connector"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.oracle.com/weblogic/weblogic-connector
http://www.oracle.com/technology/weblogic/weblogic-connector/1.3/weblogic-connector.xsd">
…
   <outbound-resource-adapter>
     <default-connection-properties>
       <pool-params>
         <initial-capacity>0</initial-capacity>
         <max-capacity>10</max-capacity>
         </pool-params>
         <transaction-support>XATransaction</transaction-support>
      </default-connection-properties>
```

```
<connection-definition-group>
   <connection-factory-interface>net.fsc.jca.communication.EISOltpConnectionFactory
    </connection-factory-interface>
  <default-connection-properties>
     <pool-params>
        <initial-capacity> [1]
        </initial-capacity>
        <max-capacity>20</max-capacity>
     </pool-params>
  </default-connection-properties>
 <connection-instance>
   <jndi-name>eis/my_BC_Factory1</jndi-name>
   <connection-properties>
     <pool-params>
       <initial-capacity>2</initial-capacity>
    </pool-params>
    <properties>
      <property>
         <name>ConnectionURL</name>
         <value>utm://accountEIS</value> [1]
       </property>
      <property>
         <name>encoding</name>
         <value>OSD_EBCDIC_DF04_DRV</value> [2]
         </property>
        <property>
          <name>encodingActive</name>
            <value>true</value>
        </property>
       <property>
         <name>transactional</name>
         <value>true</value>
       </property>
       <property>
         <name>timeout</name>
         <value>30000</value>
       </property>
       <property>
         <name>bufferedIO</name>
         <value>true</value>
       </property>
       <property>
         <name>logLevel</name>
         <value>ALL</value>
       </property>
```

[1] Für CICS partners: <value>cics://accountEIS</value>

[2] Für CICS partners: <value>jdk:Cp1047</value>

```
        <property>
          <name>displayName</name>
          <value>accountEIS</value>
        </property>
      </properties>
    </connection-properties>
  </connection-instance>
</connection-definition-group>
<connection-definition-group>
  <connection-factory-interface>
   net.fsc.jca.communication.cci.BCOltpConnectionFactory
  </connection-factory-interface>
  <connection-instance>
    <jndi-name>eis/my_CCI_factory</jndi-name>
    <connection-properties>
      <pool-params>
        <initial-capacity>3</initial-capacity>
      </pool-params>
      <properties>
        <property>
          <name>ConnectionURL</name>
          <value>utm://accountEIS"</value> 1
        </property>
        <property>
          <name>encoding</name>
          <value>OSD_EBCDIC_DF04_DRV</value> 2
        </property>
        <property>
          <name>encodingActive</name>
          <value>true</value>
        </property>
       <property>
          <name>transactional</name>
          <value>true</value>
        </property>
        <property>
          <name>timeout</name>
          <value>30000</value>
        </property>
        <property>
          <name>bufferedIO</name>
          <value>true</value>
        </property>
        <property>
          <name>logLevel</name>
          <value>NONE</value>
        </property>
        <property>
```

```
              <name>displayName</name>
              <value>cci/accountEIS</value>
            </property>
          </properties>
       </connection-properties>
      </connection-instance>
    </connection-definition-group>
  </outbound-resource-adapter>
</weblogic-connector>
```

4.3.2 Deploying an Enterprise JavaBean for OSI-TP / LU6.2

When deploying an EJB which is designed to use BeanConnect for outbound communication, you must link the EJB to the BeanConnect deployment. The following files are relevant for deploying an EJB:

- Code file of the EJB (`.java` or `.class` file)
- Standardized deployment descriptor of the EJB (`ejb-jar.xml`) or Java annotations
- Application server-specific deployment descriptor of the EJB
  (with Oracle WebLogic Server: `weblogic-ejb-jar.xml`)
- Application server-specific deployment descriptor for the resource adapter
  (with Oracle WebLogic Server: `weblogic-ra.xml`)

When an EJB is deployed, the resource reference used by the Bean developer is made known to the application server in the deployment descriptor of the EJB. In addition, a resource type is assigned to the resource reference.

BeanConnect supports the following resource types, which represent the different types of connections that can be used:

- For OSI-TP or LU6.2 communication using the BeanConnect interface:

  `net.fsc.jca.communication.EISOltpConnectionFactory`

- For OSI-TP or LU6.2 communication using the CCI interface:

  `net.fsc.jca.communication.cci.BCOltpConnectionFactory`

You must specify the resource type in the following files:

- `ejb-ra.xml` with the `<res-type>` tag
- `weblogic-jar.xml` with the `<connectionfactory-interface>` tag

The sections of the code file of the EJB as well as of the files `ejb-jar.xml`, `weblogic-ejb-jar.xml` and `weblogic-ra.xml` that are relevant for the deployment of the EJB are described in detail below. The **bold** (partial) path names indicate the relationships between the individual files.

- Code file of the EJB (`.java` or `.class` file)

  The JNDI lookup for the `ConnectionFactory` object via a resource reference (coded name) takes place here. In the following example, the resource reference used is `eis/Part1Dial`.

  ```
  ...
  
  cf=(EISConnectionFactory)
           ic.lookup("java:comp/env/eis/Part1Dial")
  
  ...
  ```

- Deployment descriptor of the EJB (`ejb-jar.xml`)

  Here the resource reference (`ConnectionFactory` object) which the EJB accesses is specified. In addition, a resource type is assigned to the resource reference. In the following example `net.fsc.jca.communication.EISOltpConnectionFactory` is used as the resource type.

  ```
  <session>
    <ejb-name>SimpleBeanConnect</ejb-name>
    ...
    <resource-ref>
      <res-ref-name>eis/Part1Dial</res-ref-name>
      <res-type>
        net.fsc.jca.communication.EISOltpConnectionFactory
      </res-type>
      <res-sharing-scope>Unshareable</res-sharing-scope>
     ...
    </resource-ref>
  </session>
  ```

i Please note that for `<res-sharing-scope>` you must always specify `Unshareable`.

- Application server-specific deployment descriptor of the EJB (with Oracle WebLogic Server: `weblogic-ejb-jar.xml`):

  Here, application server JNDI names are assigned to the EJB name and the resource references that are defined in the file `ejb-jar.xml`.

  ```
  <weblogic-enterprise-bean>
        <ejb-name>SimpleBeanConnect</ejb-name>
        <jndi-name>ejb/SimpleBeanConnect</jndi-name>
       <resource-description>
           <res-ref-name>comp/env/eis/Part1Dial</res-ref-name>
  ```

```
         <jndi-name>java:comp/env/eis/Part1Dial</jndi-name>
      </resource-description>
   </weblogic-enterprise-bean>
```

- Deployment descriptor for the resource adapter (with Oracle WebLogic Server: `weblogic-ejb-jar.xml`):

  Here the connection factory is configured for deployment in the application server instance is configured and linked to the resource reference via the JNDI name (here: `partner1Dial`). Via the configuration of the connection factory in the application server instance, the resource adapter receives the URL of the service (in the following code fragment: `utm://echo` and of the EIS partner.

  ```
  <outbound-resource-adapter>
     <connection-definition-group>
       <connection-factory-interface>
          net.fsc.jca.communication.EISOltpConnectionFactory
       </connection-factory-interface>
       <connection-instance>
         <jndi-name>eis/partner1Dial</jndi-name>
         <connection-properties>
         <properties>
          <property>
            <name>ConnectionURL</name>
            <value>utm://echo</value> 3
          </property>
          ...
         </connection-properties>
       </connection-factory-interface>
      </connection-definition-group>
  </outbound-resource-adapter>
  ```

  The value specified in `<connectionfactory-interface>` must be identical to the value that is specified with the `<res-type>` tag in the file `ejb-jar.xml`.

  An additional configuration step is required on the proxy. For each outbound communication endpoint name that is specified in a `connectionURL` configuration property in the file `weblogic-ra.xml`, you must configure a corresponding outbound communication endpoint of the same name in the proxy. The outbound communication endpoint definition maps the symbolic service name onto a real service name in the EIS partner application. You can carry out configuration of an outbound communication endpoint using the Management Console (see Section 6.7.2, "Configuring outbound communication endpoints").

[3] Für CICS partners: <value>cics://echo</value>

4.4 Configuring outbound communication via UPIC

When communications are performed via UPIC, you simply need to define the connection-specific properties in the file `weblogic-ra.xml`. In this case, the file `ra.xml` is not relevant.

Connection factories

Outbound communication is performed via so-called connection factories.

You must configure at least one connection factory for each EIS partner. However, you may also configure multiple connection factories with different properties for one and the same EIS partner.

You make the following specifications for each connection factory:

- JNDI name: This name is also required when deploying the EJB and makes it possible to address the connection factory in the EJB.
- Configuration properties: These are the parameters that apply to the connection to the EIS partners.
- Connection pooling: Connection pooling is an application server functionality. It helps improve performance on frequently used connections.
- Security settings: This is the data required when signing on at the EIS partner, e.g. user ID and password. If this data is encoded directly in the EJB then the corresponding entry can be omitted in the deployment descriptor.

4.4.1 Defining general connection-specific properties for UPIC in weblogic-ra.xml

The file `weblogic-ra.xml` is the WebLogic-specific deployment descriptor for the resource adapter. This deployment descriptor specifies the settings that are required for outbound communication with EIS partners. The communications are conducted via so-called connection factories.

General settings for outbound communication

The settings for outbound communication are defined in the element `<outbound-resource-adapter>`. Here:

- You specify the properties for connection factories in the subelements `<default-connection-properties>` and `<connection-definition-group>`.
- You specify default settings that are to apply for all the connection factories in the subelement `<default-connection-properties>` of the element `<outbound-resource-adapter>`.
- You specify default settings that are to apply for all the connection factories of a `<connection-definition-group>` in the subelement `<default-connection-properties>` of the element `<connection-definition-group>`.
- For each connection factory that is to be used by the EJBs, a `<connection-instance>` must be described in the corresponding `<connection-definition-group>`.

4.4.1.1 Defining a resource for UPIC

The connection factory's resource type is defined in the tag `<connection-factory-interface>` of the subelement`<connection-definition-group>`.

The following resource types are relevant for communication via UPIC:

- `net.fsc.jca.communication.cci.BCUpicConnectionFactory`
- `net.fsc.jca.communication.EISUpicConnectionFactory`

For the EJB that uses this connection factory, it is necessary to specify precisely this type in the deployment descriptors `ejb-jar.xml` and `weblogic-ra.xml`, see Section 4.4.2, "Deploying an Enterprise JavaBean for UPIC". You will find an example in Section 4.3.1.6, "Example: weblogic-ra.xml".

4.4.1.2 Defining the JNDI name for UPIC

The connection factory's JNDI name is defined in the `<connection-instance>` subelement of a `<connection-definition-group>` by means of the tag `<jndi-name>`.

An Enterprise Java Bean (EJB) in the application server addresses a connection factory via JNDI (Java Naming and Directory Interface). The connection factory assigns the configured properties to the connection.

4.4.1.3 Setting the configuration properties for UPIC

The configuration properties of a connection factory are defined via the subelement `<connection-properties>` in the element `<connection-instance>`. You can also make these changes using the WebLogic Server Administration Console GUI.

BeanConnect supports the following connection-specific configuration properties for outbound communication:

- `connectionURL`
- `displayName`
- `encoding`
- `encodingActive`
- `logLevel`
- `timeout`
- `reconnectThreshold`

The connection provided by a `ConnectionFactory.getConnection()` call is preinitialized with these configuration properties.

connectionURL

The `connectionURL` property defines the EIS partner and, if required, the service that is to be addressed. Only EIS partners of type openUTM are permitted for outbound communication via UPIC.

In the case of a connection with a UTM cluster application, you can specify a list of URLs under `connectionURL`. The individual URLs in this list must be separated by commas.

The `getConnection()` method of a connection factory supplies a connection with the appropriate configuration properties. The `getConnection()` property can be used by both the `EISConnection` or the `EISUpicConnection` interface.

Definition:

```
upic://<host>[:<port>]/[<local>:]<remote>[/<tac>]
[?tsel[;tsel]]
```

Explanation:

`host` Host on which the openUTM partner application is running.

`port` Port number of the port at which the openUTM partner application listens (optional). Default: 102

`local` Local name of the client (PTERM). Default: `JUPIC`

`remote` Name of the openUTM partner application (BCAMAPPL or APPLINAME).

`tac` TAC (ServiceName) which is to be called in the openUTM partner application (optional).

This value can be overwritten by methods which are defined in the `EISConnection` interface.

`tsel` TSEL format definition for the locale address (lt) or remote addresses (rt) in the form (optional):

`[lt={a|e|t}]|[rt={a|e|t}]`

a =ASCII, e = EBCDIC, t = TRANSDATA

Default:

`lt=t` if `local` does not contain any lowercase characters, `a` otherwise

`rt=t` if `remote` does not contain any lowercase characters `a` otherwise

Default: `upic://<host>:<port>/application/service`

Example: EIS partner in BS2000:

```
<property>
    <name>connectionURL</name>
    <value>upic://BS2HOST/UTMAPP/INFO</value>
</property>
```

EIS partner on Solaris/Linux/Windows systems:

```
<property>
    <name>connectionURL</name>
    <value>upic://unixhost:24000/UTMAPP/INFO?rt=a</value>
</property>
```

displayName

This attribute allows you to define a name for a managed connection factory. This name is then used by BeanConnect when outputting information about this managed connection factory, e.g. during the output of MBeans and LogWriter records.

Definition: `[<name>]`

Explanation: Freely definable name of a managed connection factory as it is to be used, for example, in MBean and LogWriter output.

Default: No default value.

If you do not enter a name then the internal name of the managed connection factory is used. This consists of the prefix "MCF" and a 5-digit number.

Example:

```
<property>
    <name>displayName</name>
    <value>sample application/test</value>
</property>
```

encoding

The `encoding` configuration property defines a code table for converting byte code (for example EBCDIC) to Unicode. These code tables are used for converting byte streams to strings and vice versa. These conversions are always called implicitly when interactions (`sndString()`, `rcvString()` for example) are executed which contain strings as I/O parameters.

The code table that is defined with the `encoding` configuration property is used as default for the corresponding connection. The Bean programmer may determine that a different code table is to be used for the connection by explicitly calling the `setEncoding(Encoding)` method of the `EISConnection` interface (see Section 11.1, "Encoding").

Code conversion using this code table is only carried out if `encodingActive` is actually activated. You can select this with the `encodingActive` configuration property or by calling the method `setEncodingActive(true)`.

Definition:

```
[<builtin_encoding_table> |
builtin:<builtin_encoding_table>|
jdk:<jdk_encoding_table>|
custom:<encoding_table>]
```

Explanation: Name of a code table to be used for code conversion.

If no prefix is specified or if the prefix `builtin:` is specified, you must specify the name of a built-in code table provided by BeanConnect.

The following built-in code tables are provided:

`OSD_EBCDIC_DF03_IRV`, `OSD_EBCDIC_DF04_1`, `OSD_EBCDIC_DF04_15`, `OSD_EBCDIC_DF04_DRV`

Use the prefix `jdk:` to specify a code table contained in the JDK.

Use the prefix `custom:` to assign your own code table. Here you must specify the fully qualified class name of the code table. For further details on using your own code tables, refer to the JavaDoc for BeanConnect

Default: `OSD_EBCDIC_DF04_DRV`

Example:

```
<property>
    <name>encoding</name>
    <value>OSD_EBCDIC_DF03_IRV</value>
</property>
```

encodingActive

The `encodingActive` configuration property specifies whether code conversion is to be activated. The `encodingActive` configuration property is mapped directly onto the `setEncodingActive(boolean activate)` method of the `EISConnection` interface.

Definition: `[true | false]`

Explanation: Flag specifying whether code conversion is to be activated.

`true` Code conversion according to the settings of the `encoding` configuration property is activated.

`false` The default code table of the JDK is used to convert byte streams to strings.

Default: `false`

Example:

```
<property>
    <name>encodingActive</name>
    <value>true</value>
</property>
```

The deployment settings can be overwritten using the `setEncodingActive()` method defined in the `EISConnection` interface. It is also possible to activate customer-defined classes for code conversion (`setEncoding(Encoding)`) at this interface.

logLevel

This attribute can be used to set the level for the output of log records to a LogWriter for a connection factory. LogWriters for connection factories are configured in different ways depending on the employed application server. For more detailed information on LogWriter configuration and output, see Section 13.4, "LogWriter for connection factories".

Definition: `[NONE | ERROR | INFO | ALL]`

Explanation:

`NONE` No output is written to the LogWriter.

`ERROR` Only information relating to exceptions is written to the LogWriter.

`INFO` Same output as for `ERROR`.

`ALL` In addition to the information listed for `INFO/ERROR`, all events relating to the connection lifecycle are logged. These include, for example, the requesting and releasing of connection handles by the application or events relating to pooling.

Default: `NONE`

Example:

```
<property>
    <name>logLevel</name>
    <value>INFO</value>
</property>
```

timeout

The `timeout` configuration property defines the maximum wait time for `receive()` or `call()` calls. This property is mapped directly onto the socket timeout of the Java socket implementation.

The value specified here must be greater than the maximum time that the EIS system needs to process a call. If the timer expires, an exception is thrown to the application and the connection between the resource adapter and the proxy is reinitialized.

Definition: Timeout value (in milliseconds)

Default: 30000 (corresponds to 30 seconds)

Example:

```
<property>
    <name>timeout</name>
    <value>30000</value>
</property>
```

reconnectThreshold

The configuration property `reconnectThreshold` defines how often a (physical) connection can be used (number of `getConnection()` calls) before it is to be disconnected and reestablished by BeanConnect. This type of forced disconnection can be of use if the EIS partner is a cluster application and it is necessary to redistribute the connections to the cluster nodes from time to time.

The value 0 means that BeanConnect will not force any disconnections for this connection.

Definition: Upper limit for the number of times a physical connection can be used

Standard: 0 I.e. BeanConnect does not force disconnection

Beispiel:

```
<property>
    <name>reconnectThreshold</name>
    <value>10</value>
</property>
```

4.4.1.4 Adapting connection pooling for UPIC

Generally for each resource type or for each connection factory in the file `weblogic-ra.xml`, you can specify how connection pooling is carried out.

Connection pooling is used in order to increase the performance. `max-capacity` defines how many connections can be active for a connection factory at any one time; the default value is 10. Oracle WebLogic Server rejects connection requests that go beyond the number defined here with a `ResourceAllocationException`.

Connections which are used frequently and by many clients should be defined with large values for `max-capacity`. For rarely used connections you need not define connection pooling at all. For the other pool parameters, see the schema description for the file `weblogic-ra.xml`.
A connection factory definition with sample settings is listed in Example :

Connection pooling

```
<connection-instance>
      <jndi-name>eis/my_EIS</jndi-name>
      <connection-properties>
          <pool-params>
             <initial-capacity>2</initial-capacity>
             <max-capacity>10</max-capacity>
          </pool-params>
           …
     </connection-properties>
</connection-instance>
```

For the other pool parameters, see the schema description for the file `weblogic-ra.xml`.

4.4.1.5 Defining transaction support for UPIC

The value `XATransaction` is set as the transaction support level for the BeanConnect resource adapter in the file `weblogic-ra.xm`. Since communication via UPIC connections is not possible in a distributed transaction, it is advisable, when using Oracle WebLogic Server as application server, to overwrite this attribute for the connection factories in the file `weblogic-ra.xml`.

transaction-support

The transaction support level of a connection factory is described in the subelement `<transaction-support>` of a `<connection-instance>` in the file `weblogic-ra.xml`. For UPIC connections, you should specify the value `NoTransaction` here.

Beispiel: `<transaction-support>NoTransaction</transaction-support>`

4.4.1.6 Example: weblogic-ra.xml (UPIC)

Example 7 shows the definition of the configuration properties in the file `weblogic-ra.xml`.

Example 7 Configuration properties in the file weblogic-ra.xml

The section with the configuration properties in `weblogic-ra.xml` has the following layout:

```
<weblogic-connector
   xmlns="http://xmlns.oracle.com/weblogic/weblogic-connector"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://xmlns.oracle.com/weblogic/weblogic-connector
http://www.oracle.com/technology/weblogic/weblogic-connector/1.3/weblogic-connector.xsd">
   …
   <outbound-resource-adapter>
     <default-connection-properties>
       <pool-params>
         <initial-capacity>1</initial-capacity
        <max-capacity>15</max-capacity>
       </pool-params>
         <transaction-support>NoTransaction</transaction-support>
     </default-connection-properties>
     <connection-definition-group>
       <connection-factory-interface>
         net.fsc.jca.communication.EISUpicConnectionFactory
       </connection-factory-interface>
       <default-connection-properties>
         <pool-params>
           <initial-capacity>2</initial-capacity>
           <max-capacity>100</max-capacity>
```

```
    </pool-params>
  </default-connection-properties>
  <connection-instance>
    <jndi-name>eis/my_BC_factory</jndi-name>
    <connection-properties>
      <pool-params>
        <initial-capacity>5</initial-capacity>
      </pool-params>
      <properties>
        <property>
          <name>ConnectionURL</name>
          <value>value="upic://BS2HOST/UTMAPP/INFO"</value>
        </property>
        <property>
          <name>encoding</name>
          <value>OSD_EBCDIC_DF04_DRV</value>
        </property>
        <property>
          <name>encodingActive</name>
          <value>true</value>
        </property>
        <property>
          <name>timeout</name>
          <value>30000</value>
        </property>
        <property>
          <name>logLevel</name>
          <value>INFO</value>
        </property>
        <property>
          <name>displayName</name>
          <value>upic/UTMAPP/INFO</value>
        </property>
      </properties>
    </connection-properties>
  </connection-instance>
</connection-definition-group>
<connection-definition-group>
  <connection-factory-interface>
   net.fsc.jca.communication.cci.BCUpicConnectionFactory
  </connection-factory-interface>
  <default-connection-properties>
    <pool-params>
      <initial-capacity>0</initial-capacity>
    </pool-params>
  </default-connection-properties>
  <connection-instance>
    <jndi-name>eis/my_CCI_factory</jndi-name>
```

```
          <connection-properties>
            <pool-params>
             <max-capacity>100</max-capacity>
            </pool-params>
            <transaction-support>NoTransaction</transaction-support>
            <properties>
              <property>
                <name>ConnectionURL</name>
                <value>upic://BS2HOST/UTMAPP/INFO</value>
              </property>
              <property>
                <name>encoding</name>
                <value>OSD_EBCDIC_DF04_DRV</value>
              </property>
              <property>
                <name>encodingActive</name>
                <value>true</value>
              </property>
              <property>
                <name>timeout</name>
                <value>30000</value>
              </property>
              <property>
                <name>logLevel</name>
                <value>ERROR</value>
              </property>
              <property>
                <name>displayName</name>
                <value>upic/cci/UTMAPP/INFO</value>
              </property>
            </properties>
          </connection-properties>
        </connection-instance>
      </connection-definition-group>
    </outbound-resource-adapter>
  </weblogic-connector>
```

4.4.2 Deploying an Enterprise JavaBean for UPIC

When deploying an EJB which is designed to use BeanConnect for outbound communication, you must link the EJB to the BeanConnect deployment. The following files are relevant for deploying an EJB:

- Code file of the EJB (`.java` or `.class` file)
- Standardized deployment descriptor of the EJB (`ejb-jar.xml`)
- Application server-specific deployment descriptor of the EJB (with Oracle WebLogic Server: `weblogic-ejb-jar.xml`)
- Application server-specific deployment descriptor for the resource adapter (with Oracle WebLogic Server: `weblogic-ra.xml`)

When an EJB is deployed, the resource reference used by the Bean developer is made known to the application server in the deployment descriptor of the EJB. In addition, a resource type is assigned to the resource reference.

BeanConnect supports the following resource types, which represent the different types of connections that can be used:

- For UPIC communication using the BeanConnect interface:

  `net.fsc.jca.communication.EISUpicConnectionFactory`
- For UPIC communication using the CCI interface:

  `net.fsc.jca.communication.cci.BCUpicConnectionFactory`

You must specify the resource type in the following files:

- `weblogic-ra.xml` with the `<connection-factory-interface>` tag
- `ejb-jar.xml` with the `<res-type>` tag

The sections of the code file of the EJB as well as of the files `ejb-jar.xml`, `weblogic-ejb-jar.xml` and `weblogic-ra.xml` that are relevant for the deployment of the EJB are described in detail below. The **bold** (partial) path names indicate the relationships between the individual files.

- Code file of the EJB (`.java` or `.class` file)

  The JNDI lookup for the `ConnectionFactory` object via a resource reference (coded name) takes place here. In the following example, the resource reference used is `eis/Part1Dial`.

  ```
  ...
  
  cf=(EISConnectionFactory)
          ic.lookup("java:comp/env/eis/Part1Dial")
  
  ...
  ```

- Deployment descriptor of the EJB (`ejb-jar.xml`)

  Here the resource reference (`ConnectionFactory` object) which the EJB accesses is specified. In addition, a resource type is assigned to the resource reference. In the following example `net.fsc.jca.communication.EISUpicConnectionFactory` is used as the resource type.

  ```
  <session>
    <ejb-name>SimpleBeanConnect</ejb-name>
    ...
    <resource-ref>
      <res-ref-name>eis/Part1Dial</res-ref-name>
      <res-type>
        net.fsc.jca.communication.EISUpicConnectionFactory
      </res-type>
      <res-sharing-scope>Unshareable</res-sharing-scope>
     ...
    </resource-ref>
  </session>
  ```

i Please note that for `<res-sharing-scope>` you must always specify `Unshareable`.

- Application server-specific deployment descriptor of the EJB (with Oracle WebLogic Server: `weblogic-ejb-jar.xml`)

  Here, application server JNDI names are assigned to the EJB names and resource references defined in the file ejb-jar.xml.

  ```
  <weblogic-enterprise-bean>
     <ejb-name>SimpleBeanConnect</ejb-name>
     <jndi-name>ejb/SimpleBeanConnect</jndi-name>
     <resource-description>
        <res-ref-name>comp/env/eis/Part1Dial</res-ref-name>
        <jndi-name>java:comp/env/eis/partner1Dial</jndi-name>
     </resource-description>
  </weblogic-enterprise-bean>
  ```

- Deployment descriptor for the resource adapter (with Oracle WebLogic Server: `weblogic-ra.xml`):

  Here the connection factory for deployment in the application server (here: Oracle WebLogic Server) is configured and linked to the resource reference via the JNDI name (here: `partner1Dial`). The configuration of the connection factory in the application server informs the resource adapter of the URL of the service (in the following code fragment: `upic://BS2HOST/UTMAPP/INFO`) and the EIS partner.

  ```
  <outbound-resource-adapter>
     <connection-definition-group>
       <connection-factory-interface>
          net.fsc.jca.communication.EISUpicConnectionFactory
       </connection-factory-interface>
       <connection-instance>
         <jndi-name>eis/partner1Dial</jndi-name>
         <connection-properties>
         <properties>
          <property>
            <name>ConnectionURL</name>
            <value>upic://BS2HOST/UTMAPP/INFO</value>
          </property>
          ...
         </connection-properties>
       </connection-factory-interface>
      </connection-definition-group>
   </outbound-resource-adapter
  ```

  The value specified in `<connection-factory-interface>` must be identical to the value that is specified with the `<res-type>` tag in the file `ejb-jar.xml`.

4.5 Setting configuration properties for inbound communication

i The configuration of the general properties for ibound communication is described in Section 4.2.1, "Defining general properties in ra.xml" and Section 4.2.2, "Defining the general properties of the resource adapter in weblogic-ra.xml".

At least one OLTP message-driven bean has to be deployed when using inbound communication. The OLTP message-driven bean must implement one of the following message listener interfaces:

- `net.fsc.jca.communication.AsyncOltpMessageListener`
- `net.fsc.jca.communication.OltpMessageListener`
- `javax.resource.cci.MessageListener`

A deployment descriptor has to be created in the file `ejb-jar.xml` for deploying the OLTP message-driven bean. If the file `ejb-jar.xml` is not created automatically by an IDE (**I**ntegrated **D**evelopment **E**nviroment) during the bean development process, you must create this file manually. Additionally, in most cases, the deployment of an OLTP message-driven bean requires also an application server-specific deployment descriptor. In the case of Oracle WebLogic Server, the application server-specific deployment descriptor is stored in the file `weblogic-ejb-jar.xml`. Examples of an `ejb-jar.xml` and an `orion-ejb-jar.xml` file for an OLTP message-driven bean are shown in Example 8 on page 136 and in Example 9 on page 139.

4.5.1 Configuration properties in the ejb-jar.xml

The following properties have to be set in the deployment descriptor (`ejb-jar.xml`) of the OLTP message-driven bean:

- The `messaging-type` property specifies the message listener interface used by the OLTP message-driven bean.
- The `activation-config` properties refer to the specified message listener interface. Each of the properties are specified in a separate `activation-config-property` element within the `activation-config` element of the file `ejb-jar.xml`.

The following `activation-config` properties are available:

`encoding`

`encodingActive`

`messageEndpoint`

`redeliveryThreshold`

The properties for the file `ejb-jar.xml` are described in detail below.

messaging-type

The `messaging-type` property specifies the message listener interface used by the OLTP message-driven bean.

Definition: Message listener interface used by the OLTP message-driven bean

Default: –

Example:

```
<messaging-type>
  net.fsc.jca.communication.AsyncOltpMessageListener
</messaging-type>
```

messageEndpoint

The `messageEndpoint` activation-config property specifies the name of the message endpoint. Note that the message endpoint name specified here must be identical to the name of the inbound message endpoint specified when configuring the proxy using the Management Console (see "Configuring inbound message endpoints" on page 224).

Definition: Name of the message endpoint.

Default: –

Example:

```
<activation-config-property>
  <activation-config-property-name>messageEndpoint
  </activation-config-property-name>
  <activation-config-property-value>SampleAsynOltpMdb
  </activation-config-property-value>
</activation-config-property>
```

encoding

The `encoding` activation-config property defines a code table for converting EIS-specific byte code (for example EBCDIC) to Unicode.

The property specified here is overwritten if the message-driven bean is called by an inbound service to which a `Partner Encoding` was assigned in the Management Console during the configuration of the proxy.

Definition:

```
[<builtin_encoding_table>|
builtin:<builtin_encoding_table> |
jdk:<jdk_encoding_table> |
custom:<encoding_table>]
```

Explanation: Name of a code table to be used for code conversion.

If no prefix is specified or if the prefix `builtin:` is specified, you must specify the name of a built-in code table provided by BeanConnect.

The following code tables are provided:

`OSD_EBCDIC_DF03_IRV`, `OSD_EBCDIC_DF04_1`, `OSD_EBCDIC_DF04_15`, `OSD_EBCDIC_DF04_DRV`

Use the prefix `jdk:` to specify a code table contained in the JDK.

Use the prefix `custom:` to assign your own code table. Here you must specify the fully qualified class name of the code table. For further details on using your own code tables, refer to the JavaDoc for BeanConnect

Default: OSD_EBCDIC_DF04_DRV

Example: for openUTM partners:

```
<activation-config-property>
  <activation-config-property-name>encoding
  </activation-config-property-name>
  <activation-config-property-value>OSD_EBCDIC_DF04_15
  </activation-config-property-value>
</activation-config-property>
```

for CICS partners:

```
<activation-config-property>
  <activation-config-property-name>encoding
  </activation-config-property-name>
  <activation-config-property-value>jdk:Cp1047
  </activation-config-property-value>
</activation-config-property>
```

encodingActive

The encodingActive activation-config property specifies whether code conversion is to be activated.

encodingActive is set to true irrespective of the value specified here if the message-driven bean is called by an inbound service to which a Partner Encoding was assigned in the Management Console during the configuration of the proxy

Definition: [true | false]

Explanation: Flag specifying whether code conversion is to be activated.

true: Code conversion according to the settings of the encoding activation-config property is activated.

false: The default code table of the JDK is used to convert byte streams to strings.

Default: false

Example:

```
<activation-config-property>
   <activation-config-property-name>encodingActive
   </activation-config-property-name>
   <activation-config-property-value>true
   </activation-config-property-value>
</activation-config-property>
```

redeliveryThreshold

The redeliveryThreshold activation-config property defines the number of additional attempts to deliver the message if the transaction is rolled back. This property can only be set for asynchronous OLTP message-driven beans, i.e. for OLTP message-driven beans which implement the message listener interface net.fsc.jca.communication.AsyncOltpMessageListener. The message listener interface is specified in the messaging-type property.

The property only takes effect if the OLTP message-driven bean has been deployed with the transaction attribute Required. In this case, the onMessage method is called inside a transaction which has been started by the proxy (never by the EIS). If this transaction is reset, the message is delivered again, unless the generated threshold has been exceeded.

Definition: Number of additional redelivery attempts if an error occurs.

Minimum value = 0
Maximum value = 254

Default: 0, i.e. the message is not delivered again.

Example:

```
<activation-config-property>
  <activation-config-property-name>redeliveryThreshold
  </activation-config-property-name>
  <activation-config-property-value>1
  </activation-config-property-value>
</activation-config-property>
```

4.5.2 Defining configuration properties for inbound communication in weblogic-ejb-jar.xml

In the application server-specific deployment descriptor `weblogic-ejb-jar.xml`, the value of the tag `<resource-adapter-jndi-name>` in the element `<message-driven-descriptor>` must correspond to the name of the resource adapter that was specified in the element `<jndi-name>` in the file `weblogic-ra.xml`.

See also Example 9, "weblogic-ejb-jar.xml file".

4.5.3 Examples for ejb-jar.xml and weblogic-ejb-jar.xml

Example 8 ejb-jar.xml file

The following code extract shows a deployment descriptor `ejb-jar.xml` for a JAR file that describes three OLTP message-driven beans. The OLTP message-driven beans implement three different message listener interfaces.

```
<?xml version="1.0" encoding="UTF-8"?>
<ejb-jar id="ejb-jar_ID" metadata-complete="false" version="3.1"
           xmlns="http://java.sun.com/xml/ns/javaee"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
                               http://java.sun.com/xml/ns/javaee/ejb-jar_3_1.xsd">
  <description xml:lang="en">Code Samples for Inbound Communication</description>
  <display-name xml:lang="en">SampleMessageDrivenBeans</display-name>
  <enterprise-beans>
    <message-driven>
      <description xml:lang="en">
           Code Sample for Dialog Inbound Communication
      </description>
      <ejb-name>SampleDialogOltpMdbBean</ejb-name>
      <ejb-class>net.fsc.jca.beanconnect.oltpmdb.SampleDialogOltpMdbBean
                                                                </ejb-class>
      <messaging-type>net.fsc.jca.communication.OltpMessageListener
      </messaging-type>
      <transaction-type>Container</transaction-type>
      <activation-config>
    <activation-config-property>
      <activation-config-property-name>messageEndpoint
      </activation-config-property-name>
      <activation-config-property-value>SampleDialogOltpMdb
      </activation-config-property-value>
    </activation-config-property>
    <activation-config-property>
      <activation-config-property-name>encodingActive
      </activation-config-property-name>
      <activation-config-property-value>true
       </activation-config-property-value>
    </activation-config-property>
    <activation-config-property>
      <activation-config-property-name>encoding
      </activation-config-property-name>
      <activation-config-property-value>OSD_EBCDIC_DF04_15[4]
      </activation-config-property-value>
    </activation-config-property>
  </activation-config>
</message-driven>
<message-driven>
  <description xml:lang="en">
       Code Sample for Asynchronous Inbound Communication
  </description>
  <ejb-name>SampleAsynOltpMdbBean</ejb-name>
  <ejb-class>net.fsc.jca.beanconnect.oltpmdb.SampleAsynOltpMdbBean</ejb-class>
  <messaging-type>net.fsc.jca.communication.AsyncOltpMessageListener
  </messaging-type>
  <transaction-type>Container</transaction-type>
  <activation-config>
    <activation-config-property>
      <activation-config-property-name>messageEndpoint
       </activation-config-property-name>
      <activation-config-property-value>SampleAsynOltpMdb
       </activation-config-property-value>
    </activation-config-property>
    <activation-config-property>
     <activation-config-property-name>encodingActive
      </activation-config-property-name>
     <activation-config-property-value>true
      </activation-config-property-value>
    </activation-config-property>
    <activation-config-property>
    <activation-config-property>
      <activation-config-property-name>encoding
      </activation-config-property-name>
```

[4] For CICS partners: <activation-config-property-value>jdk:Cp1047

```
        <activation-config-property-value>OSD_EBCDIC_DF04_15[4]
        </activation-config-property-value>
      </activation-config-property>
       <activation-config-property-name>redeliveryThreshold
        </activation-config-property-name>
        <activation-config-property-value>1</activation-config-property-value>
      </activation-config-property>
    </activation-config>
  </message-driven>
  <message-driven>
    <description xml:lang="en">
        Code Sample for CCI Inbound Communication</description>
    <ejb-name>SampleCciOltpMdbBean</ejb-name>
    <ejb-class>net.fsc.jca.beanconnect.oltpmdb.SampleCciOltpMdbBean</ejb-class>
    <messaging-type>javax.resource.cci.MessageListener</messaging-type>
    <transaction-type>Bean</transaction-type>
    <activation-config>
      <activation-config-property>
        <activation-config-property-name>messageEndpoint
         </activation-config-property-name>
        <activation-config-property-value>SampleCciOltpMdbBean
        </activation-config-property-value>
      </activation-config-property>
    </activation-config>
  </message-driven>
</enterprise-beans>
<assembly-descriptor>
  <container-transaction>
    <method>
      <ejb-name>SampleDialogOltpMdbBean</ejb-name>
      <method-name>onMessage</method-name>
      <method-params>
        <method-param>net.fsc.jca.communication.OltpMessage</method-param>
      </method-params>
    </method>
    <trans-attribute>Required</trans-attribute>
  </container-transaction>
  <container-transaction>
    <method>
      <ejb-name>SampleAsynOltpMdbBean</ejb-name>
      <method-name>onMessage</method-name>
      <method-params>
        <method-param>net.fsc.jca.communication.OltpMessage</method-param>
      </method-params>
    </method>
    <trans-attribute>Required</trans-attribute>
  </container-transaction>
  <container-transaction>
```

```
          <method>
            <ejb-name>SampleCciOltpMdbBean</ejb-name>
            <method-name>onMessage</method-name>
            <method-params>
              <method-param>net.fsc.jca.communication.OltpMessage</method-param>
            </method-params>
          </method>
          <trans-attribute>Required</trans-attribute>
        </container-transaction>
      </assembly-descriptor>
    </ejb-jar>
```

Example 9   weblogic-ejb-jar.xml file

The fiurst code extract presents the section of the application server-specific deployment descriptor `weblogic-ra.xml` for the resource adapter. The name of the resource adapter is defined in this section. The following application server-specific deployment descriptor `weblogic-ejb-jar.xml` then refers to this name:

```
<?xml version="1.0" encoding="UTF-8" ?>
<weblogic-connector
   xmlns="http://xmlns.oracle.com/weblogic/weblogic-connector"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://xmlns.oracle.com/weblogic/weblogic-connector
http://www.oracle.com/technology/weblogic/weblogic-connector/1.3/weblogic-connector.xsd">
   <jndi-name>BeanConnect</jndi-name>
```

Next comes the code extract from the application server-specific deployment descriptor `weblogic-ejb-jar.xml` which refers to the EJB for the three OLTP message-driven beans from Example 8, "ejb-jar.xml file" on page 136.

```
<?xml version="1.0" encoding="UTF-8"?>
<weblogic-ejb-jar xmlns="http://xmlns.oracle.com/weblogic/weblogic-ejb-jar"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.oracle.com/weblogic/weblogic-ejb-jar
http://xmlns.oracle.com/weblogic/weblogic-ejb-jar/1.1/weblogic-ejb-jar.xsd">
    <weblogic-enterprise-bean>
     <ejb-name>SampleDialogOltpMdbBean</ejb-name>
     <message-driven-descriptor>
       <resource-adapter-jndi-name>BeanConnect</resource-adapter-jndi-name>
     </message-driven-descriptor>
   </weblogic-enterprise-bean>
   <weblogic-enterprise-bean>
     <ejb-name>SampleAsynOltpMdbBean</ejb-name>
     <message-driven-descriptor>
       <resource-adapter-jndi-name>BeanConnect</resource-adapter-jndi-name>
     </message-driven-descriptor>
   </weblogic-enterprise-bean>
  <weblogic-enterprise-bean>
    <ejb-name>SampleCciOltpMdbBean</ejb-name>
    <message-driven-descriptor>
      <resource-adapter-jndi-name>BeanConnect</resource-adapter-jndi-name>
    </message-driven-descriptor>
  </weblogic-enterprise-bean>
</weblogic-ejb-jar>
```

4.6 Preparing resource adapter logging

After installation, the log4j property files are located in the config subdirectory of the resource adapter's installation directory.

This directory contains the following files:

1. `BeanConnect.log4j.properties.xml`
2. `BeanConnect.log4j.properties_debug.xml`
3. `BeanConnect.log4j.properties_default.xml`
4. `BeanConnect.log4j.properties_error.xml`

These files contain settings for resource adapter logging. The content of the first and third files is identical.

The BeanConnect resource adapter reads the logging settings from the file `BeanConnect.log4j.properties.xml`.

Copy this file to the directory `<WebLogicServerDomainDirectory>/config` (Oracle WebLogic Server specific).

Normally, you should not change the default settings for logging. If necessary (e.g. for diagnosis), you can subsequently extend or reduce the scope of logging, see Section 13.5.1, "Overview of logging in the BeanConnect resource adapter".

4.7 Special characteristics of multiple resource adapter mode

In a multiple resource adapter configuration, several resource adapters work together with one proxy instance. A maximum of 32 resource adapter instances are possible for each proxy instance. A unique index is assigned to each resource adapter instance. This index must be entered in the file `ra.xml`, see below.

Multiple resource adapter mode is possible for both outbound communication via OSI-TP / LU6.2 and for inbound communication.

In multiple resource adapter mode, you must perform the following configuration steps in the application server for each resource adapter:

- In the file `ra.xml`, define the index of the resource adapter using the additional configuration property `resourceAdapterIndex`. This may also be possible via the Management Console as in standard operation providing that certain requirements are fulfilled, see "Adapting the ra.xml file" on page 87.

  If you assign the indices manually, you must make sure that each resource adapter is given a unique index. In the case of inbound communication, you require this unique index to configure the inbound message endpoints via the Management Console (see Section 6.8.1, "Configuring inbound message endpoints").

- Perform the other configuration steps at the application server in exactly the same way as in default resource adapter mode, see Section 4.1.2, "Configuration steps for outbound and inbound communication".

resourceAdapterIndex

`resourceAdapterIndex` defines the index of the associated resource adapter instance in a multiple resource adapter configuration.

This property is only of any significance in a multiple resource adapter configuration with several resource adapter instances and must not be specified at the same time as the `resourceAdapterAddresses` property (see Section 4.8, "Special characteristics in cluster operation").

Definition: `<index>`

Explanation: `<index>` is a number between 1 and 32. It is defined by the Management Console if the `ra.xml` is configured using the Management Console.

Default: No default value

Example:

```
<config-property>
  <description>Index of this resource adapter instance in a
    multi-resource-adapter configuration.
  </description>
  <config-property-name>resourceAdapterIndex
  </config-property-name>
  <config-property-type>java.lang.String
  </config-property-type>
  <config-property-value>5
  </config-property-value>
</config-property>
```

4.8 Special characteristics in cluster operation

Multiple resource adapter instances may run together with multiple proxy instances in a cluster environment. The number of proxy instances does not have to be the same as the number of resource adapter instances. A maximum of 32 resource adapter instances and a maximum of 32 proxy instances are possible.

All the participating instances have an identical configuration, and in particular each resource adapter instance is deployed with the same BeanConnect RAR archive and therefore works with the same configuration values from `ra.xml`.

Cluster operation is possible for both outbound communication via OSI-TP / LU6.2 and for inbound communication.

In the case of cluster operation, the following configuration steps are necessary at the application server:

- Define the additional parameters and properties for cluster operation in the `ra.xml` file:
  - You must specify the addresses of all the proxy instances in the property `proxyURL`.
  - You must specify the addresses of all the resource adapter instances in the additional property `resourceAdapterAddresses`.
  - You can modify the parameters for the reallocation of the resource adapters and proxies in the properties `proxyReconnectCount` and `proxyReconnectInterval`.

  This is also possible via the Management Console as in standard operation, see "Adapting the ra.xml file" on page 87. However, you may also adapt the `ra.xml` file manually.
- Perform the other configuration steps at the application server in exactly the same way as when using only one resource adapter, see Section 4.1.2, "Configuration steps for outbound and inbound communication".

proxyURL

In cluster operation, the `proxyURL` defines the assignment of the resource adapter instances to the proxy instances. If you are working with multiple proxy instances then you must specify the addresses of all the proxies, each separated by a semicolon.

Definition: `oltp://<host>:<port>/<name>;` ... `;oltp://<host>:<port>/<name>`

Explanation:

`<host>` Host on which the associated proxy container is installed.

`<host>` can be specified as a symbolic name or as an IPv4 address.

`<port>` Port number of the associated proxy container + 4

`<name>` Application name of the associated proxy container (BCU<port>)

The individual entries must be separated by semicolons.

Default: `oltp://localhost:31004/BCU31004`

Example:

```
<config-property>
  <description>BeanConnect Proxy URLs for OLTP outbound
         communication with 2 Proxies</description>
  <config-property-name>proxyURL</config-property-name>
  <config-property-type>java.lang.String
  </config-property-type>
  <config-property-value>oltp://proxyhost1:31004/BCU31004;
         oltp://proxyhost2:31004/BCU31004
  </config-property-value>
</config-property>
```

resourceAdapterAddresses

This property is only of any significance in a cluster configuration with several resource adapter instances and must not be specified at the same time as the `resourceAdapter-Index` property (see Section 4.7, "Special characteristics of multiple resource adapter mode").

This property enables you to specify the addresses of all the computers on which instances of the BeanConnect resource adapter will be run. You can specify up to 32 semicolon-delimited addresses.

You specify addresses in the format `host[:port]`. If you do not specify a port number, the port number specified under `inboundListenerPort` is used as the listener port for inbound communication. This must be greater than 0. If you specify a port number, this is used as the listener port for inbound communication. The specified value must be greater than 0.

If multiple resource adapter instances are to run under one and the same host address, then you must specify this host address the corresponding number of times in the list and assign each address a different port number.

Definition: `<host>[:<port>];` ... `;<host>[:<port>]`

Explanation: `<host>` Host on which the associated resource adapter instance is running.

`<host>` can be specified as a symbolic name or as an IPv4 address.

`<port>` Port number of the associated resource adapter instance for inbound communication.

The individual entries must be separated by semicolons.

Default: There is no default value

Example:

```
<config-property>
  <config-property-name>resourceAdapterAddresses
  </config-property-name>
  <config-property-type>java.lang.String
  </config-property-type>
  <config-property-value>
     host1:31099;host2:31099;host3:31099
  </config-property-value>
</config-property>
```

proxyReconnectCount

This property is only of any significance in a cluster configuration with multiple resource adapter instances and multiple proxy instances. `proxyReconnectCount` controls the usage-driven reassignment of a resource adapter instance to a proxy application. This mechanism is activated as soon as multiple resource adapter instances are assigned to a proxy application.

Definition: `<number>`

Explanation: `<number>` specifies the number of connection requests (calls to `getConnection()`) after which a reassignment between the resource adapter instance and the proxy application is necessary.

If the value 0 is specified for `<number>` then usage-driven reassignment is deactivated

Default: `100`

Example:

```
<config-property>
  <config-property-name>proxyReconnectCount
  </config-property-name>
  <config-property-type>java.lang.String
  </config-property-type>
  <config-property-value>200
  </config-property-value>
</config-property>
```

proxyReconnectInterval

This property is only of any significance in a cluster configuration with multiple resource adapter instances and multiple proxy instances. `proxyReconnectInterval` controls the time-driven reassignment of a resource adapter instance to a proxy application. This mechanism is activated as soon as multiple resource adapter instances are assigned to a proxy application.

Definition: `<minutes>`

Explanation: `<minutes>` specifies the time in minutes after which a reassignment between the resource adapter instance and the proxy application is necessary

If the value 0 is specified for `<minutes>` then time-driven reassignment is deactivated.

Default: `10`

Example:

```
<config-property>
  <config-property-name>proxyReconnectInterval
  </config-property-name>
  <config-property-type>java.lang.String
  </config-property-type>
  <config-property-value>5
  </config-property-value>
</config-property>
```

ℹ If a cluster configuration is operated with more resource adapter instances than proxy instances (i.e. there is always at least one proxy instance which is assigned more than one resource adapter instance), then the usage-driven and time-driven reassignment should be deactivated or, at the very least, values larger than the defaults should be set in order to avoid performance losses.

5 BeanConnect Management Console

The BeanConnect Management Console is used to configure and administer one or more BeanConnect proxies. These proxies can run on the same host as the Management Console (local proxies) or on a remote host (remote proxies).

Local proxies are proxies which run on the same host and under the same user ID as the Management Console. All other proxies are referred to as **remote proxies**.

In addition, the Management Console is a JMX client. As a result, it is therefore possible, for example, to modify the settings in the resource adapter via the BeanConnect MBeans, query statistical values for connections via the BeanConnect MBeans or access the application server's MBeans.

In addition, you can also use the Management Console to set the logging properties in the application server.

The Management Console is an administration tool with a graphical user interface (GUI).

In addition to the graphical user interface, the Management Console also possesses a command line interface with which you can also run the Management Console functions via script. This Management Console command line interface (MC-CLI) uses the Jython script language. You can find more detailed information in Chapter 9, "Command Line Interface of the BeanConnect Management Console (MC-CLI)".

i You can find further detailed information about the Management Console in the Management Console's online help system.

This chapter provides an overview of

- Starting and shutting down the Management Console
- User interface - Management Console window
- Functions of the BeanConnect Management Console
- Administrative data of the Management Console

5.1 Starting and shutting down the Management Console

This section provides information about:

- Starting the Management Console
- Starting the Management Console's online Help system
- Shutting down the Management Console

5.1.1 Starting the Management Console

Starting the Management Console under Solaris and Linux systems
You start the Management Console using the shell script `startconsole.sh`:

1. Open a shell.
2. Change to the Management Console home directory.
3. Run the shell script `startconsole.sh`.

Starting the Management Console under Windows
To start the Management Console, select **Management Console** from the **BeanConnect V3.0A00 Management Console** program group in the **Start** menu.

Select:

Start - Programs - BeanConnect V3.0A00 - Management Console - Management Console

5.1.2 Starting the Management Console's online Help system

You can start the Management Console's online help system within the Management Console window in the following ways:

- Press the **F1** key.
- Choose the **Content** command from the **Help** menu.
- Click the **Help** button in a dialog box or a panel in which you require assistance.

Alternatively, you can start the online help system without starting the Management Console:

- On Solaris/Linux systems:

  1. Open a shell.
  2. Change to the Management Console home directory.
  3. Call the script `starthelp.sh`.

- On Windows systems:

  Select the **MC Help** command from the
  BeanConnect V3.0A00 Management Console program group.

Language of the online Help system

You can change the language of the Management Console's online help system:

- On Solaris/Linux systems:

  During the installation of the Management Console, both the English and the German versions of the help system are installed. By default, the Management Console uses the English version.

  To change to the German language, move the file
  `ConsoleHelp_de.jar` to `ConsoleHelp.jar`
  to the `lib` subdirectory of the BeanConnect installation directory.

- On Windows systems:

  You can select the language for the Management Console's online help system during the installation of BeanConnect (see Section 3.3, "Installing BeanConnect under Windows systems" on page 60). By default, the English version is selected.

5.1.3 Shutting down the Management Console

To shut down the Management Console, choose **Exit** from the **File** menu.

5.2 User interface - Management Console window

The Management Console window contains the components described below:

- A menu bar is located at the top of the Management Console window.
  The menu bar contains the **File**, **View**, **Extras**, **Window** and **Help** menus.

- The central part of the window is divided into the following areas:

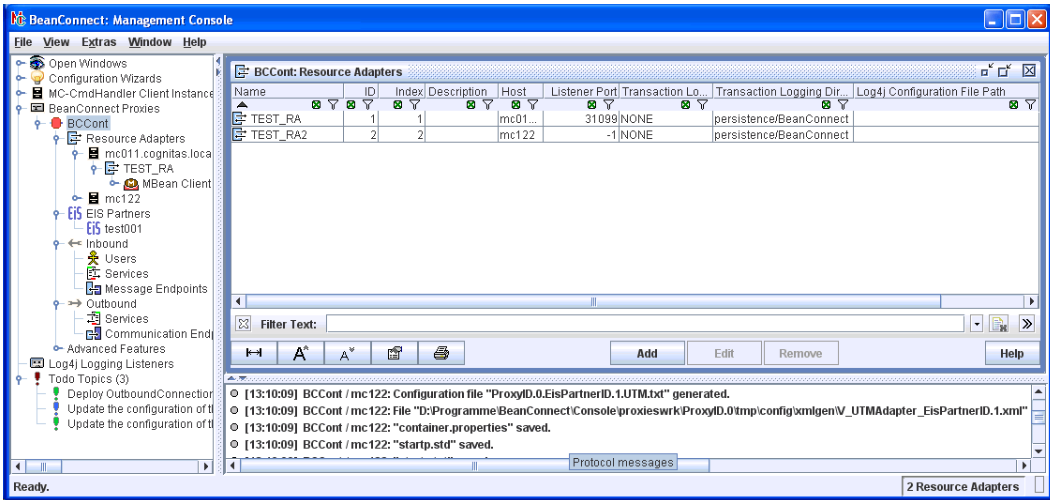
  - Navigation area (on the left)
    This area displays a tree structure (the navigation tree) containing the administered proxies together with the resources and settings they contain.

  - Work area (in the top right area of the window)
    This area displays configuration data associated with the entries you have selected in the navigation area.

  - Protocol (log) window (in the bottom right part of the window)
    This area contains the messages of the current Management Console session. The log window can be permanently displayed or hidden.

- At the bottom of the window you will find the status bar.
  The status bar displays the processing status of time-intensive jobs in the form of text messages.

Figure 12: User interface of the Management Console



Name	ID	Index	Description	Host	Listener Port	Transaction Lo...	Transaction Logging Dir...	Log4j Configuration File Path
TEST_RA	1	1		mc01...	31099	NONE	persistence/BeanConnect	
TEST_RA2	2	2		mc122	-1	NONE	persistence/BeanConnect	

```
[13:10:09] BCCont / mc122: Configuration file "ProxyID.0.EisPartnerID.1.UTM.txt" generated.
[13:10:09] BCCont / mc122: File "D:\Programme\BeanConnect\Console\proxieswrk\ProxyID.0\tmp\config\xmlgen\V_UTMAdapter_EisPartnerID.1.xml"
[13:10:09] BCCont / mc122: "container.properties" saved.
[13:10:09] BCCont / mc122: "startp.std" saved.
```

Ready.

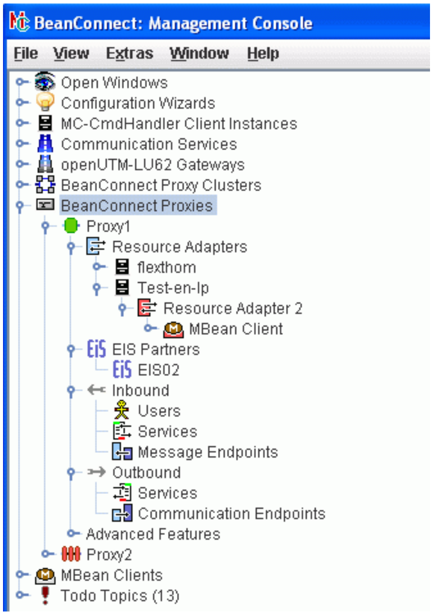
2 Resource Adapters

5.2.1 Navigation area in the Management Console

The administered BeanConnect proxies are displayed in a tree structure in the navigation area (in the same way as the drives and directories in Windows Explorer). There is a separate subtree under the BeanConnect proxies node for every BeanConnect proxy that is administered via the Management Console (Proxy1 in Figure 13). This subtree allows you to display and edit the configuration data of the BeanConnect proxy. If you click an item in the subtree, the configuration data associated with this component is displayed in a panel in the work area. The context menu of the entry in question also contains commands which allow you to modify the configuration data.

The figure below shows the navigation area of the Management Console:

Figure 13: Navigation area



5.2.2 Managed objects

No proxy container needs to be running in order to administer objects.

A distinction is made between the following objects:

MC-CmdHandler Client Instances
One object for each administered MC-CmdHandler. An MC-CmdHandler enables you to access the filesystem at the relevant host and execute scripts there.

Communication Services
One object for each administered communication service. A communication service is required for communications with CICS partners. It can run either locally or on a remote host.

openUTM-LU62 Gateways
One object for each managed openUTM-LU62 Gateway. An openUTM-LU62 Gateway is required alongside the communication service for communications with CICS partners. In each case, it runs on the same host as the associated communication service.

BeanConnect Proxy Clusters
One object for each administered BeanConnect proxy cluster. A proxy cluster consists of one or more proxies which must be configured beforehand.

BeanConnect Proxies
One object for each administered BeanConnect proxy which is not located in a proxy cluster.

Only "administrable" BeanConnect proxies can be managed using the Management Console. A BeanConnect proxy is considered administrable if one of the following applies:

- The proxy is a local proxy.
- The proxy is a remote proxy and the MC-CmdHandler associated with the proxy is available and can be accessed via the Management Console.

Resource Adapters
One or more resource adapters is assigned to each proxy. A resource adapter runs on an application server instance.

MBean Clients

An MBean client can be defined for each resource adapter. The Management Console can use the MBean client to access the MBeans of the relevant JMX server (display attributes and modify their values, run operations and receive notifications).

It is also possible to define "free" (stand-alone) MBean clients which are not assigned to any resource adapter. Free MBean clients are displayed on the topmost level of the navigation tree.

EIS Partners

One object for each administered EIS partner.

Inbound Users

The user information (user name and password) may be passed by the EIS partner on inbound communication. This user information must be known to the proxy.

Inbound Message Endpoints

An inbound message endpoint represents a communication endpoint for inbound communication. A proxy can manage several inbound message endpoints.

Inbound Services

An inbound service is a service which an EIS partner addresses during inbound communication. A service is always assigned to precisely one inbound message endpoint. However, it is possible to assign multiple services to an inbound message endpoint. You can also define coding properties for a service.

Outbound Services

An outbound service represents a service (transaction code (TAC) or CICS transaction) that is provided by the EIS partner.

Outbound Communication Endpoints

For each (symbolic) service that is specified in the `connectionURL` configuration property in the application server, you must configure a corresponding outbound communication endpoint of the same name in the proxy. The outbound communication endpoint definition maps the symbolic service name onto a real service name in the EIS partner application.

Outbound communication endpoints are specific to the EIS partners. Several outbound communication endpoints can be assigned to one EIS partner.

For detailed information see section Section 6.7.2, "Configuring outbound communication endpoints".

5.2.3 Additional functions and information

You can find more functions and information in:

Advanced Functions

The advanced features provide statistical information and diagnosis control. You can find detailed information in Chapter 13, "Logging, diagnostics and troubleshooting".

Todo Topics

The Management Console provides a list of todo topics. The todo list contains the most important activities that need to be performed in order to activate modifications made to the configuration data of the BeanConnect proxies. You can also use the todo topic list to add your own todo topics. You will find further details in the section Section 5.3.6, "Todo topics".

5.3 Functions of the BeanConnect Management Console

The Management Console supports you in tasks associated with:

- Configuration functions
- Configuration wizards
- Starting and stopping proxies
- Checking the availability of BeanConnect components and EIS partners
- Diagnosis support
- Todo topics
- Cluster support
- Management Console as a JMX client

ℹ You should avoid to concurrently configure or administrate a BeanConnect proxy from multiple Management Console sessions. Otherwise settings of one Management Console session could be overwritten by another session and data may be lost.

5.3.1 Configuration functions

Configuration using the Management Console covers the following activities:

- configuring the BeanConnect proxies
- configuring the EIS partners

You can also use the Management Console command line interface to run Management Console configuration functions via a script. You can find more detailed information in Chapter 9, "Command Line Interface of the BeanConnect Management Console (MC-CLI)".

Configuring the BeanConnect proxies

When you configure a proxy, the following activities can be initiated from the Management Console GUI:

1. define and modify the configuration
2. save the configuration
3. shut down the proxy

4. update the configuration of the proxy
5. start the proxy

If you want to integrate an EIS partner in the Management Console or if you want to modify or remove an EIS partner, you must perform the above steps in the specified order. If you want to generate, modify or delete an outbound communication endpoint or an inbound message endpoint, you can update the configuration while the proxy is running. In this case, only steps 1 and 2 have to be performed. You then have to restart the proxy.

When you configure a proxy, the Management Console assists you by displaying todo topics (see Section 5.3.6, "Todo topics").

Further details on proxy configuration can be found in section Section 6.3, "Configuring the BeanConnect proxy".

Configuring the EIS partners

The Management Console allows you to create configuration fragments for the generated EIS partners. These configuration fragments are created and saved in files when the BeanConnect proxy configuration is saved. However, it is your responsibility to perform the following tasks:

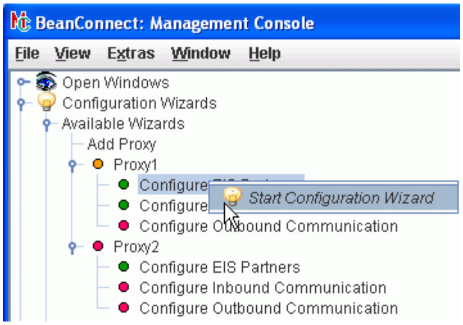
- Transfer the generated files containing the configuration statements for the EIS partner applications to the host on which the EIS partner is running.
- Integrate these files into the EIS partner's configuration.

You will find further details in Chapter 7, "Adapting the configuration in the EIS partner".

5.3.2 Configuration wizards

The Management Console provides wizards for the configuration of individual proxies and their components. This means that even less experienced users can create a configuration without forgetting parameters. You start the configuration wizards via the context menu, for example by opening the wizard subtree under **Configurations Wizards** and choosing the context menu command **Start Configuration Wizard** for the required wizard.

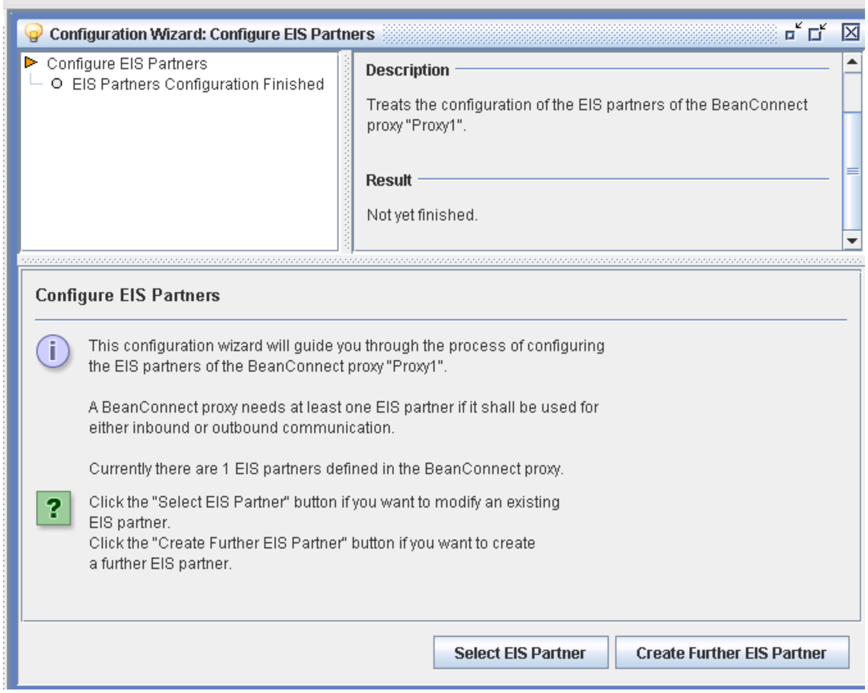
Figure 14: Starting the configuration wizard


BeanConnect: Management Console
File View Extras Window Help
Open Windows
Configuration Wizards
Available Wizards
Add Proxy
Proxy1
Configure
Configure
Start Configuration Wizard
Configure Outbound Communication
Proxy2
Configure EIS Partners
Configure Inbound Communication
Configure Outbound Communication

Alternatively, choose **Configuration Wizards** from the **File** menu.

The wizard itself runs in a window which consists of three sections.

Figure 15: Configuration wizard window


Configuration Wizard: Configure EIS Partners
Configure EIS Partners
EIS Partners Configuration Finished
Description

Treats the configuration of the EIS partners of the BeanConnect proxy "Proxy1".

Not yet finished.

This configuration wizard will guide you through the process of configuring the EIS partners of the BeanConnect proxy "Proxy1".

A BeanConnect proxy needs at least one EIS partner if it shall be used for either inbound or outbound communication.

Currently there are 1 EIS partners defined in the BeanConnect proxy.

Click the "Select EIS Partner" button if you want to modify an existing EIS partner.
Click the "Create Further EIS Partner" button if you want to create a further EIS partner.

The top left section of the wizard (in the example, **Configure EIS Partners**) provides an overview of all the individual tasks present in the configuration wizard in the form of a tree structure.

The section to the right of this describes the individual task currently selected in the tree structure.

The lower section provides a detailed description of the current individual task. You may be able to choose from a selection of options in order to guide the configuration process. You do this using the buttons at the lower edge of the screen, in this case **Select EIS Partner** and **Create Further EIS Partner**.

The user can exit a configuration wizard at any time by clicking the cross at the top right of the window to close it.

If configuration with the wizard has not been terminated in full, the user is informed of this and is able to save the current state of the wizard. This means that it is possible to continue the commenced configuration at a later date (possibly even in a subsequent program session).

See the online Help system for further details about the wizard window.

5.3.3 Starting and stopping proxies

You can use the Management Console to start, stop and restart BeanConnect proxies and proxy components on both local and remote hosts.

Starting and stopping a BeanConnect proxy

Select the command **Start Proxy** or **Stop Proxy** from the BeanConnect proxy's context menu to start or stop the proxy.

In the case of proxies for CICS partners, a proxy consists of multiple components. The Management Console allows you to start/stop individual proxy components or all the components of a proxy. A special dialog box is displayed in which you can select the components that are to be started/stopped. You can select the following components:

- proxy container
- openUTM-LU62 Gateway (for CICS partners)
- SNAP-IX or IBM Communications Server (for CICS partners)

> i For CICS partners:
>
> You must not stop the openUTM-LU62 Gateway and SNAP-IX or the IBM Communications Server as long as other applications that use these components are running (such as proxy containers, for example).

The Management Console only tries to start the proxy component if the proxy component in question is not already running.

Restarting the BeanConnect proxy

Select the command **Save/Restart - Restart Proxy** from the BeanConnect proxy's context menu.

In the case of proxies for CICS partners, when you perform a restart then, in the same way as a normal start/shutdown, a dialog box is displayed in which you can select the components that are to be restarted.

When restarting a proxy container the individual processes are stopped one after the other and are then restarted. The proxy container remains available during the restart.

5.3.4 Checking the availability of BeanConnect components and EIS partners

You can use the Management Console to check the availability of a proxy and all the components that are assigned to this proxy. These also include the EIS partners. You can perform the check for the proxy or individually for certain components.

To do this, choose the **Check Availability** command from the required object's context menu. The command is available for the following objects.

- Proxy

  Checks the availability of the proxy container and of all the components assigned to the proxy: All resource adapters, all MC-CmdHandlers on remote hosts if these are used to administer remote proxies as well as the openUTM-LU62 Gateway and SNAP-IX or IBM Communications Server if the proxy is configured for CICS partners. The availability of all assigned EIS partners is also checked.

- Resource adapter

  Only checks the availability of a specified resource adapter. The associated proxy container must be running.

- openUTM-LU62 Gateway / communication service

  Checks the availability of an openUTM-LU62 Gateway or a communication service (SNAP-IX, IBM communications service). The **openUTM-LU62 Gateway** and **communication services** objects can be accessed via the topmost level in the navigation tree. If the components are installed on a remote host then the associated MC-CmdHandler must be running.

- EIS partner

  Checks the availability of a specified EIS partner. The associated proxy container must be running.

For further details, see Chapter 8.6, "Checking the availability of BeanConnect proxies".

5.3.5 Diagnosis support

The Management Console helps you to diagnose problems that occur in the environment of a BeanConnect proxy. The Management Console therefore provides a range of functions that allow you to do the following:

- configure traces and logging
- collect and display traces and logs

For details see Chapter 13, "Logging, diagnostics and troubleshooting".

5.3.6 Todo topics

The Management Console displays a todo list containing an overview of the activities that need to be performed (e.g. for update the configuration).

> ℹ The todo list represents an overview of the actions that are to be performed. It is important to note that the list also contains todo topics relating to actions that have to be performed on remote (application/EIS) servers. Those actions cannot be started from the local host. They must be started manually on the remote host.

Certain actions cause the Management Console to add todo topics to the list automatically. Some of these topics refer to actions you can perform using the Management Console. In such cases, the relevant items are removed from the list automatically once the associated actions have been completed. You can also perform such actions directly from the topic list.

Todo topics may also involve actions that you cannot perform via the Management Console, for instance adding the generation statements created by the Management Console to the configuration of an EIS partner. In such cases, the Management Console cannot identify whether or not you have performed the action. Consequently, when you have completed an action of this type you must remove the corresponding topic from the list yourself.

You can also create your own todo topics and add them to the list to avoid having to note them elsewhere.

5.3.7 Cluster support

A proxy cluster consists of one or more proxies. You can configure and administer a proxy cluster via the Management Console. The procedure is similar to that for configuring and administering an individual proxy.

Many of the menu commands and activities are the same since, in most cases, the proxy cluster behaves in the same way as an individual proxy.

Configuration wizards are not supported in a proxy cluster.

Configuring a BeanConnect proxy cluster

Perform the steps below to configure a proxy cluster consisting of multiple proxies:

1. Configure an individual proxy which will act as the basis for the proxy cluster.
2. Choose the command **Define Proxy Cluster** in this proxy's context menu. This defines the proxy cluster and simultaneously enters the proxy in this cluster. This action automatically makes this proxy the master proxy in the new cluster.
3. Enter further proxies in the Management Console. You do not have to configure these since the configuration properties are overwritten when they are included in the cluster and synchronized with the master proxy.
4. Add the new proxies to the cluster by choosing **Add to Proxy Cluster** in the context menu.
5. Save the cluster configuration. This synchronizes the proxies' configuration data.

You can modify the configuration properties of a proxy cluster in the cluster network, e.g. modify or add EIS partners, inbound services or outbound services.

The Management Console also provides a manual synchronization function for the proxy cluster (**Synchronize Proxy Cluster** command in the proxy cluster's context menu). This function is needed, for example, if a proxy in a cluster could not be administered for a period (for example because the corresponding host was not running) and the configuration of the proxy cluster changed during this period. An explicit synchronization then harmonizes the configuration of this specific proxy in the cluster with the configuration of the proxy cluster as a whole.

For further information, see Chapter 6.4, "Configuring a BeanConnect proxy cluster".

Starting and stopping a BeanConnect proxy cluster

You start, stop and restart a proxy cluster in the same way as when starting, stopping and restarting an individual proxy. Choose the corresponding command in the proxy cluster's context menu.

Start Proxy Cluster to start
Stop Proxy Cluster to stop
Save/Restart - Restart Proxy Cluster to restart

Checking availability

You check the availability of a proxy cluster or its components in the same way as for an individual proxy by choosing the **Check Availability** command in the associated context menu.

In a proxy cluster, the command initiates a check of all the proxy containers and all the associated components, i.e. resource adapters, EIS partners, MC-CmdHandlers on remote hosts and the openUTM-LU62 Gateway and communication service in the case of connections to CICS partners.

At the level of the individual components, the command only checks these components. The openUTM-LU62 Gateway and the communication service are checked at the topmost level in the same way as for an individual proxy.

5.3.8 Management Console as a JMX client

The Management Console also contains a JMX client which permits you to monitor a resource adapter. It is possible to configure multiple JMX clients.

A JMX client can either be assigned to a resource adapter or can be configured as a free JMX client. To assign it to a resource adapter, you define the JMX client using the **Define MBean Client** command in the resource adapter's context menu and then specify its properties in the dialog which then follows, see Section 6.11.2, "Setting up the JMX client in the Management Console".

A JMX client communicates with the JMX server. This runs on the application server and makes the data available via so-called MBeans. The JMX client is therefore also known as the MBean client. This allows you to monitor the following objects:

- Resource adapters
- Connection factories
- Inbound connections
- Message endpoints

The MBean client can read the data and attributes made available by the MBeans. It also has write access to certain attributes. The following monitoring functions are provided:

- Display MBeans attributes

  The attributes include, for example, the configuration properties or certain statistical values of the corresponding MBean.

- Read notifications output by the MBeans

  Notifications must first be subscribed to via the MBeans.

  Notifications are messages which the application server generates when certain events occur. Notifications are also generated by the resource adapter and by the application server components which supply the MBeans.

- Collect and display statistical values

  Attribute values can be collected at fixed intervals and output in statistical form. These include, for example, the number of active connections or the number of rolled back transactions.

- Run operations

  Operations are specific functions that are implemented by the MBeans and accessible via the interface; in the case of connection factories, these are, for example, the functions `cleanupPool` and `resetStatisticValues`.

You can use the administration facilities to set the monitoring functions you want to use for each MBean client (notifications, statistics etc.). For this to be possible, there must be a connection between the Management Console and the JMX server on the application server. You can also define favorites to permit simple access to frequently used MBeans.

For more detailed information on defining and activating monitoring functions, see Section 8.7, "Monitoring the resource adapter with the Management Console".

5.4 Administrative data of the Management Console

The Management Console uses the files listed below:

- console.properties.xml
- log4j.properties.xml

console.properties.xml

This XML file is located in the Management Console's config subdirectory. console.properties.xml contains the administrative data of all BeanConnect proxies that are known to the Management Console. This file is also used to store additional settings for the Management Console.

The console.properties.xml file is automatically updated or extended as necessary by the Management Console. It is not necessary to save the updated data explicitly.

log4j.properties.xml

This XML file is located in the Management Console's config subdirectory. The log4j.properties.xml file is the Log4j configuration file used by the Management Console. You can update this file in the Management Console's **Settings** dialog box. Select **Extras** - **Settings...** to open this dialog box.

6 Configuration of BeanConnect

The BeanConnect proxy communicates with the BeanConnect resource adapter running within the application server on the one hand and with the EIS partner on the other hand. The proxy is not used for outbound communication via UPIC.

To ensure communication between an EJB deployed in the application server and a partner application, the proxy and the proxy components must be configured properly.

The BeanConnect Management Console allows you to carry out the configuration tasks for the BeanConnect proxy container and the proxy components. It is also possible to change the configuration of the proxy clusters, EIS partners, the outbound services, the outbound communication endpoints and the inbound message endpoints, inbound services and Management Console command handlers (MC-CmdHandlers). The Management Console can also be configured as a JMX client for the display of MBeans

You can add new objects and change or delete existing objects.

This chapter provides information on the following topics:

- Configuration steps
- Adding a BeanConnect proxy to the Management Console
- Configuring the BeanConnect proxy
- Configuring a BeanConnect proxy cluster
- Configuring the BeanConnect resource adapter
- Configuring the EIS partners
- Configuring outbound communication
- Configuring inbound communication
- Saving and activating the configuration of the BeanConnect proxy
- Configuring the Management Console command handler (MC-CmdHandler)
- Configuring the Management Console as a JMX client

Detailed information on using the BeanConnect Management Console can be found in Chapter 5, "BeanConnect Management Console" and in the online help system.

6.1 Configuration steps

The following steps must be performed to configure a proxy using the Management Console:

1. Adding the proxy to the configuration data of the Management Console

   Before it can be administered, each proxy must be made known to the Management Console. This procedure is described in Section 6.2, "Adding a BeanConnect proxy to the Management Console".

2. Configuring the proxy and the proxy components

   When configuring the proxy and the proxy components, you have to make all the specifications which allow the partners to communicate with each other. This includes:

   – the specifications for identifying the proxy (Section 6.3.1, "General information on the proxy").

   – additionally for CICS partners, the settings for communication with the openUTM-LU62 Gateway and the communication service, see Section 6.3.2, "Proxy Components: CICS partners"), as well as specifications for the network connections for CICS partners, see "CICS Partner tab" on page 213.

   – the settings for communication with the resource adapter assigned to the proxy (see Section 6.5, "Configuring the BeanConnect resource adapter").

3. Configuring the EIS partners and communication objects

   In this step of the configuration process, you define the communication relationships between the proxy and the EIS partners for inbound and outbound communication. The following objects must be created:

   – EIS parttners (see Section 6.6, "Configuring the EIS partners")

   – Outbound services and outbound communication endpoints (see Section 6.7, "Configuring outbound communication")

   – Inbound users, inbound services and inbound message endpoints (see Section 6.8, "Configuring inbound communication")

4. Saving and activating the configuration

   After all the necessary information has been provided, you have to save the configuration. The properties of the configuration objects are stored by the Management Console, so that they are available for subsequent Management Console sessions. The Management Console then generates the configuration files to be used by the proxy, the components of the proxy and the EIS partners. The last step is to activate the new configuration.

These tasks are described in the section "Saving and activating the configuration of the BeanConnect proxy" on page 233.

If you want to form a proxy cluster, you will find the relevant information on Section 6.4, "Configuring a BeanConnect proxy cluster".

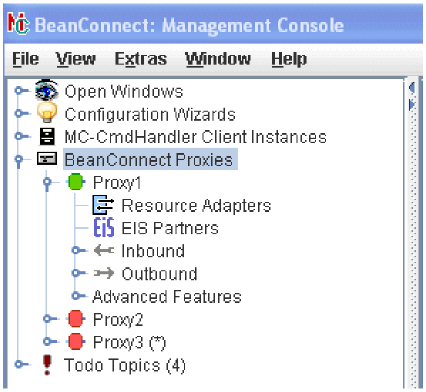
For information on configuring the Management Console as a JMX client, see Section 6.11, "Configuring the Management Console as a JMX client".

6.2 Adding a BeanConnect proxy to the Management Console

All proxies which are known to the Management Console and are not assigned to a proxy cluster are shown together with the configured EIS partners and communication objects in the navigation tree of the Management Console below the **BeanConnect Proxies** node.

One Management Console can be used to administer multiple proxies.

Figure 16: Proxies within the Management Console



To access the data of a proxy, you have to enter the administration password. By default, this is **admin.** You can change the administration password, see Section 6.3.3, "Modifying the administration password".

6.2.1 Adding a new proxy

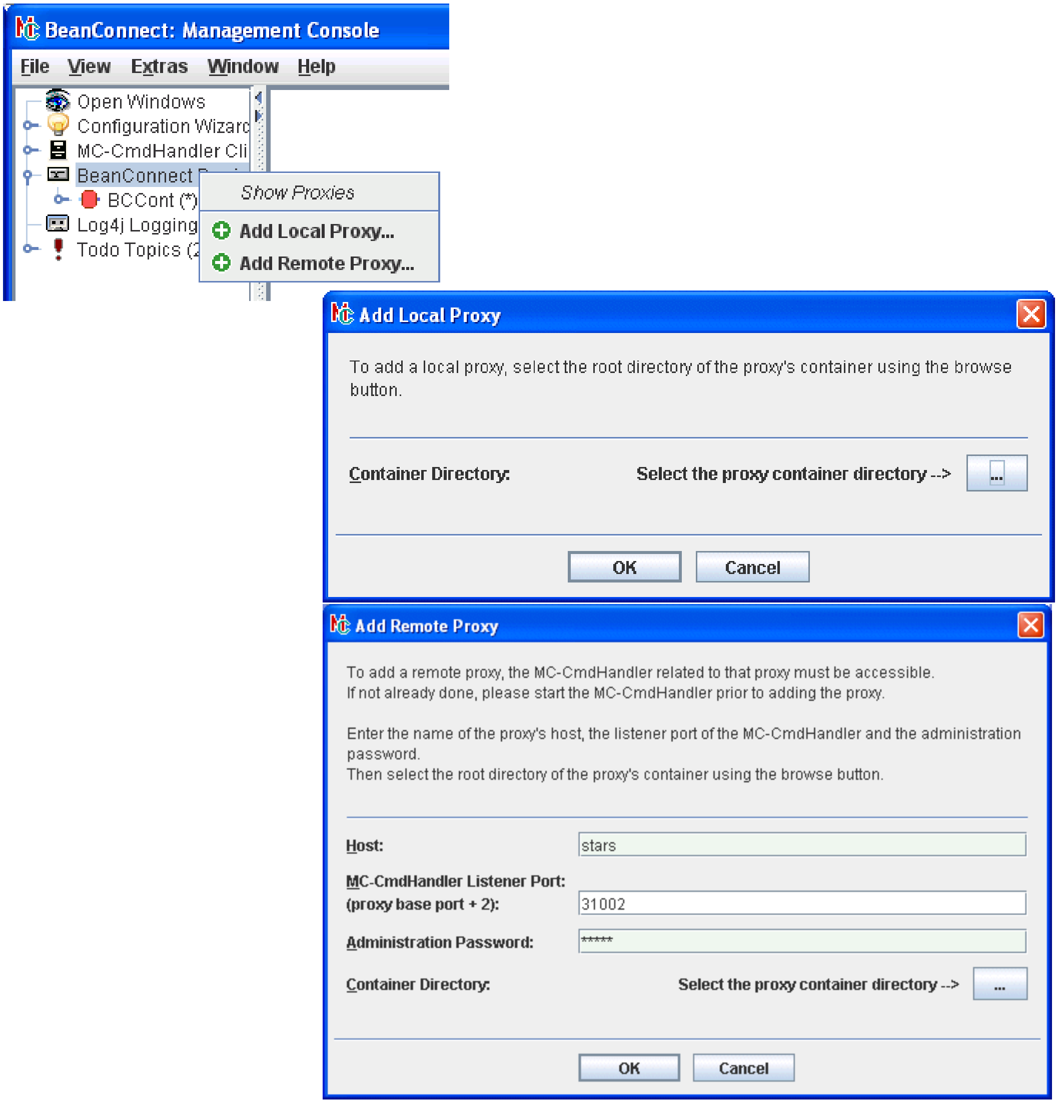
To add a new proxy to the navigation tree:

- Choose **Add Local Proxy** from the context menu of the **BeanConnect Proxies** node if the proxy is located on the same host and under the same user ID as the Management Console. It is not necessary to use the same user ID if the access permissions have been set in such a way that the Management Console is able to access all the proxy's required files under the user ID under which the proxy runs.
- Choose **Add Remote Proxy** from the context menu of the **BeanConnect Proxies** node if the proxy is located on another host. You must enter the name of the host the proxy container is running on.

  To add and manage a proxy on a remote host, the proxy's MC-CmdHandler must be running on the proxy host (see Section 6.10, "Configuring the Management Console command handler (MC-CmdHandler)". The password of the employed MC-CmdHandler and the administration password of the added proxy must be identical!

Enter the name of the remote host under **Host** and the MC-CmdHandlers's listener port on the remote host under **MC-CmdHandler Listener Port**.

Figure 17: Adding a new proxy



To add a local proxy, select the root directory of the proxy's container using the browse button.

To add a remote proxy, the MC-CmdHandler related to that proxy must be accessible.
If not already done, please start the MC-CmdHandler prior to adding the proxy.

Enter the name of the proxy's host, the listener port of the MC-CmdHandler and the administration password.
Then select the root directory of the proxy's container using the browse button.

Host: stars

MC-CmdHandler Listener Port:
(proxy base port + 2): 31002

Administration Password: *****

When adding a local proxy a container selection dialog box is opened. Choose the container directory by browsing the file system.

Both menu entries can also be found in the **File** menu and can also be activated from there.

i Adding proxies to the Management Console's configuration data does not install new proxies. You only can add proxies which have already been installed.

A proxy added to the Management Console can not be fully managed until all the necessary parameters for the proxy and the proxy components are defined (see "Configuring the BeanConnect proxy" on page 173). An appropriate message is created if you try to call a function which is not available before having finished the necessary configuration.

A proxy, installed beneath the same BeanConnect home directory as the Management Console, will be detected the next time the Management Console is started and will be added to the configuration data automatically. This also applies if the proxy is installed on the system after the Management Console has been installed.

6.2.2 Removing a proxy

To remove a proxy from the configuration data of the Management Console, select **Remove Proxy** from the context menu of the proxy.

After you have confirmed the prompt, the proxy is deleted from the configuration data. Apart from that, the proxy itself will not be changed. In particular, it will not be deinstalled.

6.3 Configuring the BeanConnect proxy

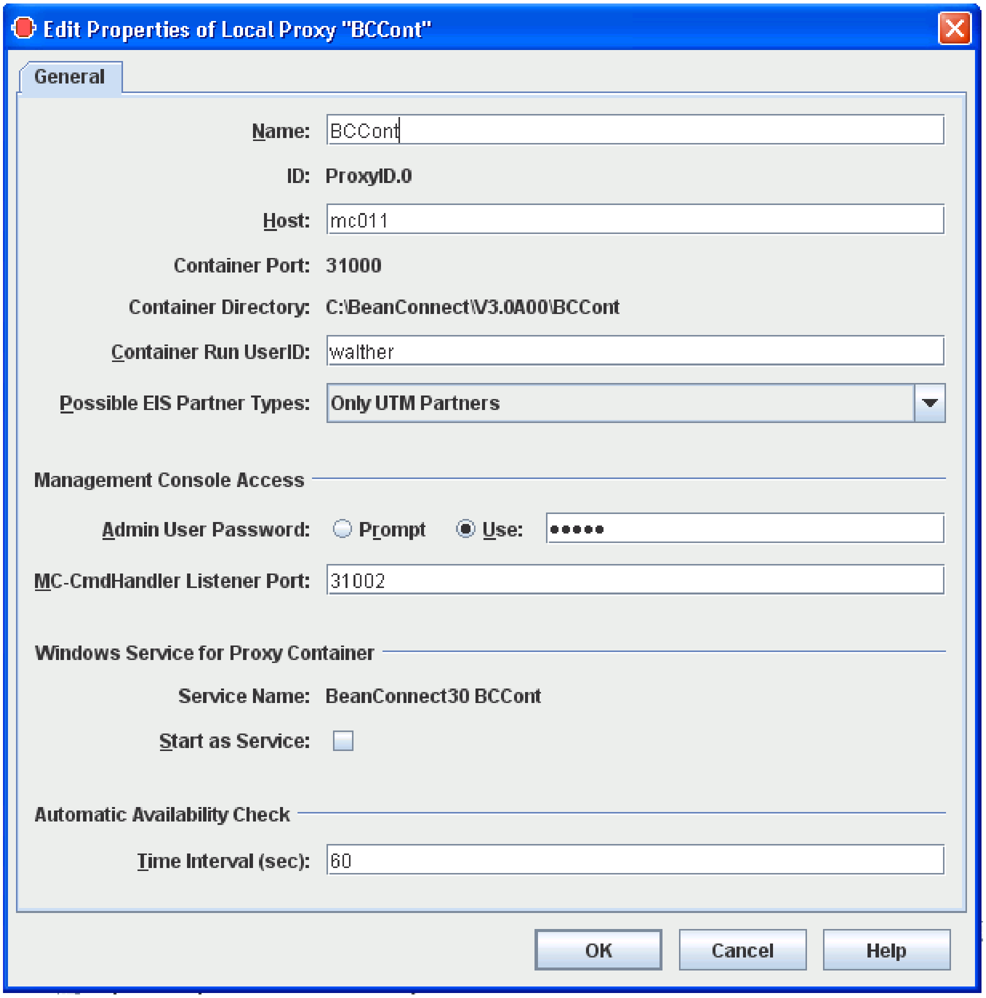
When you add a new proxy to the Management Console, a property sheet is displayed automatically to allow you to enter the configuration data for the proxy components. To change the configuration data of an existing proxy, select the desired proxy in the navigation tree and choose **Edit Properties** from the context menu.

The property sheet consists of several pages. These are described below.

6.3.1 General information on the proxy

The **General** page contains some general information to allow the Management Console to identify the proxy and access it.

Figure 18: General information on a proxy



Edit Properties of Local Proxy "BCCont"

Name: BCCont

ID: ProxyID.0

Host: mc011

Container Port: 31000

Container Directory: C:\BeanConnect\V3.0A00\BCCont

Container Run UserID: walther

Possible EIS Partner Types: Only UTM Partners

Management Console Access

Admin User Password: ○ Prompt ● Use: •••••

MC-CmdHandler Listener Port: 31002

Windows Service for Proxy Container

Service Name: BeanConnect30 BCCont

Start as Service: □

Automatic Availability Check

Time Interval (sec): 60

Name

Proxy Name. This name is only used internally by the Management Console to distinguish between the managed proxies.

ID

The Management Console assigns an **ID** to each proxy in the form `ProxyID.<number>`. This ID is shown in the dialog box after a proxy is added for the first time and cannot be changed. The Management Console uses this ID as a name component for the generated configuration files.

Host

Name or IP address of the host on which the proxy is installed

Container Port / Container Directory

These fields indicate the proxy's port number and home directory. The port number is specified during installation.

Container Run UserID

User ID under which the proxy container is started. The Management Console compares the user ID specified here with the user ID under which the MC-CmdHandler used for administering the proxy container is running. Only if both user IDs are identical does the Management Console judge that the proxy container can be administered with the MC-CmdHandler used.

In the case of local proxies, please note that in certain situations, e.g. if the proxy container's MC-CmdHandler has not been started and this MC-CmdHandler's listener port has not been assigned to the proxy then the system uses the "internal" MC-CmdHandler which runs in the Management Console process and operates without communications. In this case, the user ID specified here must match the user ID under which the Management Console itself was started.

Possible EIS Partner Types

Specifies the EIS partner type for which the proxy is configured. If you choose **Only UTM Partners** then the proxy can only communicate with openUTM partners. If you choose **Only CICS Partners** then the proxy can only communicate with CICS partners. Choosing the option **UTM and CICS Partners** enables communications with both partner types. An additional tab – **Proxy Components** – is displayed in the case of communication with CICS partners.

ℹ Separate licenses must be purchased to communicate with the partner types UTM and CICS.

Management Console Access

Additionally, you must specify the administration password used by the Management Console to access the proxy container and its MC-CmdHandler in the **Admin User Password** field. Instead of specifying a password you can choose the option **Prompt**. In this case the password is requested on the first access to any proxy container data within a Management Console session.
The field **MC-CmdHandler Listener Port** defines the listener port of the MC-CmdHandler used to administer the proxy. The default value is the value in **Container Port** + 2.

Windows Service for Proxy Container

Only if the proxy container is running on a Windows system: Proxies can also be started as Windows services. To do this, select the option **Start as Service**. The name to be used to start the proxy as a service is defined during installation and is shown in the field **Service Name**. If the proxy is operated as a Windows service, it cannot be started in debug mode.

Automatic Availability Check

Finally, you can set the time interval for the automatic availability check in the **Time Interval (sec)** field. The values you set should not be too small in order to avoid overburdening the Management Console (and the proxies) with availability checks. Values of 180 seconds and higher are recommended.

No automatic availability check is performed if you leave the field empty or enter the value 0.

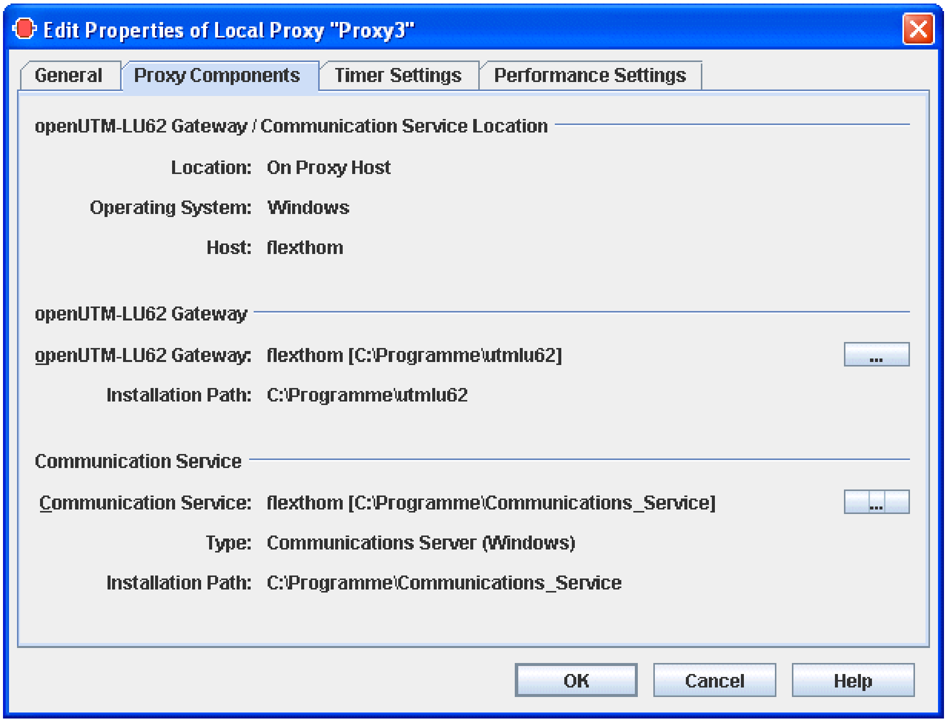
6.3.2 Proxy Components: CICS partners

In the **Proxy Components** tab in the property sheet **Edit Properties of Local/Remote Proxy**, you define the settings for the proxy components openUTM-LU62 Gateway and communication service.

These two proxy components are required for communication with CICS partners and must always run on the same computer. This does not have to be the computer on which the proxy is running.

The tab illustrated below already contains the necessary entries. When you perform initial set-up, further dialog boxes are displayed for the entry of the openUTM-LU62 Gateway and the communication server data.

Figure 19: Description of the proxy components for CICS partners



openUTM-LU62 Gateway / Communication Service Location

Location: On Proxy Host

Operating System: Windows

Host: flexthom

openUTM-LU62 Gateway

openUTM-LU62 Gateway: flexthom [C:\Programme\utmlu62]

Installation Path: C:\Programme\utmlu62

Communication Service

Communication Service: flexthom [C:\Programme\Communications_Service]

Type: Communications Server (Windows)

Installation Path: C:\Programme\Communications_Service

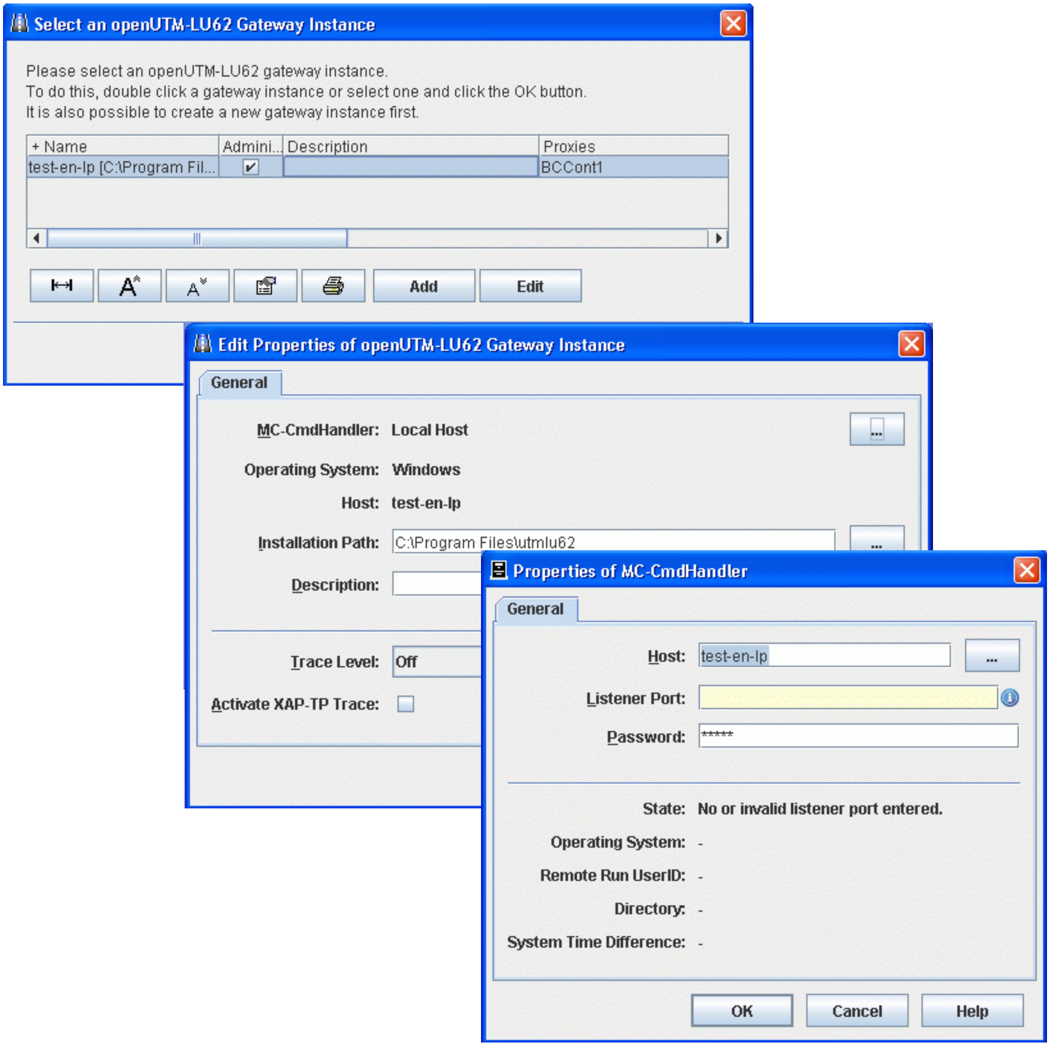
openUTM-LU62 Gateway/Communication Service Location

Displays the location, the name and the operating system of the computer on which the proxy components openUTM-LU62-Gateway and communication service are running.
Location: On Proxy Host means: The same computer on which the proxy container is running.
Location: On Separate Host means: A different computer.

openUTM-LU62 Gateway

Specifies the computer and the directory in which the openUTM-LU62 Gateway is installed. To select a gateway, click the **...** button to open the following sequence of dialog boxes:

Figure 20: Configuring the openUTM-LU62 Gateway



Select an openUTM-LU62 Gateway Instance

Please select an openUTM-LU62 gateway instance.
To do this, double click a gateway instance or select one and click the OK button.
It is also possible to create a new gateway instance first.

+ Name	Admini...	Description	Proxies
test-en-lp [C:\Program Fil...	☑		BCCont1

Edit Properties of openUTM-LU62 Gateway Instance

General

MC-CmdHandler: Local Host

Operating System: Windows

Host: test-en-lp

Installation Path: C:\Program Files\utmlu62

Description:

Trace Level: Off

Activate XAP-TP Trace: □

Properties of MC-CmdHandler

General

Host: test-en-lp

Listener Port:

Password: *****

State: No or invalid listener port entered.

Operating System: -

Remote Run UserID: -

Directory: -

System Time Difference: -

The gateway instances that have already been configured are displayed in **Select an openUTM-LU62 Gateway Instance**. You can select an instance and click **OK** to assign this instance. You can click **Edit** to modify the properties of a selected instance.

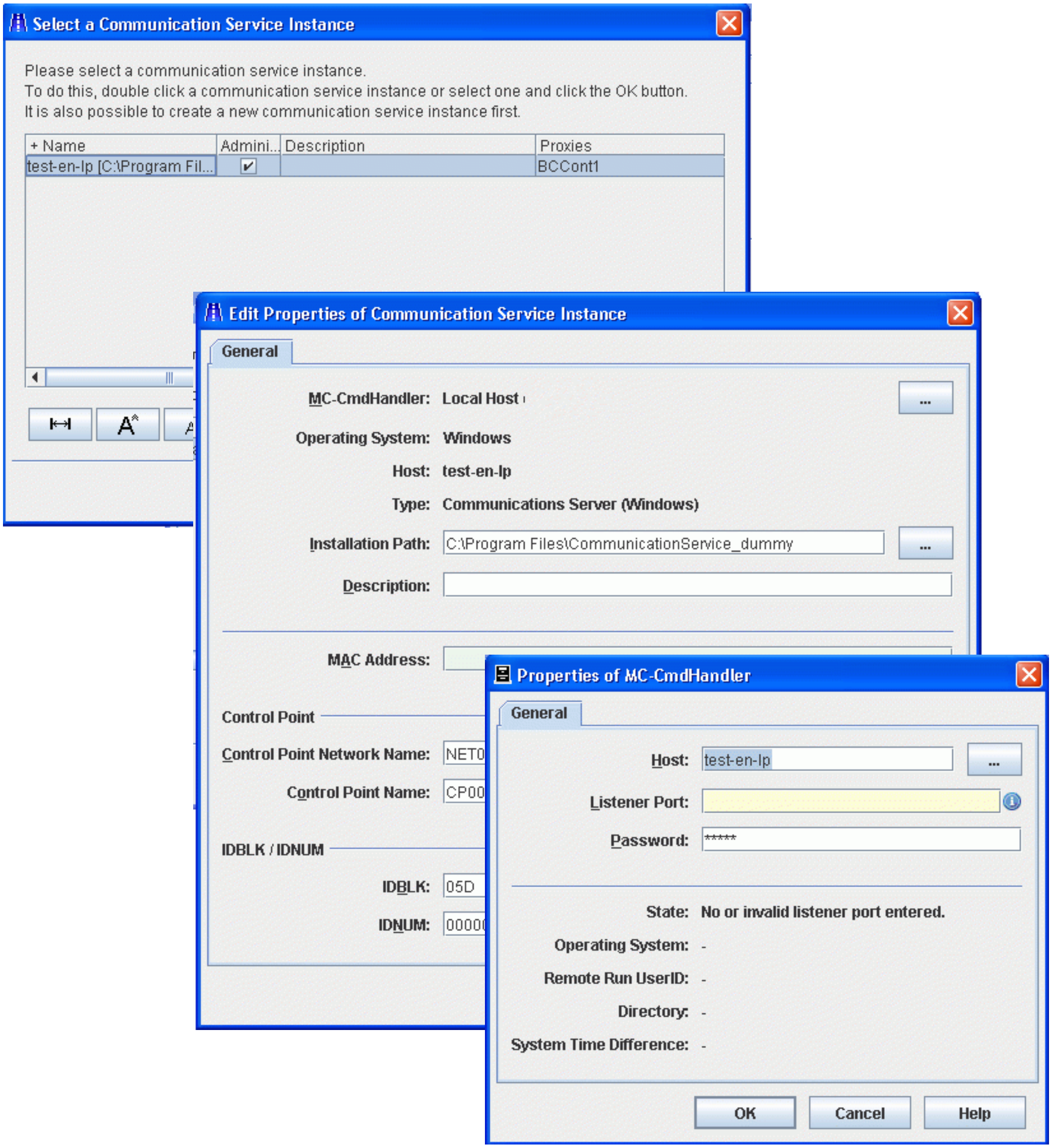
Proceed as follows if you want to configure a new instance:

- Click the **Add** button.
- In **Add openUTM-LU62 Gateway Instance**, click the **...** button under **Select MC-CmdHandler** to open the dialog **Properties of MC-CmdHandler**.
- In **Properties of MC-CmdHandler**, enter the properties of the MC handler used to administer the components and click **OK**.
- In **Add openUTM-LU62 Gateway Instance**, enter the installation path of the openUTM-LU62 Gateway and modify the traced properties if necessary. Click **OK**.

Communication Service

Specifies the computer, the type and the directory in which the communication service is installed. To select a communication service, click the **...** button to open the following sequence of dialog boxes: Proceed in the same way as when entering an openUTM-LU62 Gateways:

Figure 21: Configuring a communication service



Select a Communication Service Instance

Please select a communication service instance.
To do this, double click a communication service instance or select one and click the OK button.
It is also possible to create a new communication service instance first.

+ Name	Admini...	Description	Proxies
test-en-lp [C:\Program Fil...	☑		BCCont1

Edit Properties of Communication Service Instance

General

MC-CmdHandler: Local Host

Operating System: Windows

Host: test-en-lp

Type: Communications Server (Windows)

Installation Path: C:\Program Files\CommunicationService_dummy

Description:

MAC Address:

Control Point

Control Point Network Name: NET0

Control Point Name: CP00

IDBLK / IDNUM

IDBLK: 05D

IDNUM: 0000

Properties of MC-CmdHandler

General

Host: test-en-lp

Listener Port:

Password: *****

State: No or invalid listener port entered.

Operating System: -

Remote Run UserID: -

Directory: -

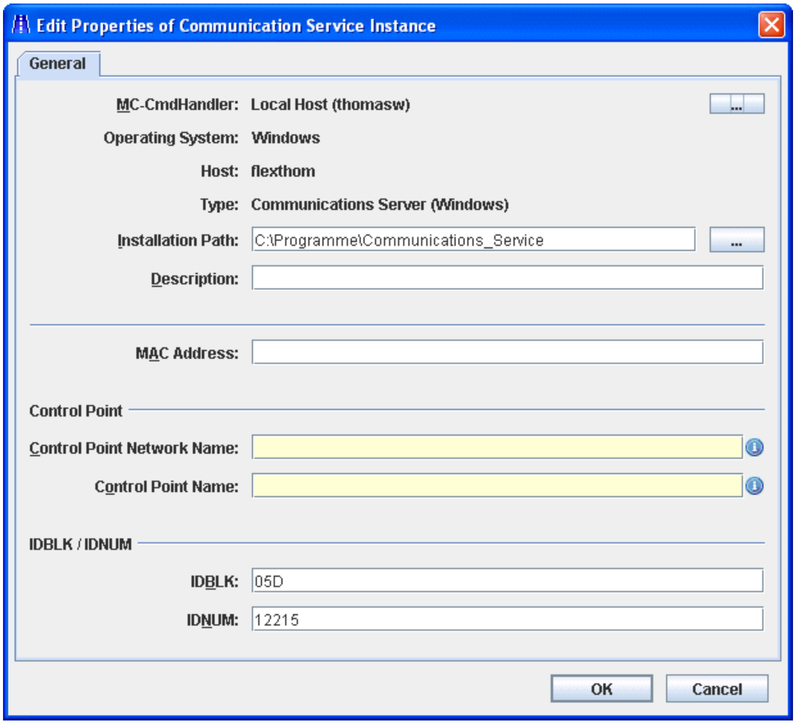
System Time Difference: -

The communication service instances that have already been configured are displayed in **Select a Communication Service Instance**. You can select an instance and click **OK** to assign this instance. You can click **Edit** to modify the properties of a selected instance.

Proceed as follows if you want to configure a new communication service instance:

- Click the **Add** button.
- In **Add Communication Service Instance**, click the **...** button under **Select MC-CmdHandler** to open the dialog **Properties of MC-CmdHandler**.
- In **Properties of MC-CmdHandler**, enter the properties of the MC handler used to administer the components and click **OK**.
- In **Add Communication Service Instance**, enter the installation path of the communication service together with the configuration parameters (as described below) and click **OK**.

Figure 22: Configuring the properties of a communication service



Edit Properties of Communication Service Instance

General

MC-CmdHandler: Local Host (thomasw)

Operating System: Windows

Host: flexthom

Type: Communications Server (Windows)

Installation Path: C:\Programme\Communications_Service

Description:

MAC Address:

Control Point

Control Point Network Name:

Control Point Name:

IDBLK / IDNUM

IDBLK: 05D

IDNUM: 12215

MAC Address

If **LAN** is used as the DLC type for at least one EIS partner then you must enter the MAC address of the computer on which the communication service is running here. For more detailed information, see Section 6.6.2.1, "Adding EIS partners of the type CICS".

Control Point
The control point refers to a unique node in the SNA network. This name is used to identify the instance of the communication service. BeanConnect generates a file containing the necessary VTAM definitions which can then be entered at the z/OS^{TM} mainframe (see Section 7.2.2, "Configuration of VTAM on an IBM mainframe"). The complete control point name must be unique in the network and consists of the following two parts:

- **Control Point Network Name** specifies the network and can be freely defined by the user. It is, however, recommended that you use the EIS network name.
- **Control Point Name** specifies the control point in this network. The name must match the VTAM definition on z/OS.

IDBLK / IDNUM
These values form the XID (node ID) and are incorporated in the PU (physical unit) definition for the VTAM generation.

- **IDBLK** specifies the block ID (IDBLK value) for the CICS generation. It must be specified as a 3-digit hexadecimal number. Alphabetic characters must be entered in uppercase.
- **IDNUM** specifies the Physical Unit ID (IDNUM value) for the CICS generation. It must be specified as a 5-digit hexadecimal number. Alphabetic characters must be entered in uppercase.

ℹ See the Glossary for an explanation of SNA-specific terms.

6.3.3 Modifying the administration password

You can change the administration password with the menu item **Modify Admin Password** in the context menu of the proxy.

This also changes the password of the MC-CmdHandler that is used for proxy administration. For this reason, every proxy should have its own MC-CmdHandler rather than a single MC-CmdHandler being used for a number of different proxies.

You can deposit your password by selecting the option **Use** for the property **Admin User Password** within the property sheet of the proxy so that you don't have to enter it in further sessions (see Section 6.3.1, "General information on the proxy").

When these actions are performed, the MC-CmdHandler must be started on the proxy host. If it is not already running, start it first, see Section 6.10.2.1, "Starting the MC-CmdHandler".

6.3.4 Configuration options in expert mode

If you operate the Management Console in expert mode, the **Edit Properties of Local/Remote Proxy** property sheet contains two more tabs: **Timer Settings** and **Performance Settings**. These tabs provide several additional configuration options which affect the proxy behavior.

In expert mode, you are also able to modify the **Application Program Interface Mode** and the **Container Application Process Title**.

To enable expert mode, select the menu item **Extras – Settings** and from the **General** tab in the **Management Console Settings** property sheet and choose the option **Expert Mode**.

6.3.4.1 Timer Settings

The **Timer Settings** tab contains timers which affect connection surveillance within the proxy.

6.3.4.2 Performance Settings

The **Performance Settings** tab allows you to make settings which affect the performance of the proxy.

The **Proxy Container Mode** defines the behavior regarding asynchronous jobs when the proxy is stopped. By default, asynchronous requests which are not yet started will be deleted when the proxy is stopped (**Performance Enhanced (Non-durable Asynchronous Processing)**). This applies both to asynchronous inbound jobs that have not yet been sent to the application server or which must be redelivered to the application server and to asynchronous outbound jobs that have not yet been sent to the EIS partner. In particular, this includes asynchronous outbound jobs whose start time has not been reached, see also Section 12.5.1, "Duration of asynchronous requests".

If asynchronous requests are to be retained after the proxy has been stopped, you can set the property **Proxy Container Mode** to **Durable Asynchronous Processing**.

In the **Number of Proxy Container Processes** area, you can set the maximum number of processes the proxy can handle at the same time for

- all connections (inbound and outbound),
- asynchronous jobs.

In the **Number of Parallel Connections** area, you can set the maximum number of parallel connections for

- inbound connections via the UPIC protocol,
- inbound connections via the openUTM socket protocol, and
- inbound connections via the RFC1006 protocol.

Additionally, you can set the size of specific storage areas of the proxy container.

Please refer to the online help system for detailed information.

6.3.4.3 Application Program Interface Mode (API Mode)

The **API Mode** field is located on the **General** tab and is preset to the value **Standard**.

If you want to configure a new connection to an EIS partner which uses the XATMI program interface as API then you must set the value **XATMI** for **API Mode**.

You can also set the value **All**. This mode may be necessary during the migration phase, for example if you change from a standard connection to an XATMI connection and vice versa.

6.3.4.4 Container Application Process Title

Only displayed if expert mode is active.
Contains the application process title (APT) of the proxy container. The APT is part of the generation information of an EIS partner and makes it possible for the EIS partner to establish a connection to the proxy container. You can also modify the APT here. If you modify the APT, you must then save the proxy and update the generation (Update Configuration command). This also applies to the proxy's EIS partners (Generate EIS Partner Configuration command). Corresponding todo actions are generated.

6.4 Configuring a BeanConnect proxy cluster

One or more instances of the BeanConnect resource adapter and one or more BeanConnect proxy instances may run in a BeanConnect cluster configuration. All the instances in the cluster are configured identically. Three different scenarios result from this:

- n:1 relation: Several resource adapter instances in the application server and one proxy instance.

  An n:1 relationship is useful in cases where the applications in an application server cluster conducts only a low level of communication with EIS partners.

  Even in the event of an n:1 relation, you must configure a proxy cluster as otherwise the system is operating in multi-resource adapter mode which behaves completely differently, see Section 2.2.3, "Multiple resource adapter mode". The proxy cluster ensures, for example, that any change is performed in all the instances of the resource adapter.
- 1:m relation: One resource adapter instance and several proxy instances
  This option makes sense if communication is almost exclusively inbound.
- n:m relation: Several resource adapter instances and several proxy instances

You always configure all the resource adapter instances in the Management Console if multiple resource adapter instances are running in a cluster in the application server. This is where you configure the cluster for the application server.

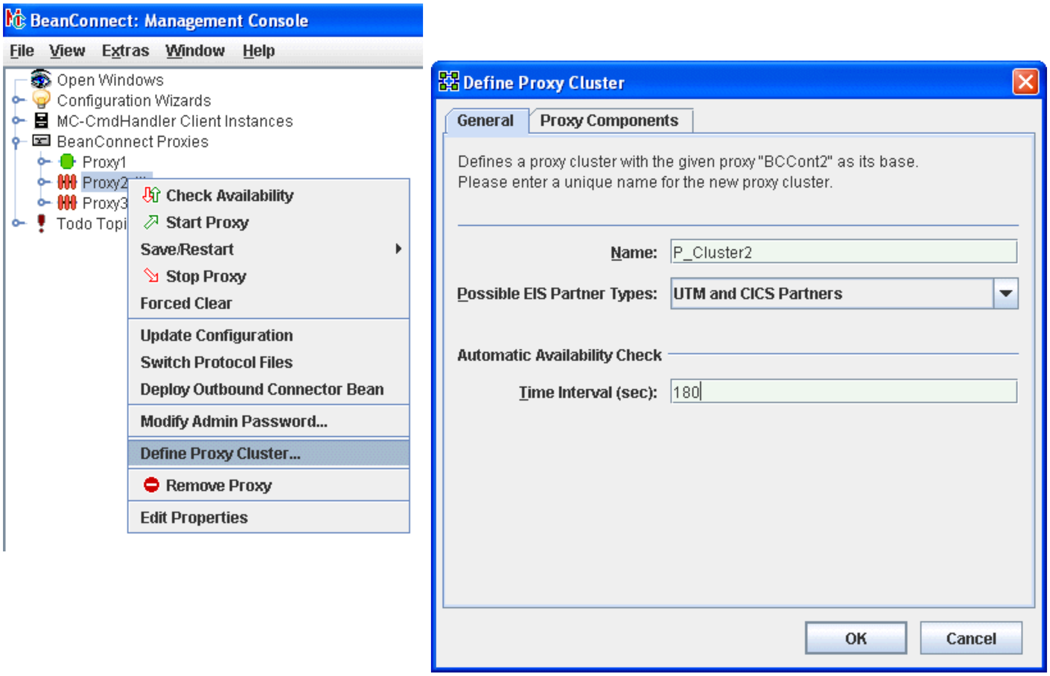
If there is more than one proxy in the proxy cluster then there is one special proxy known as the **master proxy**. This proxy is the (first) point of contact for the Management Console when fetching the configuration data for the cluster. If the currently defined master proxy cannot be administered then the Management Console demands that another proxy, for which administration is possible, is defined as the master proxy. The configuration data is then fetched from this proxy. If any changes are made to the configuration, the Management Console ensures that the administration data of all the proxies in the proxy cluster is modified consistently.

6.4.1 Generating a proxy cluster

Before you can generate a proxy cluster, you must already have configured at least one proxy, see Section 6.3, "Configuring the BeanConnect proxy".

To generate a new proxy cluster, select the required proxy under **BeanConnect Proxies** and then choose the **Define Proxy Cluster...** command in the context menu. The proxy must not be started when you do this.

Figure 23: Generating a new proxy cluster



Defines a proxy cluster with the given proxy "BCCont2" as its base.
Please enter a unique name for the new proxy cluster.

Name: P_Cluster2

Possible EIS Partner Types: UTM and CICS Partners

Time Interval (sec): 180

Enter the name of the new proxy cluster in the **Name** field in the **Define Proxy Cluster** dialog box. The values for the selected proxy are displayed in all the other fields. You can change these values if required.

When you exit the dialog with **OK**, a new node named **BeanConnect Proxy Clusters** is generated in the navigation tree. The cluster is entered here with the name defined above. At the same time, the proxy is removed from the list of **BeanConnect Proxies**. By default, the first proxy included in the proxy cluster becomes the master proxy.

Displaying the proxies in a cluster

To display the proxies in a proxy cluster, you can either click the proxy cluster node or choose the **Show Cluster Proxies** command in the context menu. A list of all the proxies is now displayed. The individual columns describe the properties of the proxies such as their names and address data. For further details, consult the online Help.

6.4.2 Adding a proxy to the proxy cluster

You can add further proxies to the proxy cluster. A maximum of 32 proxies per cluster is permitted. The corresponding proxy must be installed and the general proxy data must have been configured using the Management Console, see Section 6.3.1, "General information on the proxy".

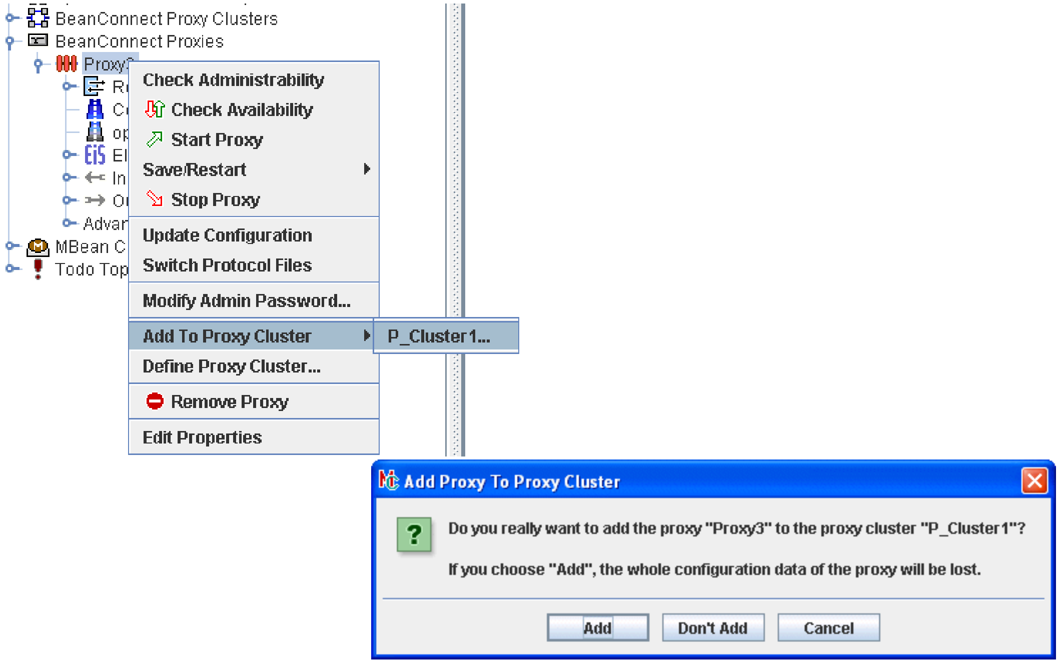
To add another proxy to the cluster:

- Select the required proxy and choose the **Add to Proxy Cluster** command in the context menu. This displays all the configured proxy clusters to which the selected proxy can be added.
- Click to select the required cluster and confirm the query with **Add**. The proxy is added to this cluster and is removed from the list of proxies.

⚠ **Caution!**
When you add a proxy to a proxy cluster, the configuration of the proxy is lost since it is overwritten by the configuration of the proxy cluster (the proxy cluster is synchronized).

If you remove a proxy from an n:1 or n:m cluster then no subsequent inbound communication is possible in the remote proxy. This is due to the fact that the proxy is started in multi-RA mode because it possesses more than one resource adapter but no valid resource adapter is assigned to any message endpoint.

Figure 24: Adding a proxy to a proxy cluster



Do you really want to add the proxy "Proxy3" to the proxy cluster "P_Cluster1"?

If you choose "Add", the whole configuration data of the proxy will be lost.

6.4.3 Removing a proxy from a cluster / removing a proxy cluster

You can remove individual proxies from the proxy cluster or remove the proxy cluster itself.

- To remove a proxy from the proxy cluster, select the required proxy on the list of cluster proxies and choose the **Remove Proxy From Cluster** command from the context menu. After you confirm the prompt, the proxy is removed from the cluster and listed in the **BeanConnect Proxies** area again.
- You remove a proxy cluster by choosing the **Remove Proxy Cluster** command from the context menu of the proxy cluster. After this action, all the proxies present in the proxy cluster are listed in the **BeanConnect Proxies** area again. The proxies are therefore not uninstalled but are simply removed from the cluster.

Caution!
When you remove a proxy from a proxy cluster or remove the proxy cluster itself, the proxy retains the configuration that it had in the cluster. In other words, it does **not** return to the state it had before it was added to the proxy cluster.

6.5 Configuring the BeanConnect resource adapter

The configuration data for the BeanConnect resource adapter is described in the resource adapter's deployment descriptor `ra.xml`. This deployment descriptor is located in the BeanConnect RAR archive.

For information on how to adapt the `ra.xml` file, see Section 4.2, "Configuring general properties for the resource adapter".

Alternatively, you can edit the `ra.xml` file via the BeanConnect Management Console. To do this, you must configure the BeanConnect resource adapter in the BeanConnect Management Console.

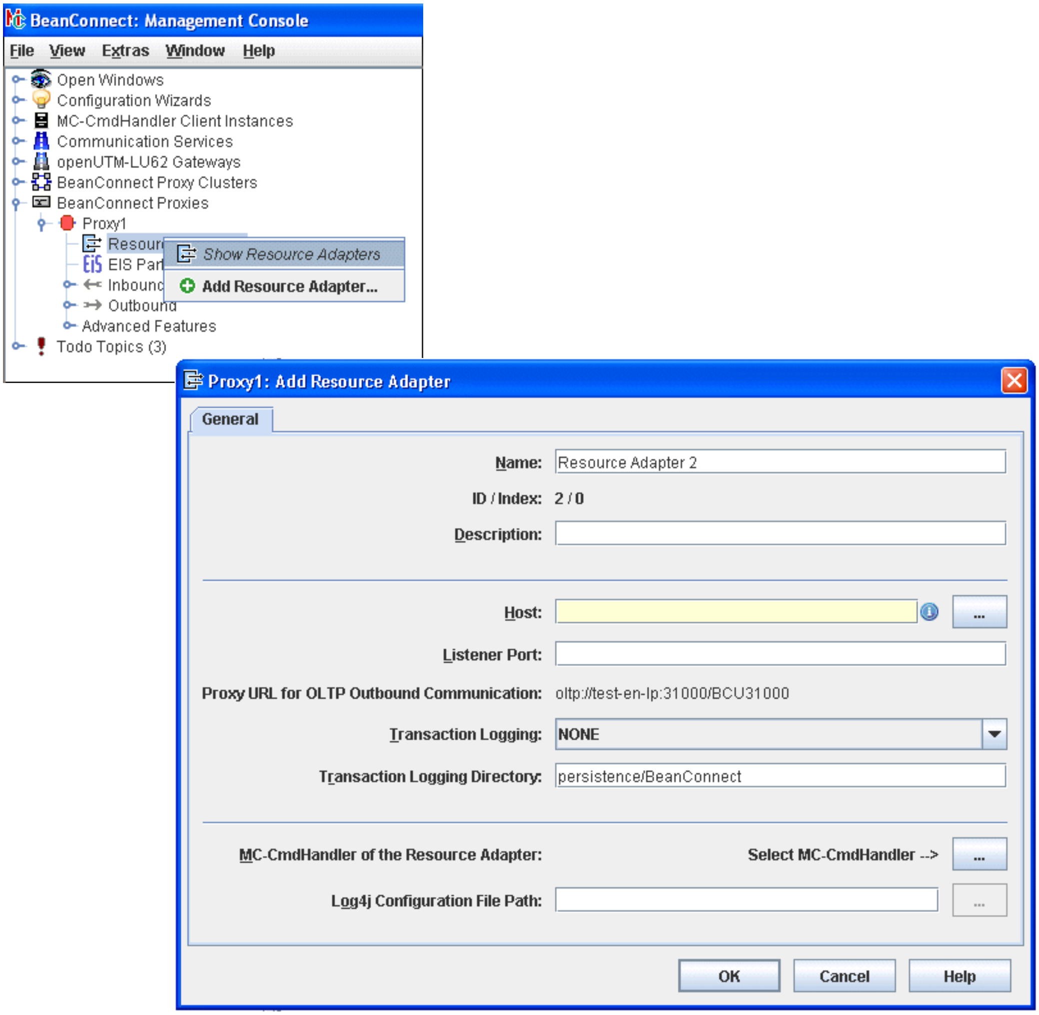
The configuration dialog differs depending on whether or not the proxy belongs to a cluster.

6.5.1 Adding a resource adapter (no cluster operation)

To add a resource adapter, open the context menu for the **Resource Adapter** node in the navigation tree's proxy subtree and choose **Add Resource Adapter**... Alternatively, you can click the required node and then run the **Add** command in the resource adapter list. The proxy must not be running when it is added. You may therefore need to stop it, for example using the **Stop Proxy** command in the proxy's context menu.

Enter the parameters for the resource adapter in the **General** tab:

Figure 25: Adding a resource adapter without cluster operation



Name: Resource Adapter 2

ID / Index: 2 / 0

Description:

Host:

Listener Port:

Proxy URL for OLTP Outbound Communication: oltp://test-en-lp:31000/BCU31000

Transaction Logging: NONE

Transaction Logging Directory: persistence/BeanConnect

MC-CmdHandler of the Resource Adapter: Select MC-CmdHandler -->

Log4j Configuration File Path:

Name

Freely definable name of the resource adapter. This name is only relevant locally in the Management Console.

ID / Index

ID is the ID of the resource adapter assigned by the Management Console. This is a number between 1 and 256.
Index is the index of the resource adapter which is assigned by the Management Console. The index is only relevant in multiple resource adapter mode This property corresponds to the `resourceAdapterIndex` property in the resource adapter's deployment descriptor `ra.xml`.

Description

You can enter a freely definable description of the resource adapter here.

Host

Name or IPv4 address of the computer running the application server on which the resource adapter is deployed.

Listener Port

Number of the listener port of the resource adapter for inbound communication. This property corresponds to the `inboundListenerPort` property in the resource adapter's deployment descriptor `ra.xml`. If you enter 0 here (or leave the field blank), inbound communication is not possible.

Proxy URL of OLTP Outbound Communication

URL used for outbound communication in the application server. This URL is defined when the proxy is configured and cannot be changed. This property corresponds to the `proxyURL` property in the resource adapter's deployment descriptor `ra.xml`.

Transaction Logging

Specifies whether persistent transaction logs are written in the resource adapter in the case of outbound communication (**FILE**) or not (**NONE**). This property corresponds to the `transactionLogging` property in the resource adapter's deployment descriptor `ra.xml`. If transaction logging is configured then the resource adapter writes a separate log file for each transaction. The file name consists of the prefix `tx.` and a number.

Transaction Logging Directory

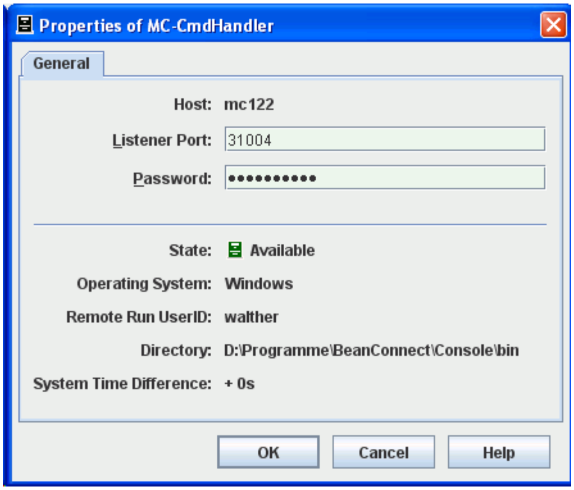
Directory in which the transaction logs are created in the resource adapter. This property corresponds to the `transactionLogDir` property in the resource adapter's deployment descriptor `ra.xml`.

MC-CmdHandler of the Resource Adapter

This allows you to define an MC-CmdHandler that is available on the resource adapter computer and that can be used to administer the resource adapter via the Management Console. To do this, proceed as follows:

- Click the **...** button in **Select MC-CmdHandler** to display an additional dialog box **Properties of MC-CmdHandler** in which you can enter the data for the MC-CmdHandler (listener port, password):

Figure 26: Configuring the properties of an MC-CmdHandler



Host: mc122
Listener Port: 31004
Password: ●●●●●●●●●●
State: Available
Operating System: Windows
Remote Run UserID: walther
Directory: D:\Programme\BeanConnect\Console\bin
System Time Difference: + 0s

If the BeanConnect-RAR archive is also located on this computer, then you can use this MC-CmdHandler to edit the entries in the resource adapter's deployment descriptor `ra.xml` or update these entries in line with the entered parameters. You can do this using the commands **Edit ra.xml of BeanConnect Resource Adapter RAR** or **Update ra.xml of BeanConnect Resource Adapter RAR** in the resource adapter's context menu, see Section 6.5.3, "Resource adapter configuration file".

You can also use this MC-CmdHandler to modify the Log4j diagnostic settings.

- You can enter the fully qualified path name of the Log4j configuration file in **Log4j Configuration File Path** (optional).

The Log4j configuration file is always located on the computer on which the application server runs. This does not have to be the computer on which the BeanConnect resource adapter is located since the resource adapter cannot be uploaded to the application server computer before deployment.

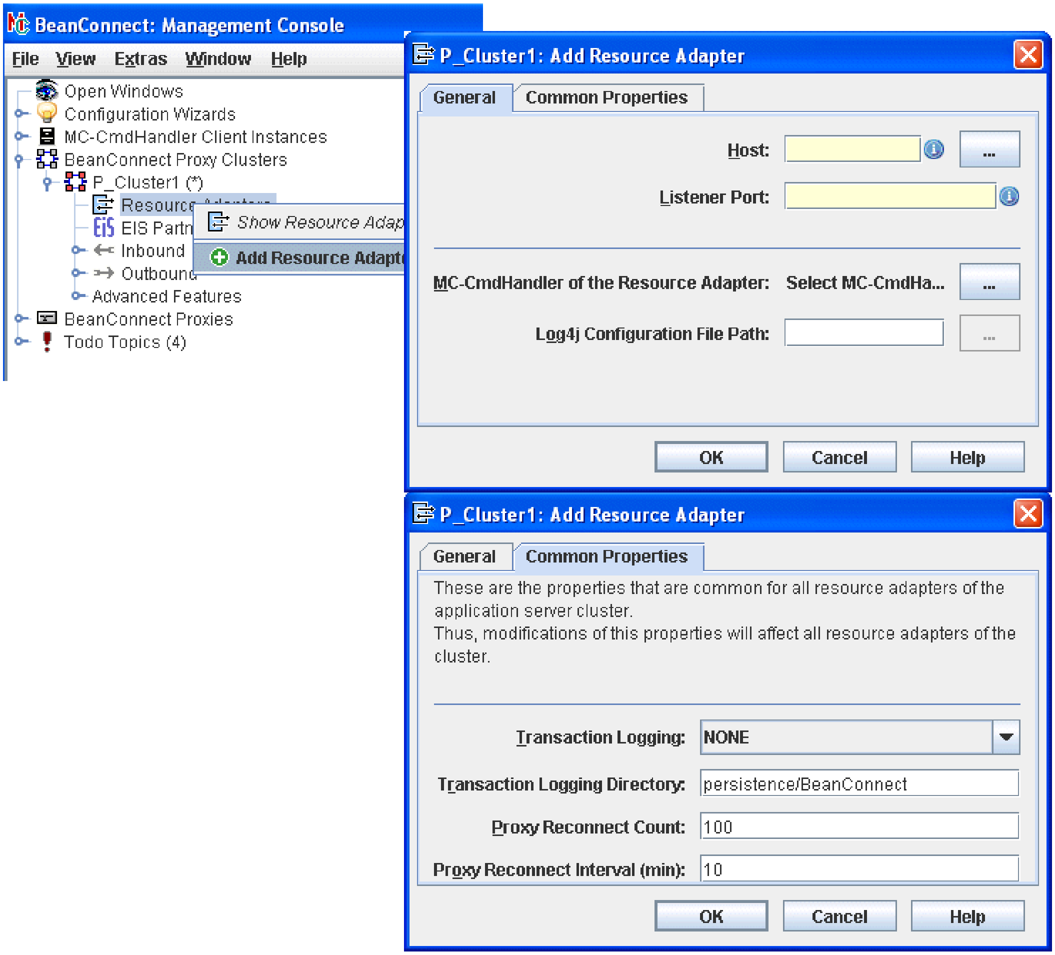
If you want to change the resource adapter's logging configuration then an MC-CmdHandler must be running on the computer on which the application server runs.

6.5.2 Adding a resource adapter in cluster operation

To add a resource adapter in cluster operation, open the context menu for the **Resource Adapter** node in the navigation tree's cluster subtree and choose **Add Resource Adapter**... Alternatively, you can click the required node and then run the **Add** command in the resource adapter list.

Enter the parameters for the resource adapter in the two tabs **General** and **Common Properties**. The parameters specified in **Common Properties** are the same for all the resource adapters in a cluster. Any change to these properties therefore affects all the resource adapters defined in the cluster, whereas the parameters specified in **General** are specific to each resource adapter:

Figure 27: Adding a resource adapter in cluster operation



P_Cluster1: Add Resource Adapter

General | Common Properties

Host:

Listener Port:

MC-CmdHandler of the Resource Adapter: Select MC-CmdHa...

Log4j Configuration File Path:

OK Cancel Help

P_Cluster1: Add Resource Adapter

General | Common Properties

These are the properties that are common for all resource adapters of the application server cluster.
Thus, modifications of this properties will affect all resource adapters of the cluster.

Transaction Logging: NONE

Transaction Logging Directory: persistence/BeanConnect

Proxy Reconnect Count: 100

Proxy Reconnect Interval (min): 10

OK Cancel Help

Host (General)

Name or IPv4 address of the computer running the application server on which the resource adapter is deployed. The host specified here must match the value of `<host>` in a `<host:port>` entry of the `resourceAdapterAddresses` property of the deployment descriptor `ra.xml` for the resource adapter.

Listener Port (General)

Number of the listener port of the resource adapter for inbound communication. The host specified in **Host** and the listener port must match a `<host:port>` entry in the `resourceAdapterAddresses` property of the deployment descriptor `ra.xml` for the resource adapter. If no port is specified there, the specification must match the value in the `inboundListenerPort` property.

Within a cluster, the Management Console does not permit a value of `0` or no value to be entered. A port between `1025` and `65535` must be specified even if no inbound communication is planned.

MC-CmdHandler of the Resource Adapter (General)

These fields allow you to define an MC-CmdHandler that is available on the resource adapter computer and that can be adapted via the Management Console using the resource adapter's deployment descriptor `ra.xml`.

- Click the **...** button in **Select MC-CmdHandler** to display an additional dialog box in which you can enter the data for the MC-CmdHandler (listener port, password):

  If the BeanConnect-RAR archive is also located on this computer, then you can use this MC-CmdHandler to edit the entries in the resource adapter's deployment descriptor `ra.xml` or update these entries in line with the entered parameters. You can do this using the commands **Edit ra.xml of BeanConnect Resource Adapter RAR** or **Update ra.xml of BeanConnect Resource Adapter RAR** in the resource adapter's context menu, see Section 6.5.3, "Resource adapter configuration file".
  You can also use this MC-CmdHandler to modify the Log4j diagnostic settings.

- You can enter the fully qualified path name of the Log4j configuration file in **Log4j Configuration File Path** (optional).

  The Log4j configuration file is always located on the computer on which the application server runs. This does not have to be the computer on which the BeanConnect resource adapter is located since the resource adapter cannot be uploaded to the application server computer before deployment.

  If you want to change the resource adapter's logging configuration then an MC-CmdHandler must be running on the computer on which the application server runs.

Transaction Logging (Common Properties)

Specifies whether persistent transaction logs are written in the resource adapter in the case of outbound communication (**FILE**) or not (**NONE**). This property corresponds to the `transactionLogging` property in the resource adapter's deployment descriptor `ra.xml`. If transaction logging is configured then the resource adapter writes a separate log file for each transaction. The file name consists of the prefix `tx.` and a number.

Transaction Logging Directory (Common Properties)

Directory in which transaction logging is stored in the resource adapter. This property corresponds to the `transactionLogDir` property in the resource adapter's deployment descriptor `ra.xml`.

Proxy Reconnect Count (Common Properties)

Specifies the number of connection requests (calls to `getConnection()`) after which the assignment between the resource adapter instance and the proxy application must be made again. This property controls the usage-driven reassignment of a resource adapter instance to a proxy application. It corresponds to the `proxyReconnectCount` property in the resource adapter's deployment descriptor `ra.xml`.

The default value is 100.

If a cluster is operated with more resource adapter instances than proxy instances then a larger value or the value 0 should be entered for this parameter. The value 0 deactivates the reconnect counter.

Proxy Reconnect Interval (min) (Common Properties)

Specifies the time in minutes after which the assignment between the resource adapter instance and the proxy application must be made again. This property controls the time-driven reassignment of a resource adapter instance to a proxy application. It corresponds to the `proxyReconnectInterval` property of the resource adapter's deployment descriptor `ra.xml`.

The default value is 10 minutes.

If a cluster is operated with more resource adapter instances than proxy instances then a larger value or the value 0 should be entered for this parameter. The value 0 deactivates the reconnect timer.

6.5.3 Resource adapter configuration file

The resource adapter's general configuration data is defined in the file `ra.xml`, see Section 4.2, "Configuring general properties for the resource adapter". `ra.xml` is located in the BeanConnect-RAR archive. The Management Console can access this file and modify the configuration properties if one of the following conditions is satisfied:

- The BeanConnect RAR archive is located on the computer on which the **Management Console** runs. In this case, it must be located under the user ID under which the Management Console runs or the file access permissions must be set accordingly.
- The BeanConnect RAR archive is located on a computer on which an **MC-CmdHandler** runs. In this case, it must be located under the user ID under which the MC-CmdHandler runs or the file access permissions must be set accordingly.

Updating the configuration file via the Management Console

To update the configuration file `ra.xml`, choose the command **Update ra.xml of BeanConnect Resource Adapter RAR** from the resource adapter's context menu. In a proxy cluster, you must open the context menu in the **Resource Adapters** node and not in the resource adapter itself.

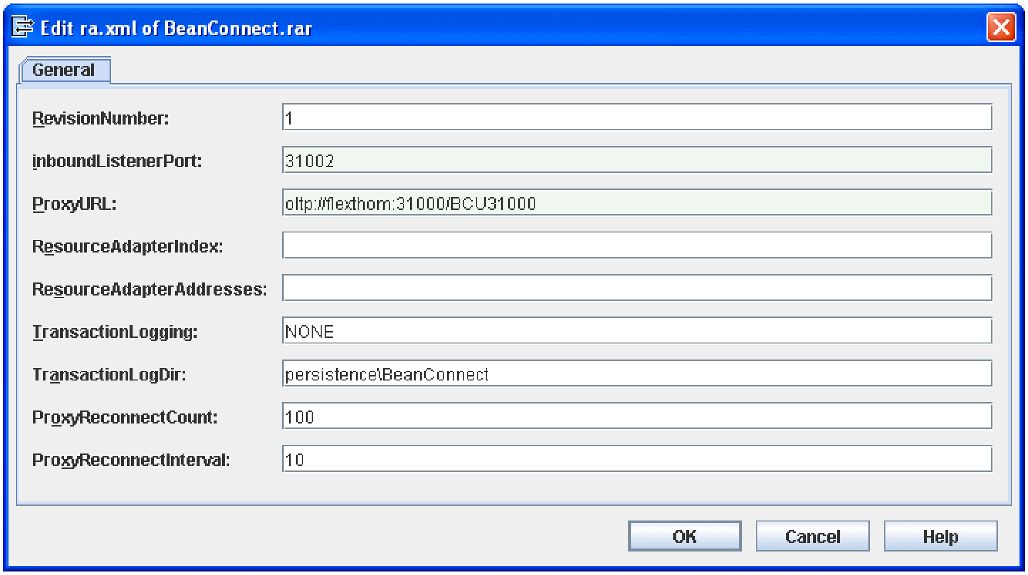
This overwrites the values in `ra.xml` with the values specified in the Management Console.

Editing the configuration file via the Management Console

To edit the configuration file `ra.xml`, choose the command **Edit ra.xml of BeanConnect Resource Adapter RAR** from the resource adapter's context menu. In a proxy cluster, you must open the context menu in the **Resource Adapters** node and not in the resource adapter itself.

The dialog box **Edit ra.xml of BeanConnect Resource Adapter RAR** is now opened. Here, you can modify all the properties of `ra.xml`, see also Section 4.2.1, "Defining general properties in ra.xml" and the online Help.

Figure 28: Editing the configuration file ra.xml



Edit ra.xml of BeanConnect.rar

General

RevisionNumber: 1

inboundListenerPort: 31002

ProxyURL: oltp://flexthom:31000/BCU31000

ResourceAdapterIndex:

ResourceAdapterAddresses:

TransactionLogging: NONE

TransactionLogDir: persistence\BeanConnect

ProxyReconnectCount: 100

ProxyReconnectInterval: 10

After an update or following direct editing, the modified values do not take effect until the BeanConnect RAR archive has been deployed (see Section 4.2.3.1, "Deploying the resource adapter".

i Please note that you must enter the required changes in the Management Console of the corresponding resource adapter(s) **before** using the **Update ra.xml...** command.

6.6 Configuring the EIS partners

A proxy or proxy cluster can communicate with multiple EIS partners, i.e. with multiple openUTM/CICS applications. When you configure the proxy/proxy cluster, you specify the type of EIS partners it is to communicate with (only openUTM, only CICS, or both).

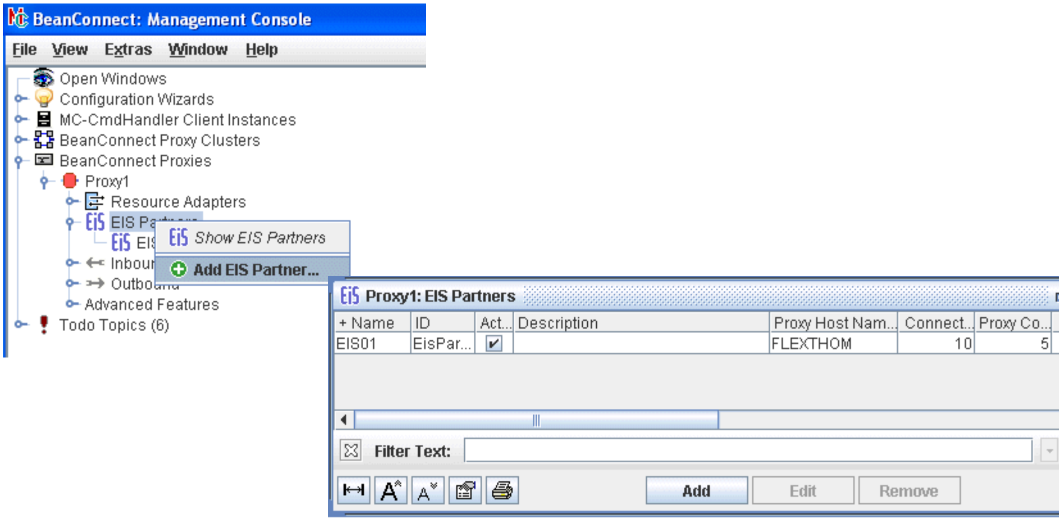
To allow an EIS partner to be managed, the partner must be added to the Management Console's configuration data. As far as this operation is concerned, there are only slight differences between proxies and proxy clusters.

Each partner application known by the Management Console is represented by an EIS partner object in the navigation tree's proxy subtree beneath the **EIS Partners** node.

i Here, you only have to configure EIS partners that are either of type openUTM and are communicated with via the OSI-TP protocol or are of type CICS.

Click on the **EIS Partners** node to display a list of the managed EIS partners of the proxy.

Figure 29: Displaying and configuring EIS partners



+ Name	ID	Act...	Description	Proxy Host Nam...	Connect...	Proxy Co...
EIS01	EisPar...	☑		FLEXTHOM	10	5

6.6.1 Configuring EIS partners of type openUTM

Before you can configure an EIS partner of type openUTM, the proxy or proxy cluster must be configured accordingly in the **General** tab, see Figure 18, "General information on a proxy" on page 174.

6.6.1.1 Adding EIS partners of the type UTM

To add a new EIS partner:

1. Click the **Add** button beneath the list of EIS partners or choose **Add EIS Partner** from the context menu of an existing EIS partner object or the **EIS Partner** node.

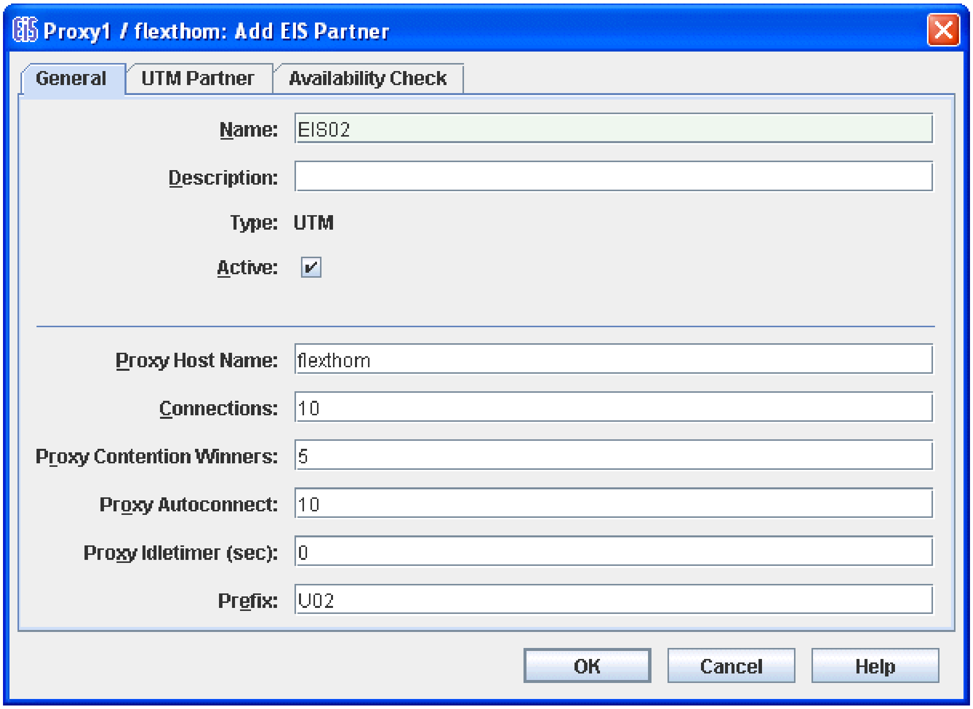
   If the proxy/proxy cluster is configured for the two partner types (UTM and CICS), then the dialog box **Choose EIS Partner Type** is displayed during this operation. Here you must choose **UTM application**.

2. Define the properties for the EIS partner. The property sheet is opened automatically and contains the tabs **General**, **UTM Partner** and **Availability Check**.

The following properties are requested:

General tab (openUTM Partners)

Figure 30: General properties of an EIS partner of Type openUTM



Name: EIS02

Description:

Type: UTM

Active: ☑

Proxy Host Name: flexthom

Connections: 10

Proxy Contention Winners: 5

Proxy Autoconnect: 10

Proxy Idletimer (sec): 0

Prefix: U02

Name / Description

Name is the name of the EIS partner. This must be unique to the proxy. Additionally, you can enter a **Description** for the EIS partner. For a created EIS partner, the EIS **ID** is shown beneath the name. (cannot be changed).

Type

Displays the type of EIS partner: here **UTM**, cannot be changed.

Active

The option **Active** controls whether the EIS partner definition is active or not. Only active EIS partners will be included in the configuration of the proxy and the EIS partner (**Update Configuration** command in the proxy node or proxy cluster node).

Proxy Host Name

Computer name under which the proxy host is known in the openUTM partner application.

In the case of an EIS partner in the proxy cluster, there is an entry field for each cluster proxy in which you must specify the computer name under which the corresponding proxy is known to the EIS partner.

By default, this field is set to the computer name entered under **General** for **Host** in the **Edit Properties** command. If an IP address was specified for the proxy host then this must be replaced by the name

Connections

Specifies the maximum number of concurrent connections allowed between proxy and the EIS partner.

Proxy Contention Winners

Specifies the number of connections for which the proxy should be contention winner. The number you should specify depends on the favored communication type (outbound or inbound communication).

The following applies in principle:

- Outbound communication: the proxy should be the contention winner.
- Inbound communication: the EIS partner should be the contention winner.

Proxy Autoconnect

Defines the number of connections to be established when starting the proxy.

Proxy Idletimer (sec)

Specifies the time in seconds after which the BeanConnect proxy container is to clear down the connection to the EIS partner if no communication has taken place over the connection during that time.

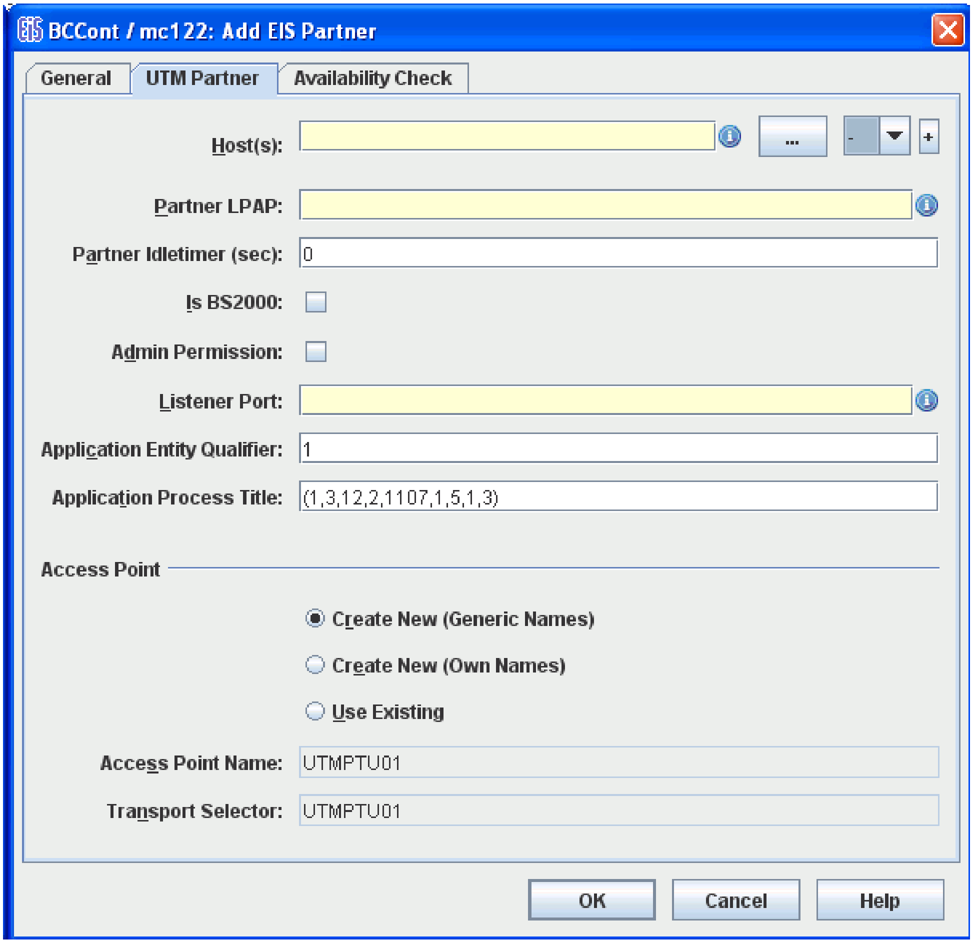
Possible values: 0 (default) through 32767.
0 means that the timer is not used.

Prefix

The prefix is included as a component in names which are used in configuration statements. This prefix must be unique for all proxies running on the same host. The prefix must comprise exactly three characters (uppercase letters or digits). The first character must be an uppercase letter.

UTM Partner tab

Figure 31: Properties of the openUTM partner



Host(s):

Partner LPAP:

Partner Idletimer (sec): 0

Is BS2000: □

Admin Permission: □

Listener Port:

Application Entity Qualifier: 1

Application Process Title: (1,3,12,2,1107,1,5,1,3)

Access Point

● Create New (Generic Names)

○ Create New (Own Names)

○ Use Existing

Access Point Name: UTMPTU01

Transport Selector: UTMPTU01

If expert mode is enabled then additional fields are displayed. These are described later on, see "UTM Partners - Additional Fields in Expert Mode" on page 206.

Host(s)

The name of the computer on which the openUTM partner application is located, from 1 to a maximum of 8 characters in length.

If the partner application is an openUTM cluster application, enter the names of the computers for all the cluster nodes here.

Alternatively, you can make your entry in the **Host(s)** by clicking the **...** button to the right of the text field and specifying the host address.

If the partner is an openUTM cluster, specify the cluster node index of the node application in the drop-down list to the right of the button.

If the partner is a stand-alone openUTM application, select "-" in the drop-down list.

The node index of the cluster node is determined by the sequence of CLUSTER-NODE statements in the KDCDEF input of the cluster partner application. The node that is defined by the first CLUSTER-NODE statement has the index 1, the node corresponding to the second CLUSTER-NODE statement has the index 2 and so on.

If the partner is an openUTM cluster, click the button **+** (plus) to add a further cluster node. An additional row with the same structure as the first row is now displayed. Click the button **-** (minus) to remove a cluster node from the configuration again. You can specify a maximum of 32 cluster nodes. You cannot remove the last remaining cluster node.

Partner LPAP

Specifies the LPAP name under which the openUTM partner application addresses the BeanConnect proxy and therefore also the application server during inbound communication, 1 to a maximum of 8 characters in length. From this name, the Management Console generates an OSI-LPAP statement for generating the openUTM partner application.

Partner Idletimer (sec)

Specifies the time in seconds after which the EIS partner is to clear down the connection to the BeanConnect proxy container if no communication has taken place over the connection during that time.

It is recommended that you choose a value for **Partner Idletimer** that is smaller than the one specified for **Proxy Idletimer**.
0 means that the timer is not used.
Possible values: 0 (default) through 32767.

Is BS2000

This option specifies whether or not the openUTM partner application is located on a BS2000 system. If this option is activated then the Management Console generates KDCDEF and BCMAP statements for a BS2000 system.

Admin Permission

This option specifies whether the proxy, and, implicitly, the application server also, are to possess administrative permissions in the openUTM partner application during outbound communication.

Listener Port

Defines the port at which the openUTM partner application waits for requests to establish a connection. Permitted values: 102 and 1025 through 32767

If you have not enabled **Is BS2000** and you select the **Use Existing** option for **Access Point** , you must enter the value generated in the openUTM partner application here (ACCESS-POINT statement, LISTENER-PORT operand).

Application Entity Qualifier

The **Application Entity Qualifier** is an address component for the openUTM partner application's access point.

If you select the **Use Existing** option for **Access Point** , you must enter the value generated in the openUTM partner application here (ACCESS-POINT statement, APPLICATION-ENTITY-QUALIFIER operand).

Application Process Title

The **Application Process Title** is an address component for the openUTM partner application's UTMD statement.

If you select the **Use Existing** option for **Access Point** , you must enter the value generated in the openUTM partner application (UTMD statement, APPLICATION-PROCESS-TITLE operand).

If the EIS partner is a UTM cluster application and the specified application process title (APT) consists of fewer than 10 elements then the Management Console adds the node index to the APT in order to ensure that the APTs are unique.

Access Point

Defines the properties of the access point used to address the openUTM partner application during outbound communication. The access point is generated in the openUTM partner application using the KDCDEF statement ACCESS-POINT.

The three options **Create New (Generic Names)**, **Create New (Own Names)** and **Use Existing** are used to determine whether the Management Console generates KDCDEF statements for the access point, or whether an access point that already exists in the openUTM partner application is to be used.

- **Create New (Generic Names)**
  If you select this option, the Management Console generates an ACCESS-POINT statement with generic values. The Management Console enters these values in the **Access Point Name**, **Transport Selector** and **Transport Selector Format** fields. They cannot be changed.
- **Create New (Own Names)**
  If you select this option, the Management Console generates an ACCESS-POINT statement with specific values. The Management Console enters these values in the fields **Access Point Name**, **Transport Selector** and **Transport Selector Format**. You can modify the values.

- **Use Existing**
  If you select this option, the Management Console does not generate an ACCESS-POINT statement. This option is designed for circumstances in which you want to use an existing access point in the openUTM partner application. You must enter the values for this access point in the **Access Point Name**, **Transport Selector** and **Transport Selector Format** fields. Furthermore, the values you enter in the **Application Entity Qualifier** and **Listener Port** fields must have been generated in the openUTM partner application.

- **Access Point Name**
  Name of the access point in the openUTM partner application.

  If you select **Create New (Generic Names)**, this contains the generated name (cannot be changed).

  If you select **Create New (Own Names)**, you must enter a freely-definable name here (1 through 8 characters in length).

  If you select **Use Existing**, you must enter the name generated in the ACCESS-POINT statement of the openUTM partner application.

- **Transport Selector**

  Transport selector of the access point in the openUTM partner application.

  If you select **Create New (Generic Names)**, this contains the generated name (cannot be changed).

  If you select **Use Existing**, you must enter the name generated with T-SEL= in the ACCESS-POINT statement of the openUTM partner application.

  If you select **Create New (Own Names)**, you must enter a freely-definable name here. This can be 1 through 8 characters in length, the first character must be an uppercase alphabetic character, otherwise alphabetic characters, numeric characters and the special characters #, $ and @ are permitted.

UTM Partners - Additional Fields in Expert Mode

If expert mode is active then the following additional fields are also displayed:

- **Application Program Interface Mode of EIS Partner**

  This field is only relevant for partners that use the XATMI interface, see Section 6.6.3, "Configuring EIS partners of type XATMI". In the case of openUTM partner applications that use the KDCS interface, the value **Standard** must always be set for **API Mode**.

- **Transport Selector Format**
  Format used to encode the transport selector.

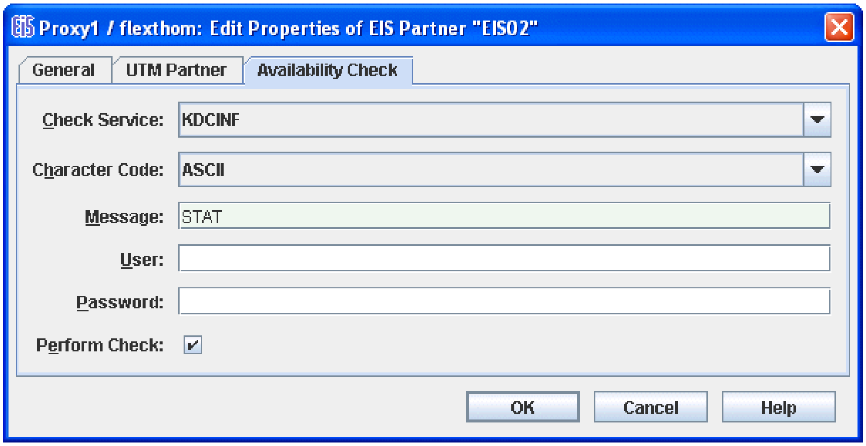
  If you select **Create New (Generic Names)**, this contains the generated value (cannot be changed).

  If you select **Create New (Own Names)**, you can choose between **ASCII**, **EBCDIC** and **TRANSDATA** (default). If the EIS partner is running under BS2000, **TRANSDATA** must be used as the transport selector format.

  If you select **Use Existing**, you must enter the value generated with TSEL-FORMAT= in the ACCESS-POINT statement of the openUTM partner application.

Availability Check tab (openUTM partners)

Figure 32: Properties of the EIS partner for the availability check



Proxy1 / flexthom: Edit Properties of EIS Partner "EIS02"

Check Service: KDCINF
Character Code: ASCII
Message: STAT
User:
Password:
Perform Check: ☑

This dialog box allows you to select a dialog service in the partner application which is called when checking the availability of an EIS partner. When this is done, the message defined in **Message** is passed to the specified service in the EIS partner. As soon as the response is received, the EIS partner is flagged as available. The response is output in the Management Console.

You cannot complete this dialog box properly unless you have defined at least one outbound service for the EIS partner, see Section 6.7.1, "Configuring outbound services".

Check Service

This lists all the outbound services defined for this partner. Select a suitable service from the list. The service is always of type **Dialog**.

Character Code

Character set used by the EIS partner:
Possible values: **ASCII** or **EBCDIC**.

Message

Message sent to the EIS partner when checking availability. The maximum length of the message is 80 characters.

User

User ID if the service is to be called under a specific user ID.

Password

Password for the user ID if a password is required.

Perform Check

This option allows you to temporarily disable the availability check for this EIS partner without losing the settings, such as the service to be called.

If the option is enabled, the availability check is performed with the specified parameters.

If you disable this option, the availability check for this EIS partner is disabled until the option is enabled again. Both automatic availability checking and the availability check performed by issuing a command in the proxy or proxy cluster's context menu are disabled. If the EIS partner is checked "manually" (using its context menu), the check is performed irrespective of this setting.

6.6.1.2 Configuration files for EIS partners of type openUTM

The Management Console creates configuration fragments for the EIS partners. The next time you save the proxy configuration (see Section 6.9, "Saving and activating the configuration of the BeanConnect proxy"), the Management Console generates all the configuration statements for all the EIS partners. Before this is possible, you must have enabled the **Active** option on the **General** tab.

Files with KDCDEF statements are created. In the case of openUTM partner applications on BS2000 systems, files containing BCMAP statements are also generated.

The configuration files are located in the directory `<MC_Home>/genfiles` under the following names:

- Single proxy:

  `ProxyID.<p-id>.EISPartnerID.<e-id>.UTM.txt` (KDCDEF statements)

  `ProxyID.<p-id>.EISPartnerID.<e-id>.BCMAP.txt` (BCMAP statements)

- Proxy cluster:

  `ClusterID.<c-id>.EISPartnerID.<e-id>.UTM.txt` (KDCDEF statements)

  `ClusterID.<c-id>.EISPartnerID.<e-id>.BCMAP.txt` (BCMAP statements)

`<MC_Home>` is the directory under which the Management Console is installed. `<p-id>`, `<e-id>` and `<c-id>` designate the IDs for the proxy, EIS partner and proxy cluster assigned by the Management Console.

The file containing the BCMAP statements is only generated for openUTM partners in BS2000 systems (the **Is BS2000** option is enabled).

For further details on the configuration tasks in EIS partners, see Chapter 7, "Adapting the configuration in the EIS partner".

6.6.2 Configuring EIS partners of type CICS

Before you can configure an EIS partner of type CICS, the proxy or proxy cluster must be configured accordingly in the **General** tab, see, for example, "Possible EIS Partner Types" on page 175.

Generally speaking, an EIS partner in the proxy cluster is configured in exactly the same way as an individual proxy except for one small difference relating to the configuration of the communication service.

6.6.2.1 Adding EIS partners of the type CICS

To add a new EIS partner:

1. Click the **Add** button beneath the list of EIS partners or choose **Add EIS Partner** from the context menu of an existing EIS partner object or the **EIS Partners** node.

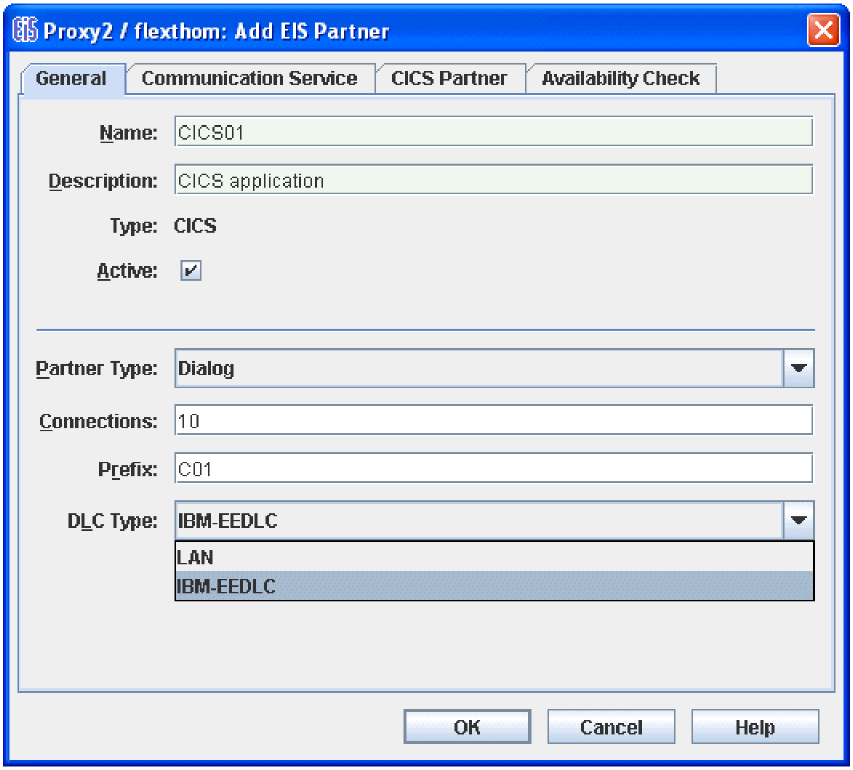
   If the proxy/proxy cluster has been configured for the two partner types (UTM and CICS), then the dialog box **Choose EIS Partner Type** is also displayed. Here, you should select **CICS application**.

2. Define the properties for the EIS partner. The property sheet is opened automatically and contains the tabs **General**, **Communication Service**, **CICS Partner** and **Availability Check**.

The following properties are requested:

General tab (CICS partners)

Figure 33: General properties of an EIS partner of type CICS



Name: CICS01

Description: CICS application

Type: CICS

Active: ☑

Partner Type: Dialog

Connections: 10

Prefix: C01

DLC Type: IBM-EEDLC

Name / Description

The **Name** of the EIS partner. This must be unique to the proxy. Additionally, you can enter a **Description** for the EIS partner. For a created EIS partner, the EIS **ID** is shown in the ID field beneath the name (cannot be changed).

Type

Displays the type of EIS partner: here **CICS**, cannot be changed.

Active

The option **Active** controls whether the EIS partner definition is active or not. Only active EIS partners will be included in the configuration of the proxy and the EIS partner. Inactive EIS partners are struck through to identify them in the navigation tree.

Partner Type

Specifies the type of outbound communication performed with the CICS partner:

- **Dialog**: Communication over dialog services.
- **Asynchronous**: Communication over asynchronous services.

In BeanConnect, a CICS partner may be configured either as a dialog partner only or as an asynchronous partner only. If both communication types are to be permitted for a single real CICS partner, the real CICS partner must be configured twice using the same address data and different LU names: Once as a dialog partner and once as an asynchronous partner.

Connections

Specifies the maximum number of concurrent connections allowed between proxy and the EIS partner.

Prefix

The prefix is included as a component in names which are used in configuration statements. This prefix must be unique for all proxies running on the same host. The prefix must comprise exactly three characters (uppercase letters or digits) to ensure that the generated names are unique. The first character must be an uppercase letter.

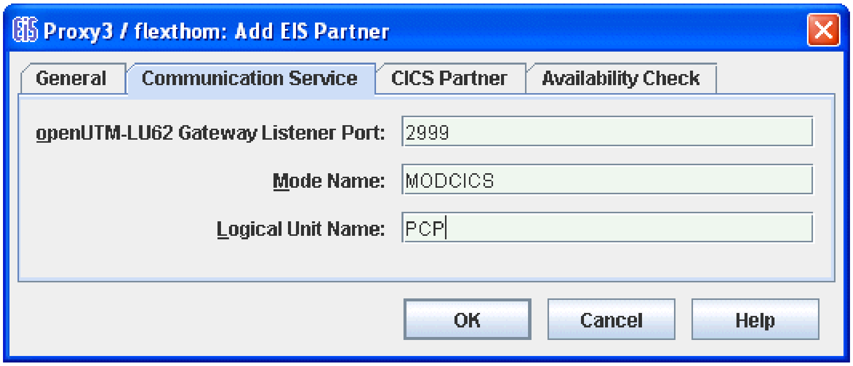
DLC Type

Specifies the communication type of the connection between the proxy and the EIS partner. It can be **LAN** or **IBM-EEDLC**

Communication Service tab

This tab has a different layout for proxy clusters containing multiple proxies. Consequently, both variants are displayed here.

Figure 34: Communication service properties of a CICS partner (single proxy)

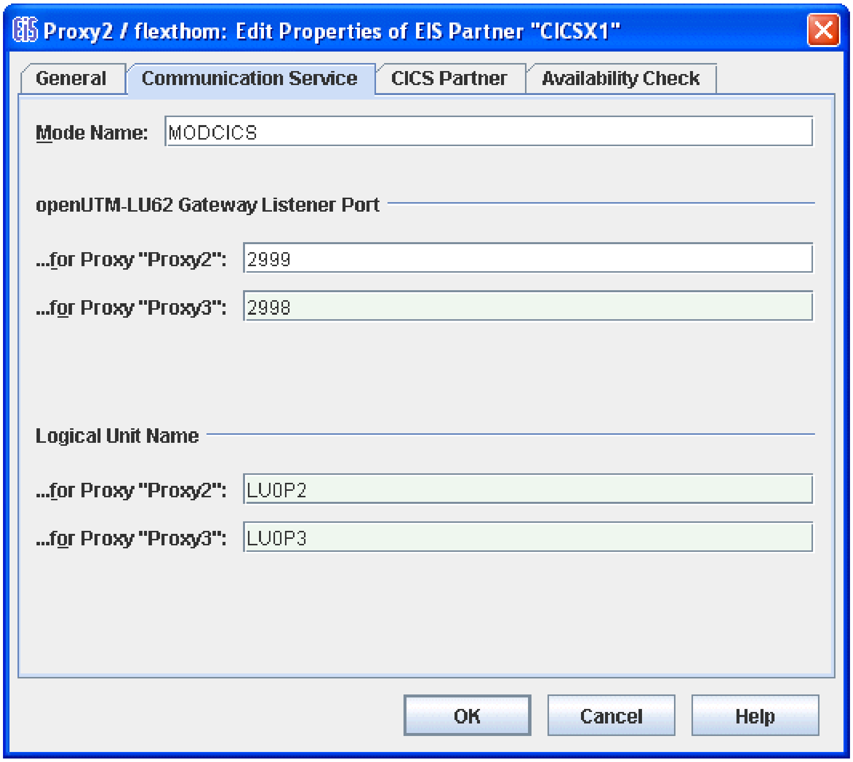


openUTM-LU62 Gateway Listener Port: 2999

Mode Name: MODCICS

Logical Unit Name: PCP

Figure 35: Communication service properties of a CICS partner in a proxy cluster



Mode Name: MODCICS

openUTM-LU62 Gateway Listener Port

...for Proxy "Proxy2": 2999

...for Proxy "Proxy3": 2998

Logical Unit Name

...for Proxy "Proxy2": LU0P2

...for Proxy "Proxy3": LU0P3

Mode Name

Specifies the name of an entry in the VTAM MODETAB on the z/OS mainframe. Such entries describe the properties of sessions between the communication service and VTAM. You must specify the name of an entry that is suitable for LU6.2 communication with CICS.

openUTM-LU62 Gateway Listener Port

Specifies the partner-specific number of the port where the openUTM-LU62 Gateway is listening for messages. This number must not be used by another EIS partner communication service configuration or by another application.

In the case of a proxy cluster, one field is output for each proxy. In these fields, enter the port number on which the openUTM-LU62 Gateway receives messages for the EIS partner for each proxy. The port numbers must be different and must be uniquely assigned to an "EIS partner <-> proxy" pair. In other words, the same port number must not be assigned for different "EIS partner <-> proxy" pairs.

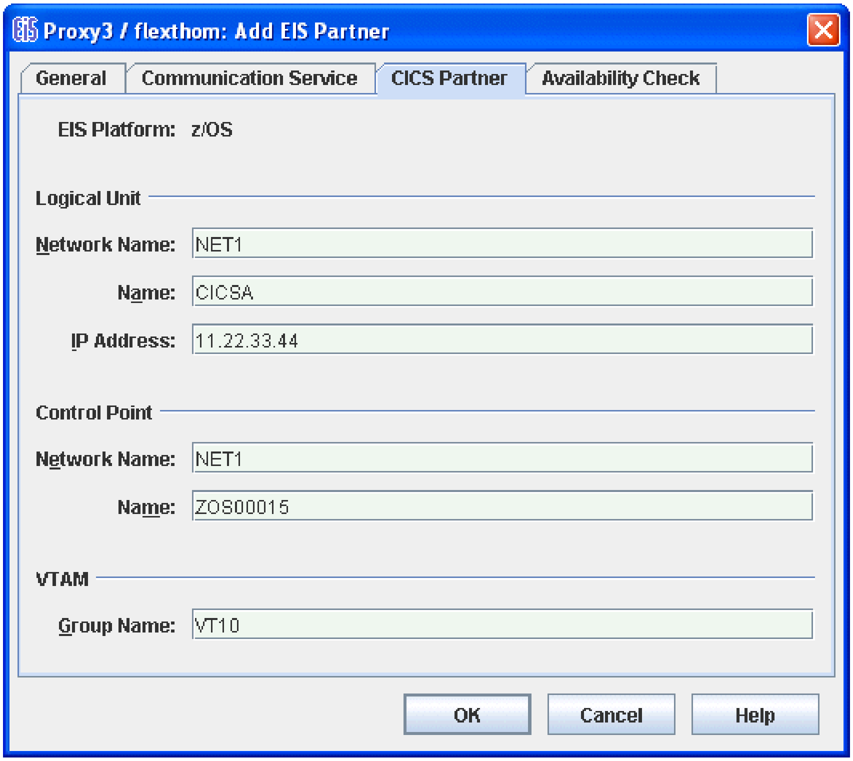
Logical Unit Name

Specifies the proxy's unique application name in the SNA network.

In the case of a proxy cluster, one field is output for each proxy. Here, you can enter the application name of the corresponding proxy.

CICS Partner tab

Figure 36: Properties of the CICS partner on a z/OS mainframe



EIS Platform: z/OS

Logical Unit

Network Name: NET1

Name: CICSA

IP Address: 11.22.33.44

Control Point

Network Name: NET1

Name: ZOS00015

VTAM

Group Name: VT10

The layout of this dialog box depends on the DLC type. Many of the fields are only displayed for a specific setting. All the fields are described below.

EIS Platform

The **EIS Platform** the CICS partner is running on. Currently only **z/OS** is possible, cannot be changed.

Logical Unit

Describes the CICS partner's logical unit.

- **Network Name** is the NETID in the VTAM start options on the z/OS mainframe.
- **Name** is the application name of the CICS region as specified in the VTAM statement APPL at the z/OS mainframe.
- For **IP Address**, enter the IP address of the z/OS mainframe on which CICS runs.

Control Point

Specifies the VTAM control point on the z/OS mainframe.

- **Network Name** is the NETID in the VTAM start options on the z/OS mainframe.
- **Name** is the value specified in the SSCPNAME parameter in the VTAM start options on the z/OS mainframe.

VTAM

Group Name specifies the VTAM group name. This is the name of the GROUP macro in the VTAM major node definition for the Enterprise Extender.

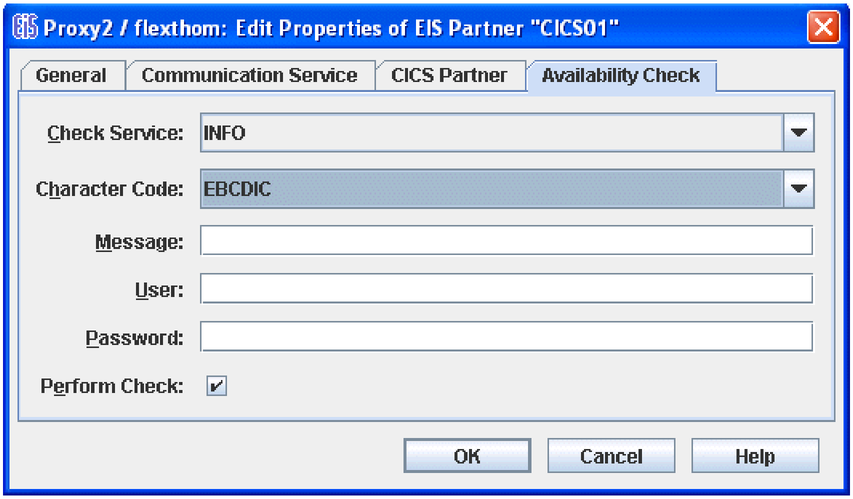
MAC Address

MAC address of the z/OS mainframe on which the CICS partner runs if the DLC type is LAN. Otherwise, this field is not displayed

i Explanations of SNA-specific terms can be found in the glossary.

Availability Check tab (CICS partners)

Figure 37: Properties of the CICS type EIS partner for the availability check



Proxy2 / flexthom: Edit Properties of EIS Partner "CICS01"

Check Service: INFO

Character Code: EBCDIC

Message:

User:

Password:

Perform Check: ☑

This dialog box allows you to select a dialog service in the partner application which is called when checking the availability of an EIS partner. When this is done, the message defined in **Message** is passed to the specified service in the EIS partner application. As soon as the response is received, the EIS partner is flagged as available. The response is output in the Management Console.

You cannot complete this dialog box properly unless you have defined at least one outbound service for the EIS partner, see Section 6.7.1, "Configuring outbound services".

Check Service

This lists all the outbound services defined for this partner. Select a suitable service from the list. The service is always of type **Dialog**.

Character Code

Character set used by the EIS partner:
Possible values: **ASCII** or **EBCDIC**.

Message

Message sent to the EIS partner when checking availability. The maximum length of the message is 80 characters.

Users

User ID if the service is to be called under a specific user ID.

Password
Password for the user ID if a password is required.

Perform Check
This option allows you to temporarily disable the availability check for this EIS partner without losing the settings, such as the service to be called.

If the option is enabled, the availability check is performed with the specified parameters.

If you disable this option, the availability check for this EIS partner is disabled until the option is enabled again. Both automatic availability checking and the availability check performed by issuing a command in the proxy or proxy cluster's context menu are disabled. If the EIS partner is checked "manually" (using its context menu), the check is performed irrespective of this setting.

6.6.2.2 Configuration files for the EIS partners of the type CICS

The Management Console creates configuration fragments for the defined proxy and EIS partners. Generation of all configuration files is performed automatically by the Management Console the next time you save the proxy configuration (see "Saving and activating the configuration of the BeanConnect proxy" on page 233). All configuration statements for the proxy and all defined EIS partners are generated.

The Management Console generates configuration statements for the following components:

- BeanConnect proxy container
- openUTM-LU62 Gateway of the proxy
- all partner applications of the BeanConnect proxy
- VTAM (for the connection)
- Communications service, i.e. SNAP-IX or IBM Communications Server (for the proxy side of the connection)

The configuration files are located in the directory `<MC_Home>/genfiles` under the following names:

- Single proxy:

  `ProxyID.<p-id>.EISPartnerID.<e-id>.CICS.txt`

  This file contains generation statements for the CICS partner application.

  `ProxyID.<p-id>.EISPartnerID.<e-id>.VTAM.txt`

  This file contains generation statements for the communication component at the EIS partner's side of the connection (VTAM).

- Proxy cluster:

  `ClusterID.<c-id>.ProxyID.<p-id>.EISPartnerID.<e-id>.CICS.txt`

  This file contains generation statements for the CICS partner application.

  `ClusterID.<c-id>.ProxyID.<p-id>.EISPartnerID.<e-id>.VTAM.txt`

  This file contains generation statements for the communication component at the EIS partner's side of the connection (VTAM).

`<MC_Home>` is the directory under which the Management Console is installed.
`<p-id>`, `<e-id>` and `<c-id>` designate the IDs for the proxy, EIS partner and proxy cluster assigned by the Management Console.

Information on configuration tasks in the EIS partner can be found in Chapter 7, "Adapting the configuration in the EIS partner".

6.6.3 Configuring EIS partners of type XATMI

You must enable expert mode before you can use this function.

If you want to configure an EIS partner of type XATMI, then the proxy must be configured with **API Mode XATMI** or **ALL**, see Section 6.3.4.3, "Application Program Interface Mode (API Mode)".

When configuring the proxy, you must specify either **Only UTM Partners** or **UTM and CICS Partners** as the EIS partner type. You always configure an EIS partner of type XATMI in the same way as a partner of type openUTM.

Proceed as follows to add a new XATMI partner:

1. Click the **Add** button below the list of EIS partners or open the context menu for an existing EIS partner object or for the **EIS Partners** node and choose **Add EIS Partner**.
2. Define the properties for the EIS partner. The properties sheet opens automatically. It contains the tabs **General, UTM Partner** and **Availability Check**.

   Complete these fields in the same way as for an openUTM partner, see Section 6.6.1.1, "Adding EIS partners of the type UTM". The only difference compared to an EIS partner of type **openUTM** is that, when expert mode is enabled, the value **XATMI** is displayed in the **API Mode** field.

The same configuration data is displayed as for an EIS partner of type **openUTM**, see also Section 6.6.1.2, "Configuration files for EIS partners of type openUTM".

6.6.4 Removing an EIS partner

To remove an EIS partner, choose **Remove EIS Partner** from the context menu of the EIS partner. Alternatively, you can select one or more EIS partners in the list and click the **Remove** button beneath the list.

6.7 Configuring outbound communication

Outbound communication denotes communication from the application server to the EIS partner. For outbound communication, you have to configure outbound services and outbound communication endpoints.

An outbound service represents a service (e.g. transaction code) inside the EIS partner. Each service of an EIS partner which is to be called from the application server, must be configured as an outbound service. It can either be used implicitly if it is assigned as a service to a communication endpoint or can be set explicitly in the EJB via the method `setServiceName`.

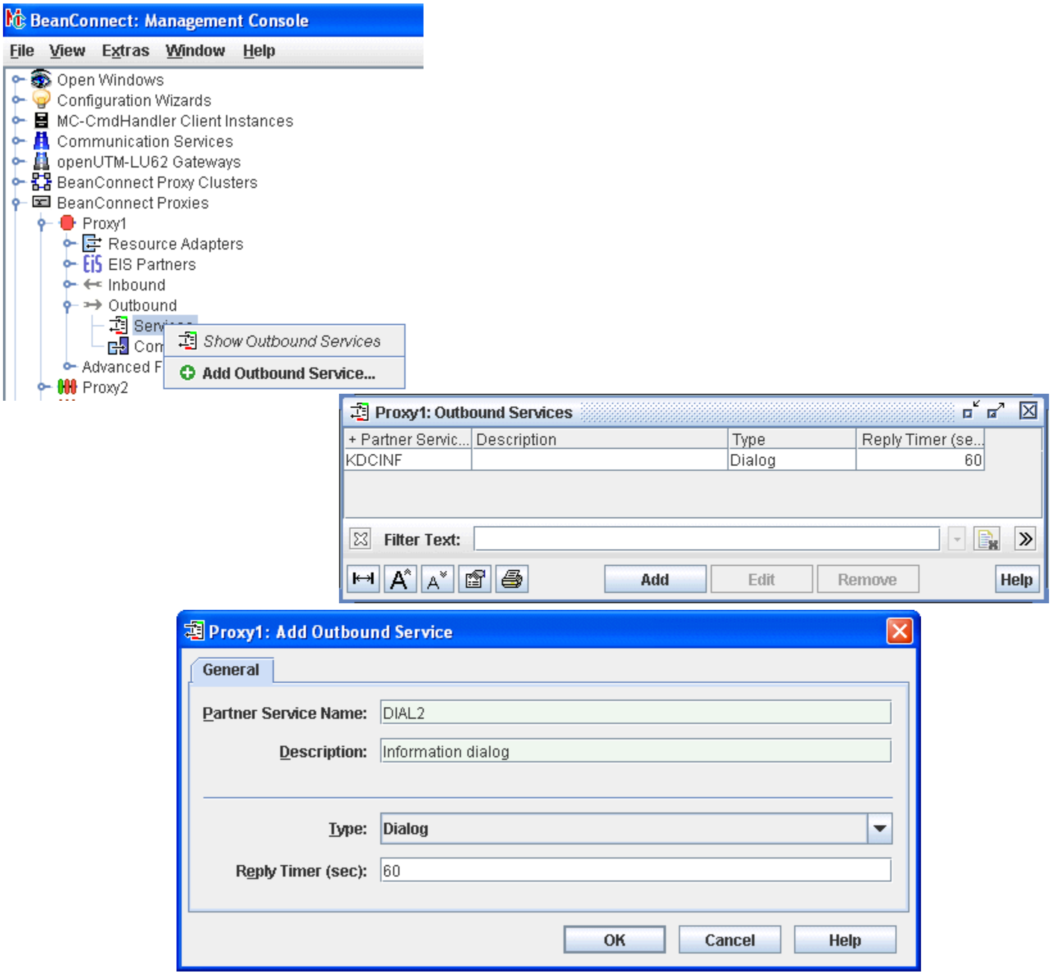
Each outbound communication endpoint represents a communication endpoint inside the EIS partner for outbound communication and is therefore EIS-specific. A connection factory is assigned to a communication endpoint with the configuration property `connectionURL` and is thus assigned to a specific EIS partner. An EIS partner can have multiple outbound communication endpoints.

You can find examples of how to use the BeanConnect programming interfaces for developing EJBs in the section Section 10.2.6, "Code samples for outbound communication".

6.7.1 Configuring outbound services

To configure outbound services, click the **Services** node beneath the **Outbound** node in the navigation tree of a proxy.

Figure 38: Configuring outbound services



+ Partner Servic...	Description	Type	Reply Timer (se...
KDCINF		Dialog	60

Partner Service Name: DIAL2

Description: Information dialog

Type: Dialog

Reply Timer (sec): 60

To display a list of the outbound services of a proxy, open the subtree beneath the **Outbound** node and then click on **Services** or alternatively choose **Show Outbound Services** from the context menu of the **Services** node.

To add a new outbound service, click the **Add** button below the list or choose **Add Outbound Service** from the context menu of an existing service or the **Services** node. The property sheet is opened.

The following properties have to be specified for an outbound service:

Partner Service Name / Description

Specifies the name of the service inside the EIS partner that is represented by this outbound service. Additionally you can enter a **Description** for the outbound service.

Type

Type specifies the communication type of the service. It can be **Dialog** or **Asynchronous**.

Reply Timer (sec)

The proxy monitors the responses of an outbound service. If no response is received within the time defined here, the proxy rolls back the corresponding transaction. This parameter can only be set or changed for services of the type **Dialog**.

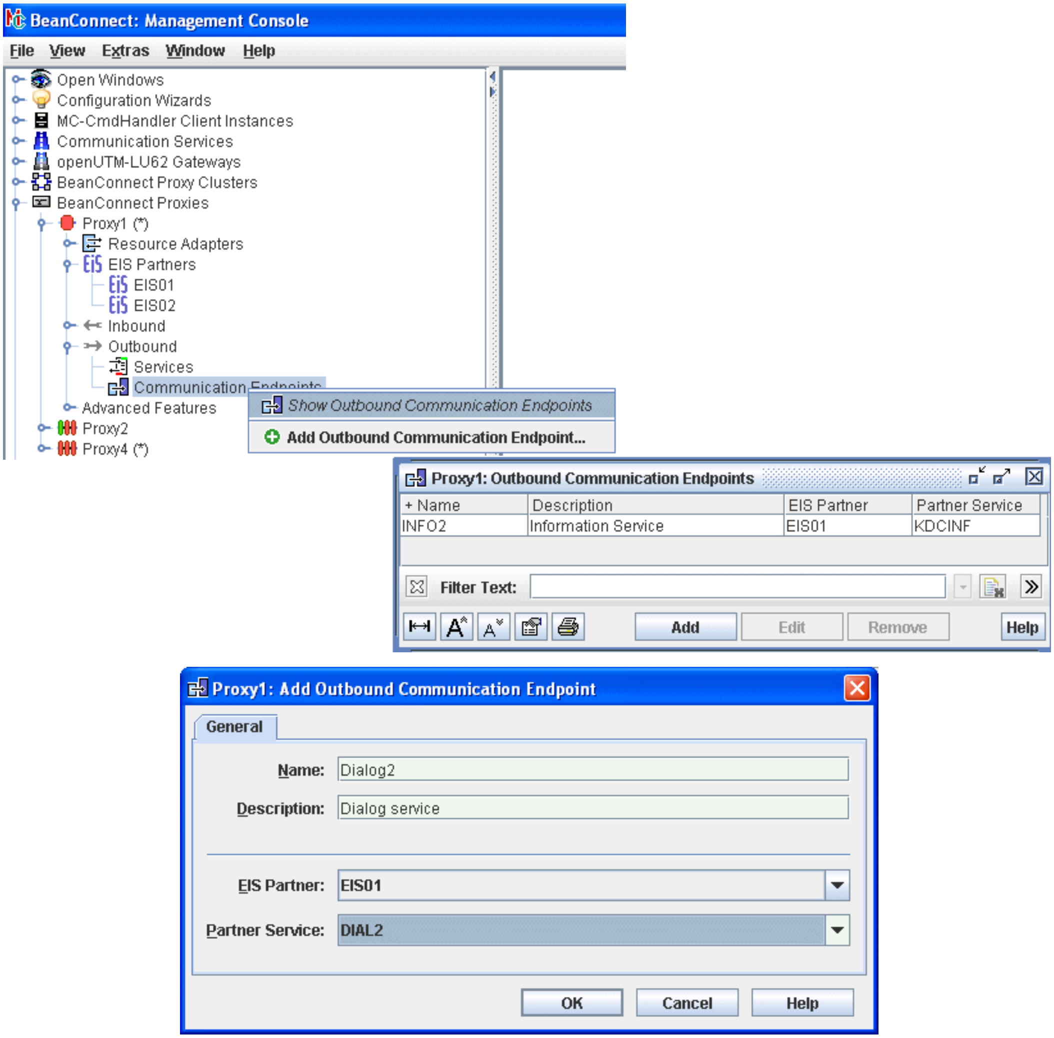
Removing an outbound service

To remove an outbound service, choose **Remove Outbound Service** from the context menu of the service. Alternatively you can select one or more services in the list and click the **Remove** button beneath the list.

6.7.2 Configuring outbound communication endpoints

To configure outbound communication endpoints, click the **Communication Endpoints** node beneath the **Outbound** node in the navigation tree of a proxy.

Figure 39: Configuring outbound communication endpoints


BeanConnect: Management Console
File View Extras Window Help
Open Windows
Configuration Wizards
MC-CmdHandler Client Instances
Communication Services
openUTM-LU62 Gateways
BeanConnect Proxy Clusters
BeanConnect Proxies
Proxy1 (*)
Resource Adapters
EIS Partners
EIS01
EIS02
Inbound
Outbound
Services
Communication Endpoints
Advanced Features
Proxy2
Proxy4 (*)
Show Outbound Communication Endpoints
Add Outbound Communication Endpoint...
Proxy1: Outbound Communication Endpoints
Filter Text:
Add
Edit
Remove
Help
Proxy1: Add Outbound Communication Endpoint
General
OK
Cancel
Help

+ Name	Description	EIS Partner	Partner Service
INFO2	Information Service	EIS01	KDCINF

Name: Dialog2

Description: Dialog service

EIS Partner: EIS01

Partner Service: DIAL2

To display a list of the outbound communication endpoints of a proxy, click on the **Communication Endpoints** node or alternatively choose **Show Outbound Communication Endpoints** from the context menu of the **Communication Endpoints** node.

To add a new outbound communication endpoint, click the **Add** button below the table. Alternatively, you can choose **Add Outbound Communication Endpoint** from the context menu of an existing endpoint or the **Communication Endpoints** node. The property sheet is opened.

The following properties have to be specified for an outbound communication endpoint:

Name / Description

Name specifies the symbolic name of the outbound communication endpoint. Additionally, you can enter a **Description** for the outbound communication endpoint.

The deployer of a bean uses this name when deploying a bean in the application server to refer to the service inside the EIS partner. For detailed information on this topic please refer to section Section 4.3, "Setting configuration properties for outbound communication via OSI-TP / LU6.2".

EIS Partner

Name of the EIS partner the communication endpoint belongs to. The EIS partner must be specified before.

Partner Service

Specifies the real name of the service inside the EIS partner that is represented by this outbound communication endpoint. This service must have been defined already as an outbound service. All the defined services are shown in the drop-down list.

i Each `<connector-instance>` entry contained in the `weblogic-ra.xml` file, must point to a defined outbound communication endpoint (see Section 4.3.1.3, "Defining configuration properties for OSI-TP / LU6.2").

To remove an outbound communication endpoint, choose **Remove Outbound Communication Endpoint** from the context menu of the endpoint. Alternatively you can select one or more endpoints in the list and click the **Remove** button beneath the list.

6.8 Configuring inbound communication

During inbound communication, an EIS partner sends messages to a message endpoint application in a Java EE application server.

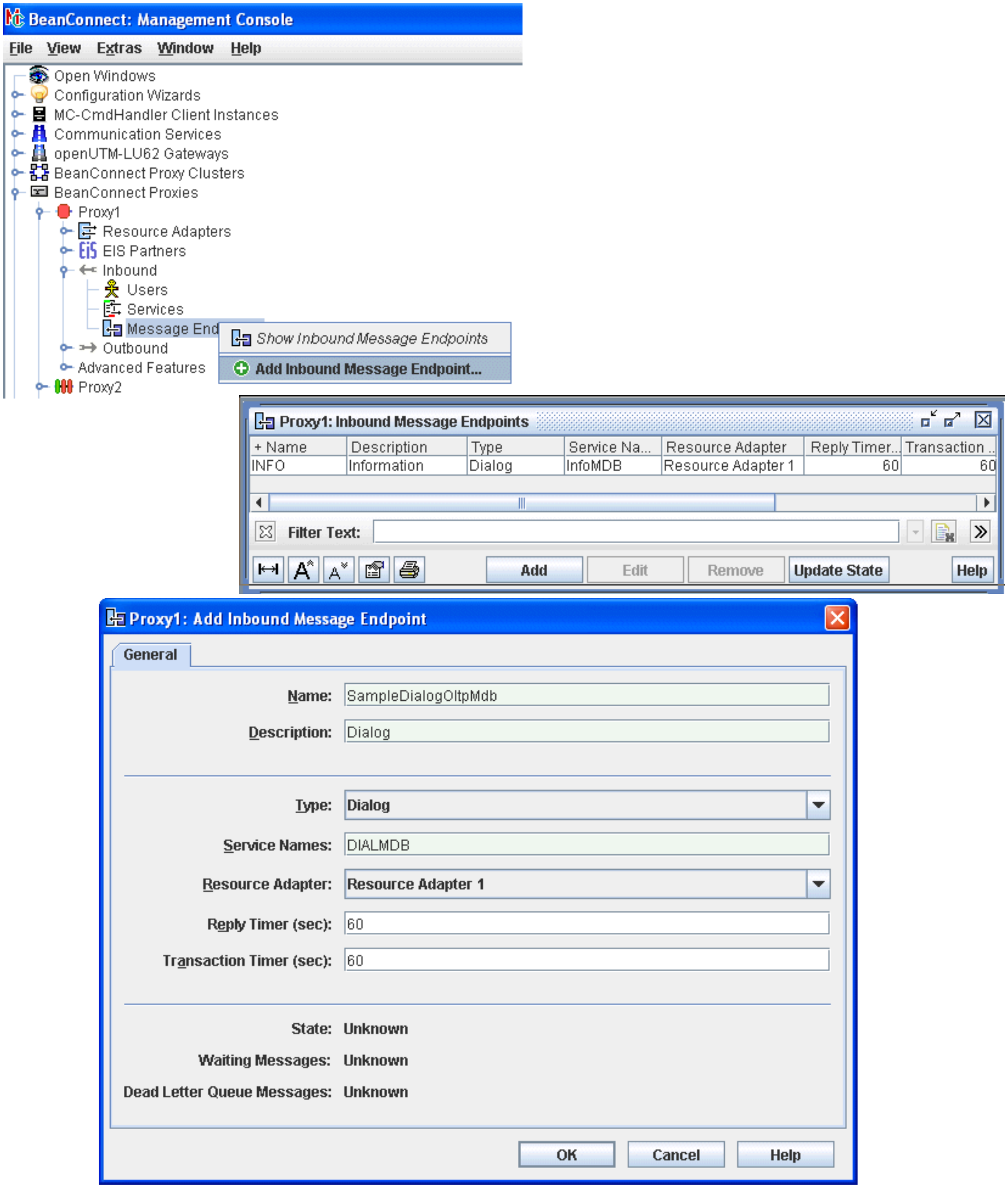
The name of the message endpoint, which is defined in the OLTP message-driven bean's deployment descriptor, must be known to the proxy. You must therefore configure an inbound message endpoint in the Management Console. The name of this inbound message endpoint must match the OLTP message-driven bean's `messageEndpoint` property which is defined in the file `ejb-jar.xml`.

You will find examples on using the BeanConnect programming interfaces for developing OLTP message-driven beans for inbound communication in the section Section 10.3.8, "Code samples for inbound communication".

6.8.1 Configuring inbound message endpoints

The inbound message endpoints are accessible from the **Message Endpoints** node beneath the **Inbound** node in the navigation tree's proxy subtree.

Figure 40: Configuring inbound message endpoints



+ Name	Description	Type	Service Na...	Resource Adapter	Reply Timer...	Transaction ...
INFO	Information	Dialog	InfoMDB	Resource Adapter 1	60	60

Name: SampleDialogOltpMdb

Description: Dialog

Type: Dialog

Service Names: DIALMDB

Resource Adapter: Resource Adapter 1

Reply Timer (sec): 60

Transaction Timer (sec): 60

State: Unknown

Waiting Messages: Unknown

Dead Letter Queue Messages: Unknown

To display the list of the inbound message endpoints of a proxy, click on the **Message Endpoints** node or alternatively choose **Show Inbound Message Endpoints** from the context menu of the **Message Endpoints** node.

To add a new inbound message endpoint, click the **Add** button below the list, choose **Add Inbound Message Endpoint** from the context menu of an existing endpoint or from the context menu of the **Message Endpoints** node. The property sheet is opened.

The following properties have to be specified for an inbound message endpoint:

Name / Description

Name specifies the symbolic name of the inbound message endpoint. This name must correspond to the name used in the deployment descriptor of the OLTP message-driven bean within the application server (property `messageEndpoint` defined in `ejb-jar.xml`) (see Section 4.5, "Setting configuration properties for inbound communication"). Additionally, you can enter a **Description** for the inbound message endpoint.

Type

Type specifies the communication type of the connection. Depending on the message listener interface implemented by the OLTP message-driven bean, the type can be **Dialog** or **Asynchronous**.

BeanConnect supports the following message listener interfaces (defined as `messaging-type` in the `ejb-jar.xml` file):

- `net.fsc.jca.communication.AsyncOltpMessageListener` (asynchronous communication)
- `net.fsc.jca.communication.OltpMessageListener` (dialog communication)
- `javax.resource.cci.MessageListener` (dialog communication)

Service Names

Defines one or more inbound services that are assigned to the inbound message endpoint; a service name may be up to eight characters in length. If you specify multiple services then they must be separated by commas. However, exactly one inbound message endpoint must be assigned to an inbound service.

In the case of inbound communication with an openUTM partner over OSI-TP, each of these names must be explicitly generated in the EIS partner. To this end, the name is specified as the value of the RTAC parameter in an LTAC statement.

In a CICS program, the service name is used to address the inbound message endpoint.

You can modify the properties of an inbound service, see Section 6.8.2, "Configuring inbound services".

Resource Adapter

In multiple resource adapter mode, you select the resource adapter assigned to the inbound message endpoint here. If only one resource adapter has been defined then this is displayed (cannot be changed). This field is not output in cluster operation.

Reply Timer (sec)

Monitors the response time of the resource adapters on calling the OLTP message-driven bean.

If no response has been received from the resource adapter after this time has elapsed, the proxy container clears the connection to the resource adapter and rolls back the transaction if necessary.

If you specify 0, no monitoring is performed.

Transaction Timer (sec)

Monitors the transaction duration in the application server if a transaction is propagated to the application server. If you specify 0, no monitoring is performed.

A transaction is propagated to the application server

- if the `onMessage()` method of the OLTP message-driven bean was deployed with the transaction attribute `Required` and `Asynchronous` was selected as the **Type** or
- if the `onMessage()` method of the OLTP message-driven bean was deployed with the transaction attribute `Required` and `Dialog` was selected as the **Type** and a transaction was propagated from EIS to the proxy.

If the transaction is not completed in the specified period, it is rolled back.

Make sure you take account of the time the EIS partner requires for processing, e.g. for accessing a database. Do not set this value too low.

If you activate both timers (both values > 0), you should set a value for the **Transaction Timer** that is at least as great as that of the **Reply Timer**.

State

Displays the state of the inbound message endpoint. The inbound message endpoint can have the state **Unknown**, **Available** or **Not Available**.

- **Unknown** means that the state has not yet been checked.
- **Available** means that an inbound message endpoint with this name exists in the resource adapter. The proxy container must be available to obtain the state of an inbound message endpoint.
- **Not Available** means that an inbound message endpoint with this name does not exist in the resource adapter.

You can update the state by clicking the **Update State** button beneath the list or by choosing **Update State** from the context menu of the endpoint. Additionally, the names of other inbound message endpoints available in the resource adapter which you have not defined in the Management Console are displayed.

Waiting Messages

This field is only displayed in expert mode. The value specifies the number of messages addressed to the services indicated in **Service Names** that are currently waiting in the proxy container.

Unknown means that the status has not yet been checked. Values > `0` are only possible if the type is **Asynchronous**.

Dead Letter Queue Messages

This field is only displayed in expert mode. The value specifies how many messages were originally sent to this message endpoint and are currently in the dead letter queue of the proxy container.

Unknown means that the status has not yet been checked. Values > `0` are only possible if the type is `Asynchronous`.

If the proxy is not running at the time the inbound message endpoint is added then the configuration must be updated before the proxy is started again (**Update Configuration**, see Section 6.9, "Saving and activating the configuration of the BeanConnect proxy").

To remove an inbound message endpoint, choose the **Remove Inbound Message Endpoint** command in the endpoint's context menu. Alternatively, you select one or more endpoints in the list and then click the **Remove** button below the list.

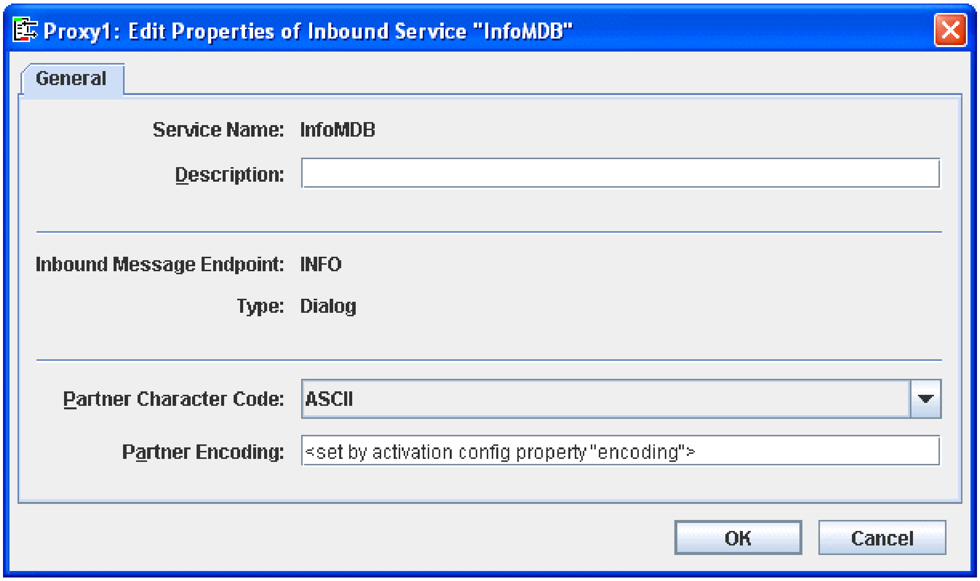
6.8.2 Configuring inbound services

An inbound service is a service which an EIS partner addresses during inbound communication. You specify the name of the service when configuring an inbound message endpoint. You can also specify multiple service names for an inbound message endpoint. However, exactly one inbound message endpoint must be assigned to any given service name.

You can modify the coding properties of an inbound service. To do this, proceed as follows:

- Click the **Services** node in the **Inbound** node and then click **Show Inbound Services**. A list of all the inbound services is now displayed.
- Select the required inbound service in the list and click the **Edit** button (alternatively: double-click the service).
- Configure the inbound service in the dialog box **Edit Properties of Inbound Service...** .

Figure 41: Configuring an inbound service



Service Name: InfoMDB

Description:

Inbound Message Endpoint: INFO

Type: Dialog

Partner Character Code: ASCII

Partner Encoding: <set by activation config property "encoding">

The following inbound service properties are displayed or can be defined.

Service Name
Name of the inbound service as specified in the assigned message endpoint, cannot be changed. This is the name used by the EIS partner to address the service.

Description

You can enter a description here.

Inbound Message Endpoint

Name of the inbound communication message endpoint, cannot be changed.

Type

Type of the inbound service (**Dialog** or **Asynchronous** cannot be changed).

Partner Character Code

Specifies the type of character set used in the EIS partner (possible choices: **ASCII** or **EBCDIC**). This setting is used to send a correctly encoded message to the EIS partner if an error occurs before the inbound message endpoint is called.

Partner Encoding

Name of a code table for converting byte code (e.g. EBCDIC) to Java Unicode. If you specify a code table here, the following values are overwritten in the deployment descriptor of the message-driven bean:

- `encoding` is replaced by the value specified here
- `encodingActive` is set to `true`

The value `<set by activation config property "encoding">` (default) or a blank entry causes the setting in the deployment descriptor to be used.

i If the proxy is configured with **API Mode: All** then the **XATMI** option is also displayed in expert mode. You must enable this option if the service is to be used by an XATMI partner.

6.8.3 Setting up users for access to inbound message endpoints

If the EIS partner uses user IDs for inbound communication then you must define these in the proxy as otherwise the job will be rejected. if the user ID needs a password then you must also define this.

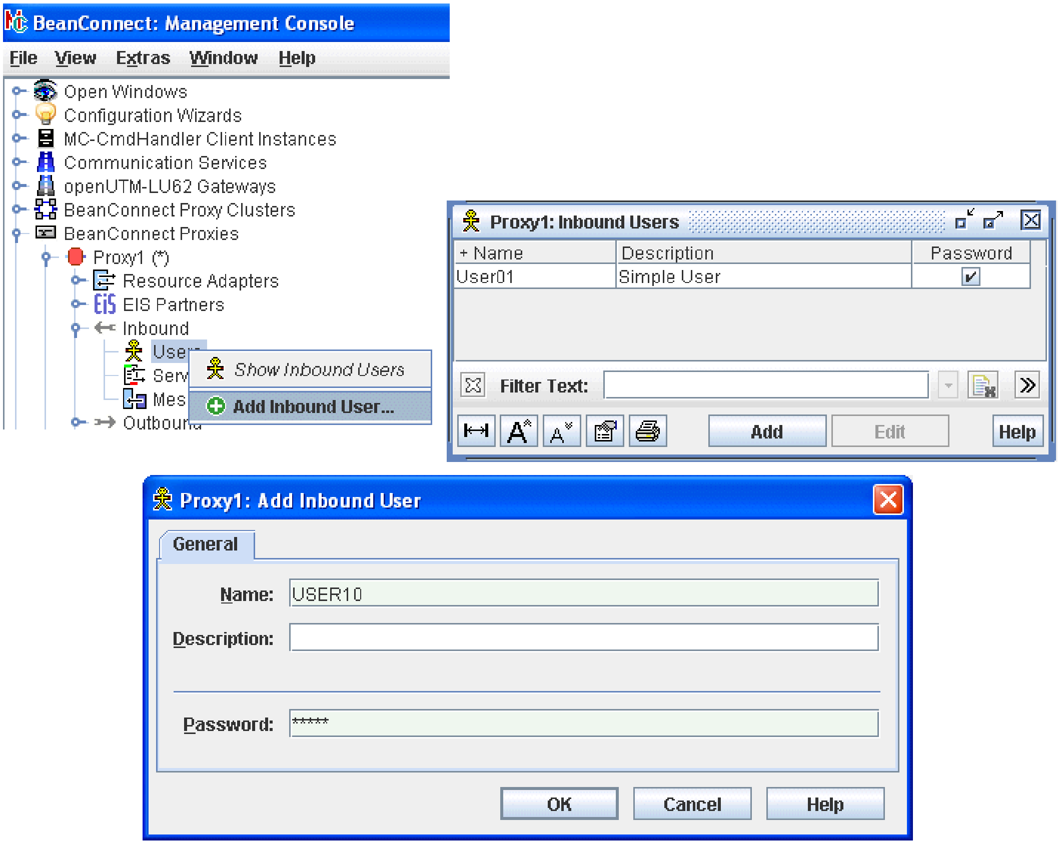
If you want to use the JCA 1.6 security inflow functionality then the EIS partner must use user IDs for inbound communication.

Otherwise, this operation is optional.

You access the user entries via the **Users** node which is located below the **Inbound** node in the proxy subtree.

To display the list of users, click the **Users** node. Alternatively, open the context menu for the **Users** node and choose **Show Inbound Users**.

Figure 42: Configuring users for inbound communication



+ Name	Description	Password
User01	Simple User	☑

Name: USER10

Description:

Password: *****

To add a new user, click the **Add** button below the list or choose **Add Inbound User** from the context menu of an existing user or the **Users** node.

Enter the **Name**. Additionally, you optionally can enter a **Description** and a **Password** for the inbound user.

To remove a user from the Management Console, choose **Remove Inbound User** from the context menu of the user. Alternatively you can select one or more users in the list and click the **Remove** button beneath the list.

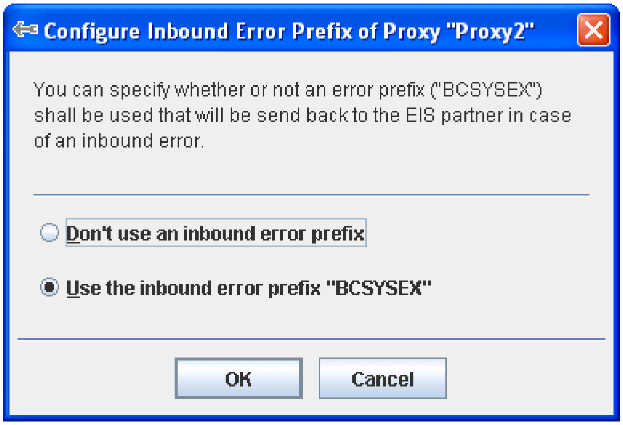
6.8.4 Configuring the error message prefix for inbound communication

If errors occur during inbound communication then an error message may be sent to the affected EIS partner. By default, these error messages have the prefix **BCSYSEX**.

You can activate and deactivate this prefix as follows:

- In the context menu of the **Inbound** node, choose the command **Configure Inbound Error Prefix...** .

Figure 43: Configuring the inbound error prefix



Configure Inbound Error Prefix of Proxy "Proxy2"

You can specify whether or not an error prefix ("BCSYSEX") shall be used that will be send back to the EIS partner in case of an inbound error.

○ Don't use an inbound error prefix

● Use the inbound error prefix "BCSYSEX"

- Select the appropriate options in the dialog box:

  Don't use an inbound error prefix (deactivate prefix)

  Use the inbound error prefix "BCSYSEX" (activate the prefix)

The affected proxy must not be running when you do this. The change takes effect when you save the proxy.

6.9 Saving and activating the configuration of the BeanConnect proxy

After adding a new proxy or changing the configuration of an existing proxy in the Management Console, you must save and activate the configuration to bring the configuration into effect. To obtain information on outstanding activities you can use the todo topics feature (see Section 5.3.6, "Todo topics").

i The following description also applies if you are running the BeanConnect proxy in a cluster.

The necessary steps depend on the changes you have made.

If data such as the LU name or control point name has been modified for the openUTM-LU62 Gateway and the communication service then the configuration must be saved and the components must be restarted. Only then do the changes take effect.

The configuration must be saved and the proxy container must be restarted to bring the changes into effect if you have

- changed the settings for communication with the resource adapter (see Section 6.5, "Configuring the BeanConnect resource adapter").
- created, modified or deleted an inbound user, an inbound message endpoint, an outbound service or an outbound communication endpoint.

Select one of the following commands from the context menu of the proxy:

- If the proxy is running: **Save/Restart** – **Save & Restart Proxy**
- If the proxy is not running: **Save/Restart** – **Save** and then start the proxy with **Start Proxy**.

The steps listed below are necessary and sufficient if you have added or deleted an EIS partner or changed its configuration. Activation of the new configuration cannot be carried out while the proxy is running. Therefore, proceed as follows after you have finished the configuration activities:

1. Select **Save/Restart** – **Save** from the context menu of the proxy.

   The newly defined configuration is saved and new configuration files are generated for all EIS partners of the proxy.

2. If the proxy is running, stop it by selecting **Stop Proxy** from the context menu of the proxy. All proxy components are stopped.
3. Select **Update Configuration** from the context menu of the proxy to activate the new configuration.
4. Start the proxy by selecting **Start Proxy** from the context menu of the proxy. The proxy and proxy components are started with the changed configuration.
5. It may be necessary to enter the new configuration data for the EIS partner in the EIS partner

6.10 Configuring the Management Console command handler (MC-CmdHandler)

The Management Console Command Handler (MC-CmdHandler) is a stand-alone Java application that enables the Management Console to administer remote proxies, proxy components, resource adapters or the Log4j configuration.

An MC-CmdHandler is installed for each BeanConnect proxy. In addition, you can also install the MC-CmdHandler separately on computers other than the proxy computer. This is necessary in the following cases:

- If it is not possible to access required files such as the deployment descriptor `ra.xml` in the BeanConnect RAR archive or the resource adapter's Log4j configuration because they are present on different computers and/or under different IDs.
- If it is not possible to access components such as openUTM-LU62 or communication services because they are present on different computers and/or under different IDs.

The MC-CmdHandler is a socket listener that waits at a listener port for orders given by the Management Console. The MC-CmdHandler is able to perform basic file transfer tasks such as listing directories, supplying information about files, fetching and updating files. Beyond this, it is also possible to execute scripts on a remote system.

The following conditions must be met before the above-mentioned components can be administered on remote hosts:

- The MC-CmdHandler must be started on the host on which the proxy or the components to be administered are running (resource adapter, openUTM-LU62 and communication service).
- The Management Console must be able to access the MC-CmdHandler.

6.10.1 Security and privileges

The MC-CmdHandler checks user authorizations. This prevents unauthorized individuals from using the MC-CmdHandler to manipulate file systems on remote hosts. Requests are accepted only if the (encrypted) password that accompanies the request matches the password of the MC-CmdHandler.

The privileges of the MC-CmdHandler are the same as the privileges of the system user ID under which the MC-CmdHandler was started. This is relevant when the MC-CmdHandler accesses files or executes scripts. It is therefore necessary to start each MC-CmdHandler under the same user ID as the component(s) to be administered.

Notes on using the MC-CmdHandler

The following points must be noted if an MC-CmdHandler is to be used to administer a proxy:

- When the proxy is entered in the Management Console's administration data, the password of the employed MC-CmdHandler must correspond to the administration password of the proxy that is to be included.
- If the administration password for a proxy changes then the password of the MC-CmdHandler that is used to administer the relevant proxy also changes.

  Therefore, if several proxies are installed on a computer, a separate MC-CmdHandler should be used for each proxy.

6.10.2 Administering the MC-CmdHandler

You will find scripts for starting, checking and shutting down the MC-CmdHandler under`shsc` in the proxy container's home directory or under `shsc` in the MC-CmdHandler's installation directory if the MC-CmdHandler is installed separately.

6.10.2.1 Starting the MC-CmdHandler

Starting the MC-CmdHandler on UNIX systems

You start the MC-CmdHandler using the following script in the proxy container's home directory or in the MC-CmdHandlers' installation directory

- `shsc/startmccmdhandler.sh`

i If you do not want the MC-CmdHandler to be shut down automatically on the next logoff, you must either start it as a service (see Section 6.10.2.3, "Configuring an MC-CmdHandler as a service") or start it using the following command:

```
nohup shsc/startmccmdhandler.sh &
```

Starting the MC-CmdHandler on Windows systems

You start the MC-CmdHandler using the following script in the proxy container's home directory or in the MC-CmdHandlers' installation directory:

- `shsc\startmccmdhandler.cmd`

If the MC-CmdHandler is located in the proxy container's home directory, you can also start it via the program group. To do this, choose

Start - Programs - BeanConnect V3.0A00 - Proxy <container> - MC-CmdHandler - MC-CmdHandler Startup

6.10.2.2 Shutting down the MC-CmdHandler

Shutting down the MC-CmdHandler in UNIX systems

You shut down the MC-CmdHandler with the following script in the proxy container's home directory or in the MC-CmdHandler's installation directory:

- `shsc/shutmccmdhandler.sh`

Shutting down the MC-CmdHandler in Windows systems

You shut down the MC-CmdHandler with the following script in the proxy container's home directory or in the MC-CmdHandler's installation directory:

- `shsc\shutmccmdhandler.cmd`

If the MC-CmdHandler is located in the proxy container's home directory, you can also shut it down it via the program group. To do this, choose

Start - Programs - BeanConnect V3.0A00 - Proxy <container> - MC-CmdHandler - MC-CmdHandler Shutdown

6.10.2.3 Configuring an MC-CmdHandler as a service

Configuring an MC-CmdHandler as a service on UNIX systems

If the MC-CmdHandler is to be started as a service then it must be configured. To do this, enter a line in the file `/etc/init.d/bcmccmdhandler.dat` for every service that is to be started. This line contains the user ID under which the service is to be started as well as the directory under which the MC-CmdHandler was installed, e.g.:

- `proxyuser   /home2/proxyuser/BCCONT`

To start/stop the MC-CmdHandler as a service, call the following script:

- `/etc/init.d/bcmccmdhandler.sh start | stop`

  Alternatively, you may also call the script with the options `restart` or `reload`. `restart` and `reload` are identical and each contain `stop` and `start`.

To remove a service from the UNIX system again, delete the corresponding line in the above-mentioned configuration file.

To enter and delete the service and to call the script `/etc/init.d/bcmccmdhandler.sh`, you require system administrator permissions. For further information on deleting a service, see Section 3.9, "Uninstalling the BeanConnect tools".

Configuring an MC-CmdHandler as a service on Windows systems

If the MC-CmdHandler is installed without a BeanConnect container, then the MC-CmdHandler is entered as a service at installation time under the name `BeanConnect MC-CmdHandler <port-number>` with autostart type `Manual`.

If the MC-CmdHandler is installed together with the BeanConnect container, then you must subsequently explicitly enter the MC-CmdHandler as a service. You do this using the script `shsc/MCCmdHandler_InstallSrv.cmd` in the container's home directory. Call the script with administration authorization:

- `<Cont_Home>/shsc/MCCmdHandler_InstallSrv.cmd`

This script enters the service with autostart type `Manual`.

If you want to use the MC-CmdHandler to administer a proxy container that is installed on a remote host then it can be necessary under certain circumstances to run the service under the corresponding user account and not under the system account (default value on the installation of the MC-CmdHandler). You change the user account setting via the Windows Control Panel (Control Panel/Administrative Tools/Services).

You can use the script `shsc/MCCmdHandler_UnInstSrv.cmd` to remove the service again.

6.11 Configuring the Management Console as a JMX client

The Management Console contains a JMX client This allows the Management Console to read current statistical and administrative data for the running resource adapter and to modify certain attribute values, see Section 8.7, "Monitoring the resource adapter with the Management Console".

The data is made available using MBeans. The JMX client is therefore also referred to as the MBean client.

> As JMX client, the Management Console can access all the MBeans of the relevant application server instance. However, this section describes only the MBeans which affect the resource adapter.

6.11.1 Defined resource adapter MBeans

The BeanConnect resource adapterprovides a number of types of MBeans. The following types of data are provided for each MBean:

- Attributes

  These are values, e.g. configuration values or statistical counters. Most of the attributes of the MBeans are read-only. Some of them can also be modified via the MBean interface.

- Operations

  These are operations (methods) that can be executed using MBeans, for example resetting counters.

- Notifications

  Notifications sent to the MBean client when certain events occur (e.g. when a transaction is rolled back). Before it is possible to send notifications, the MBean client must have explicitly subscribed to them.

The following MBeans are available:

- ResourceAdapter MBean

  MBean description:

  Administration interface of the BeanConnect Resource Adapter

  This MBean indicates the resource adapter configuration settings as defined in the file `ra.xml`. There is one MBean of this type for each resource adapter.

  Operations:

  – `checkProxyApplication`
    Checks whether the proxy application is available

  – `selectProxyApplication`
    Starts the selection of a new application if necessary.

- ManagedConnectionFactory MBean

  There are three variants of the ManagedConnectionFactory MBean:

  – OltpMBean for non-transactional OSI TP/LU6.2-connections:

    MBean description:

    Administration interface of a non-transactional BeanConnect OltpManagedConnectionFactory

  – OltpTaMBean for transactional OSI TP/LU6.2-connections:

    MBean description:

    Administration interface of a transactional BeanConnect OltpManagedConnectionFactory

  – UPICMBean for UPIC connections:

    MBean description:

    Administration interface of a BeanConnect UpicManagedConnectionFactory

  These MBeans indicate the configuration properties of the individual managed connection factories (deployment data) and provide information about connection usage (statistical data). The three variants of this MBean differ at the level of the attributes that are displayed.

Each managed connection factory has a corresponding MBean.

Operations:

- `cleanupPool`
  Remove Manged Connections from the pool
- `resetStatisticValues`
  Reset statistics

- Inbound MBean

  MBean description:

  Statistic data of inbound connections

  This MBean indicates the statistical data for all the inbound activities that cannot be assigned to a message endpoint. There is only one MBean of this type.

  Operations:

  - `resetStatisticValues`
    Rest statistics

- MessageEndpoint MBean

  MBean description:

  Administration interface of a BeanConnect Message Endpoint

  These MBeans indicate the configuration properties of the individual message endpoints (deployment data) and provide information about message endpoints usage (statistical data). Each message endpoint has a corresponding MBean.

  Operations:

  - `resetStatisticValues`
    Reset statistics

- Logging MBean

  MBean description:

  Administration interface of the BeanConnect logging

  This MBean indicates the Log4j configuration settings. All the Log4j Loggers are displayed together with their log levels. There is only one MBean of this type.

Operations:

- getLogLevel
  Show the LogLevel of a Log4j logger
- setLogLevel
  Modify the LogLevel of a Log4j logger

For further details on the data supplied by the MBeans and the options available via the administration functions, see Section 8.7, "Monitoring the resource adapter with the Management Console".

6.11.2 Setting up the JMX client in the Management Console

The JMX client requires teh following additional Java libraries to communicate with the Oracle WebLogic Server:

either `wljmxclient.jar` und `wlclient.jar`

or `wlfullclient.jar`.

If the Management Console runs on a different computer from the application server then you must download these libraries from Oracle's official download page and make them available.

In all cases, you must declare these additional Java libraries. To do this, it is necessary to extend the Management Console's classpath. The scripts `mc.cmd` (Windows systems) and `mc.sh` (Solaris, Linux systems) for starting the Management Console contain the l`wlfullclient.jar` library under `Enable WebLogic JMX client`. To use these statements, you must start the scripts using the `-weblogicjmx` option. Otherwise, the statements under `Enable WebLogic JMX client` will not execute.

In Windows systems, the script `mc.cmd` is located under `<MC_home>/bin` where `<MC_home>` is the Management Console's installation directory.

In Solaris/Linux systems, the script `mc.sh` is located in the installed BeanConnect's `Console/bin` directory. You must adapt the script `mc.sh` for the JMX client. However, the Management Console is started with the script `startconsole.sh` which is located in the Management Console's installation directory.

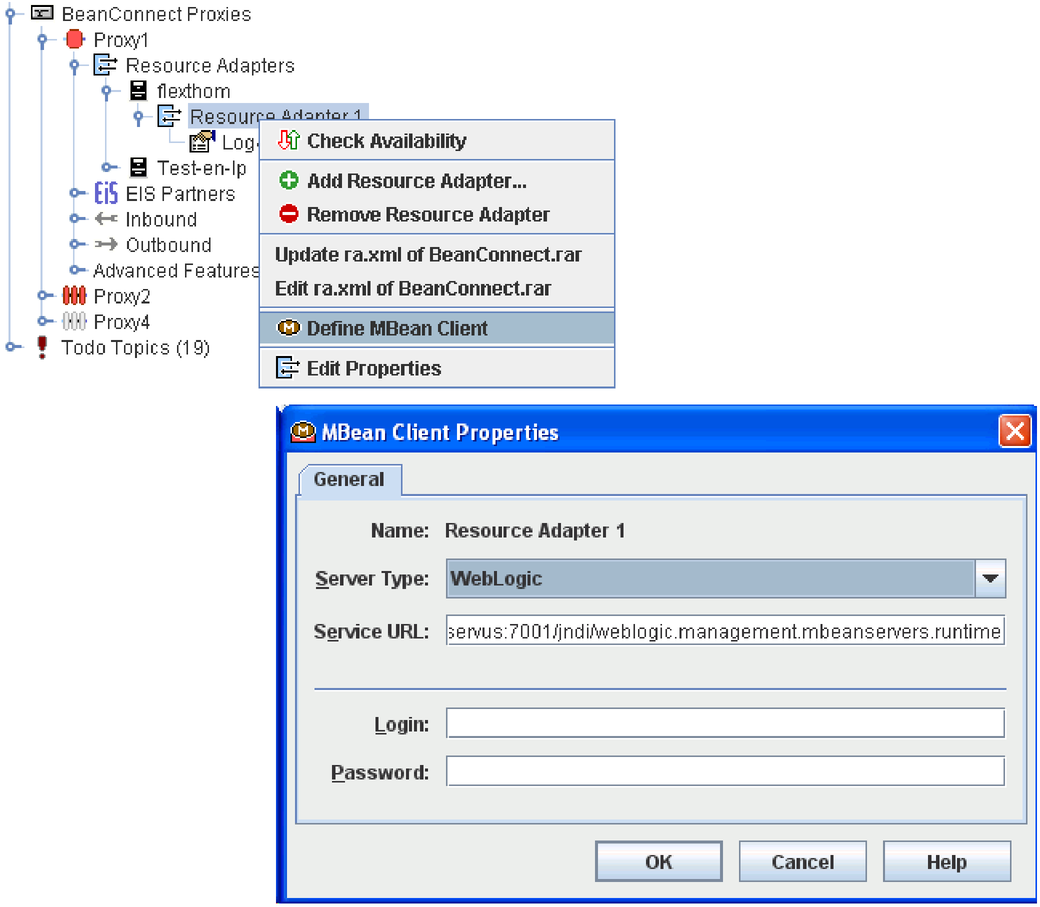
6.11.2.1 Setting up a JMX client

A JMX client is usually assigned to a resource adapter. However, you can also set up a free "stand-alone" JMX client, see section "Setting up free JMX clients".

Setting up a JMX client for resource adapters

To set up a JMX client for a resource adapter, choose the **Define MBean Client** command in the resource adapter's context menu and define the properties of the MBean client in the **MBean Client Properties** dialog box

Figure 44: Setting up a JMX client for resource adapters



Name: Resource Adapter 1

Server Type: WebLogic

Service URL: servus:7001/jndi/weblogic.management.mbeanservers.runtime

Login:

Password:

Name

Name of the resource adapter, cannot be changed.

Server Type

Select the type of application server.

Server URL

Specifies the URL that is to be used to establish the connection to the JMX server. The format of the URL depends on the type of application server. The Management Console proposes a default URL which you may need to modify. The default URL has the following format:

```
service:jmx:iiop://<server-host>:7001/jndi/
weblogic.management.mbeanservers.runtime
```

(Oracle WebLogic Server)

`<server-host>` is the name of the JMX server. This is followed by the port number of the relevant server. Here, the Management Console enters the name of the computer on which the resource adapter is running and proposes a default port number for the JMX server. You may need to adapt the port number.

Login

User ID required for login at the application server. The user ID usually possesses administration permissions.

Password

Password for the specified user ID.

The newly set up MBean client is represented by a separate node below the resource adapter.

Setting up free JMX clients

A free JMX client is a client that is not assigned to any resource adapter. To set up this type of client, open the **File** menu and choose the command **Add MBean Client**. Enter the properties in the dialog box **MBean Client Properties**. You proceed in the same way as for a client with a fixed resource adapter except that you must assign the name of the JMX client and that the name of the JMX client computer is not predefined. The MBean clients defined in this way are listed at the topmost level under the **MBean Clients** node. This node only exists if you have set up at least one free MBean client.

6.11.2.2 Establishing and clearing a connection to the JMX server

You must explicitly establish and clear the connection between the Management Console and the JMX server.

Establishing a connection to the JMX server

In the MBean context menu, choose **Connect to MBean Server**. If it has been possible to establish the connection, the icon for the MBean node changes and all the available MBean domains are displayed below the MBean node. If the connection is not established, an error message is output. This may provide information about the cause of the error.

Clearing a connection to the JMX server

To clear a connection to the JMX server, choose the command **Disconnect From MBean Server** in the MBean client's context menu. When the connection is cleared, the contents of the MBean domain nodes are removed from the navigation tree.

6.11.2.3 Removing a JMX client

To remove a JMX client, choose the command **Remove MBean Client** from the MBean client's context menu and confirm the query. The Management Console removes the node from the navigation tree, closes any windows that are open and deletes the configuration data from its administration files

7 Adapting the configuration in the EIS partner

In order to enable communication between the application server and an EIS partner, it is not sufficient to configure the application server, the resource adapter and the proxy. Some additional configuration activities are necessary in the EIS itself and on the platform that hosts the EIS.

This chapter contains information on

- Defining connections between BeanConnect and openUTM
- Defining connections between BeanConnect and other EIS partners

For detailed information on configuring openUTM applications, please refer to the openUTM documentation.

7.1 Adapting the configuration in EIS partners of type openUTM

An EIS partner of type openUTM is usually an openUTM application which communicates with the application server via outbound or inbound communications. However, it is also possible for an application from the openUTM environment, for example a UPIC application, to perform inbound access to the application server.

7.1.1 Defining connections between BeanConnect and openUTM

openUTM partners can be connected to BeanConnect via the

- OSI-TP protocol
- UPIC protocol
- Socket or RFC1006 protocol

The following sections indicate the parameters that have to be configured in the EIS partner.

In the case of connections to an openUTM partner running on BS2000/OSD, you may also need to create BMAP entries.

7.1.1.1 Defining an OSI-TP connection between BeanConnect and openUTM

An OSI-TP connection can be used for both outbound communication and inbound communication. An OSI-TP connection enables both transactional and non-transactional communication.

An OSI-TP connection between BeanConnect and openUTM requires an `OSI-LPAP` and an `OSI-CON` statement within the EIS partner's openUTM generation (KDCDEF).

On the proxy side the generation is processed as described in Section 6.6, "Configuring the EIS partners". On the EIS partner side the BeanConnect Management Console generates these statements in a text file in the directory `<MC_Home>/genfiles`.

In the case of a single proxy, the name of the generated input file is:

`ProxyID.<p-id>.EisPartnerID.<e-id>.UTM.txt`

In the case of a proxy cluster, the name of the generated input file is:

`ClusterID.<c-id>.EisPartnerID.<e-id>.UTM.txt`

`<p-id>`, `<e-id>` and `<c-id>` designate the IDs for the proxy, EIS partner and proxy cluster assigned by the Management Console.

BeanConnect does not transfer this configuration file to the EIS platform. The generated input file has to be transferred to the EIS partner host using common file transfer mechanisms. After that, the openUTM administrator can use the file to carry out the configuration activities.

7.1.1.2 Defining a UPIC connection for outbound communication between the openUTM partner and BeanConnect

From the openUTM partner point of view, the BeanConnect resource adapter is seen as a UPIC client.

A UPIC connection from BeanConnect to an openUTM partner requires a `TPOOL` statement within the KDCDEF generation of openUTM partner application. The Management Console does not generate this statement.

The configuration enables non-transactional UPIC outbound communication.

The openUTM connection is configured as follows:

```
BCAMAPPL UTMSERV
   ,T-PROT=RFC1006
  [,LISTENER-PORT = 11111] // only required for UNIX/Windows systems
  [,TSEL-FORMAT=T|A|E]        // only required for UNIX/Windows systems
```

```
TPOOL NUMBER=99
   ,PTYPE=UPIC-R
   ,LTERM=UPIC#R
   ,BCAMAPPL=UTMSERV
   ,PRONAM=*ANY
   ,Connect-Mode=MULTI
```

Please note the following:

- `TSEL-FORMAT` must match the configuration property `connectionURL`, parameter `TSEL`, attribute `rt` (see "connectionURL" on page 120).
- `UTMSERV` and the port number must match the values which are specified in the configuration property `connectionURL` parameters `remote` and `port` or in the program.

Example 10 Example of an URL

The openUTM partner application configured above can be addressed as follows with the configuration property `connectionURL`:

```
<config-property name="connectionURL"
                 value="upic://host:11111/UTMSERV?rt=t"/>
```

(for UNIX/Windows systems)

7.1.1.3 Defining a socket connection between the openUTM partner and BeanConnect

A socket connection or RFC1006 connection from an openUTM partner to BeanConnect requires a pair of `PTERM/LTERM` statements within the openUTM partner application's KDCDEF generation either with `PTYPE=SOCKET` for the openUTM socket protocol or with `PTYPE=APPLI` for the RFC1006 protocol. The Management Console does not generate these statements.

The configuration enables non-transactional asynchronous inbound communication.

7.1.1.4 Defining a BCMAP entry (only for BS2000/OSD partners)

An additional configuration step has to be carried out on the BS2000/OSD partner system in BCAM if a communication port number other than port 102 is used.

To assign another port than the default port `102` to the `OSI-CON`, `PTERM` or `TPOOL` statements in the openUTM configuration, you have to define a `BCMAP` entry.

When configuring a BS2000/OSD EIS partner with the Management Console the following text file with the according `BCMAP` entry is generated in the directory `<MC_Home>/genfiles`.

In the case of a single proxy, the name of the generated input file is:

`ProxyID.<p-id>.EisPartnerID.<e-id>.BCMAP.txt`

In the case of a proxy cluster, the name of the generated input file is:

`ClusterID.<c-id>.EisPartnerID.<e-id>.BCMAP.txt`

`<p-id>`, `<e-id>` and `<c-id>` designate the IDs for the proxy, EIS partner and proxy cluster assigned by the Management Console.

BeanConnect does not transfer this configuration file to the EIS platform. The generated input file has to be transferred to the EIS partner host using common file transfer mechanisms and can be used by the BCAM administrator to carry out the configuration activities.

For detailed information on configuring BCMAP entries, please refer to the BCAM documentation.

7.1.2 Defining connections between BeanConnect and other EIS partners

Each UPIC application and each application using transport-level protocols like RFC1006 or the openUTM socket protocol can be used as an EIS partner for non-transactional inbound communication. A UPIC application may only use dialog communication.

For detailed information on communication between UPIC applications or applications using transport-level protocols like RFC1006 or openUTM socket protocol and BeanConnect please refer to the openUTM documentation.

7.2 Adapting the configuration in EIS partners of type CICS

An EIS partner of type CICS is a CICS application which runs on an IBM mainframe.

7.2.1 Configuration in the CICS

The following configuration activities have to be carried out in the CICS itself:

- Definition of a connection, used to specify the connection parameters. This definition contains the name of the logical unit (`NETNAME` parameter) of the CICS partner application.
- Definition of a session, used to specify the session parameters. This definition contains the name of the connection (`CONNECTION` parameter) and the name of the session mode (`MODE` parameter). The properties of a session are then defined via the mode.

The BeanConnect Management Console generates a text file that contains the definitions of `CONNECTION` and `SESSION` (see Section 6.6.2, "Configuring EIS partners of type CICS").

In the case of a single proxy, the name of the generated input file is:

`ProxyID.<p-id>.EisPartnerID.<e-id>.CICS.txt`

In the case of a proxy in a cluster, the name of the generated input file is:

`ClusterID.<c-id>.ProxyID.<p-id>.EisPartnerID.<e-id>.CICS.txt`

`<p-id>`, `<e-id>` and `<c-id>` designate the IDs for the proxy, EIS partner and proxy cluster assigned by the Management Console.

BeanConnect does not transfer the file to the EIS platform. This file has to be transferred to the EIS partner host using common file transfer mechanisms and can be used by the CICS administrator to carry out the configuration activities.

7.2.2 Configuration of VTAM on an IBM mainframe

The VTAM (Virtual Telecommunications Access Method) component has to be configured for CICS partners running on the IBM mainframe (z/OS system). VTAM is an IBM product that runs on the mainframe and enables access to a CICS region on that mainframe. Requests from the proxy are passed to VTAM and then to CICS. Therefore, some VTAM definitions are needed.

VTAM definitions are coded using macros. The Management Console generates an input file for the VTAM configuration during the configuration run (see section Section 6.6.2, "Configuring EIS partners of type CICS"). This input file contains definitions for the Physical Unit (PU) in VTAM.

In the case of a single proxy, the name of the generated input file is:

```
ProxyID.<p-id>.EisPartnerID.<e-id>.VTAM.txt
```

In the case of a proxy in a cluster, the name of the generated input file is:

```
ClusterID.<c-id>.ProxyID.<p-id>.EisPartnerID.<e-id>.VTAM.txt
```

`<p-id>`, `<e-id>` and `<c-id>` designate the IDs for the proxy, EIS partner and proxy cluster assigned by the Management Console.

BeanConnect does not transfer the VTAM configuration file to the EIS platform. The generated input file has to be transferred to the EIS partner host using common file transfer mechanisms. After that, the VTAM administrator can use the file to carry out the configuration activities. Note that the VTAM administrator has to adapt some VTAM definitions to the needs of the partner application. BeanConnect cannot guarantee the uniqueness of the definitions when generating the file because it has no access to the complete VTAM definitions. Therefore, the configuration file contains some question marks which must be replaced with the proper values.

8 Administering BeanConnect

This chapter describes the administration tasks involved in operating BeanConnect.

You can carry out all the necessary operations using the BeanConnect Management Console. These operations include:

- Starting and stopping the proxy container and the proxy components
- Checking the availability of the proxy

To start and stop the proxy container there are also scripts available. On Windows systems, these are available in the proxy container program group.

In addition, the Management Console provides a command line interface (MC-CLI) for the script-based automation of administration functions. You can find more detailed information in Chapter 9, "Command Line Interface of the BeanConnect Management Console (MC-CLI)".

Administration of proxies running on a remote system is only possible using the Management Console.

This chapter deals with the following topics:

- Administering a BeanConnect proxy via the Management Console
- Administering a BeanConnect proxy container on command level
- Starting an MC-CmdHandler as a service on Windows systems
- Checking the availability of a proxy
- Administering the openUTM-LU62 Gateway
- Administering the communication service
- Monitoring the resource adapter with the Management Console

8.1 Administering a BeanConnect proxy via the Management Console

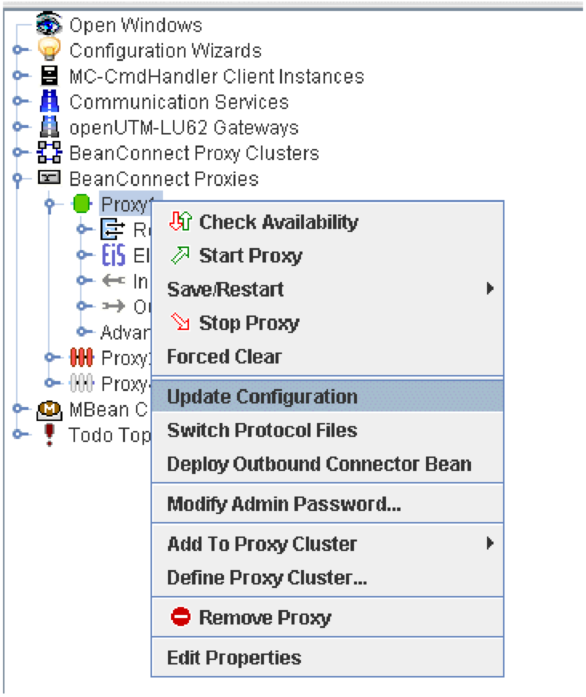
The Management Console is able to administer a number of installed proxies. These proxies can be installed either on the same computer as the Management Console, in which case they are referred to as local proxies, or they can be installed on a different computer, in which case they are referred to as remote proxies.

Proxies can be administered via the Management Console if one of the following conditions is met:

- The proxy is local from the point of view of the Management Console. and runs under the same user ID as the Management Console.
- The proxy is a (possibly) remote proxy whose associated MC-CmdHandler is available and can be accessed via the Management Console.

The Management Console offers the necessary administration functions in the context menu of a proxy node.

Figure 45: Proxy context menu



8.1.1 Starting a proxy

The proxy container or proxy component must be configured before you can start it successfully.

You start a proxy by selecting the menu item **Start Proxy** in the context menu of the relevant proxy node in the navigation tree.

If the proxy is only configured for partners of type openUTM then it is started immediately.

If the proxy is (also) configured for partners of type CICS, a selection dialog box then appears which allows you to select those components of the proxy which you want to be started:

- Proxy container
- openUTM-LU62 Gateway
- Communication Service

> Before you start openUTM-LU62 and the communication service, the associated MC-CmdHandler must already be started.

If you are starting the openUTM-LU62 Gateway (CICS) you can also select the option **Cold Start**. All restart information is lost during a cold start.

Before starting a proxy, the Management Console first checks whether the proxy container or the proxy component is already available.

The proxy container or the proxy component must be configured before it can be started successfully.

The openUTM-LU62 Gateway and the Communication Service cannot be started until at least one EIS partner of type CICS has been defined for the proxy. For more information, refer to the section "Configuring the BeanConnect proxy" on page 173.

> For CICS partners:
> To administer the Communication Service the administration user must be member of the predefined SNA administrator group `sna`. The user group `sna` is present after the installation of the communication service. The administrator is the user under whose user ID, the MC-CmdHandler was started.

The Management Console issues appropriate messages in an action dialog box in which you can monitor the actions and results. You can

- identify what actions have already been carried out and control the results of the completed actions,
- display detailed information on the results if an error occurs,
- if necessary, cancel the entire operation. In this case, only the execution of the subactions that have not yet been performed is canceled. Subactions that have already been executed are not undone.

If an availability check has been performed, the status icons in front of the relevant node in the navigation tree are colored depending on result (green if the proxy container or, in the case of CICS partners, the proxy component is running or red if not running). In the case of openUTM proxies, the icon consists of one part and describes the status of the proxy container. In the case of CICS proxies, the icon consists of three parts.

The individual icons stand for the following components:

Left	Proxy container
Middle	openUTM-LU62 Gateway
Right	Communication Service

An availability check is performed automatically at predefined time intervals or can be started manually (see Section 8.6, "Checking the availability of BeanConnect proxies").

To permit communication between the application server and the EIS partner, all the proxy components must be active (green).

In the case of CICS partners, this is not always sufficient, i.e. the communication may not function properly even though the icons are green. In such cases, you must also check the established connections and opened sessions, see Section 13.9.2, "Diagnosis information for the openUTM-LU62 Gateway"

Starting the proxy container as a Windows service

On Windows systems, you can also start a proxy container as a Windows service using the Management Console:

1. Select the proxy in the navigation area of the Management Console.
2. From the context menu, select the entry **Edit Properties**, open the **General** tab and select the option **Start as Service**.
3. Save the configuration with **Save/Restart** - **Save**. The next time the proxy is started, it is started as a Windows service. If the proxy is already running then this change does not take effect with the proposed restart. Instead, it only becomes effective when the proxy is shut down and then restarted

Further information can be found in section Section 8.2.1.3, "Starting as a Windows service".

8.1.2 Restarting a proxy

Restarting the proxy involves a combination of stopping (if necessary) and starting the proxy or individual proxy components. This is necessary after certain configuration activities (see Section 6.9, "Saving and activating the configuration of the BeanConnect proxy").

Restart a proxy by selecting the menu item **Save/Restart – Restart Proxy** in the context menu of the relevant proxy node in the navigation tree.

In the case of a proxy that is configured for CICS partners, a dialog box is displayed where you can select the proxy components to be restarted. Again, the action dialog box is available for monitoring the actions and results.

The proxy container is reloaded, i.e. the individual processes are stopped and restarted in sequence. The proxy container as a whole remains available.

In the case of a proxy that is configured for CICS partners, the openUTM-LU62 Gateway and communication service are first stopped and then restarted.

8.1.3 Stopping a proxy

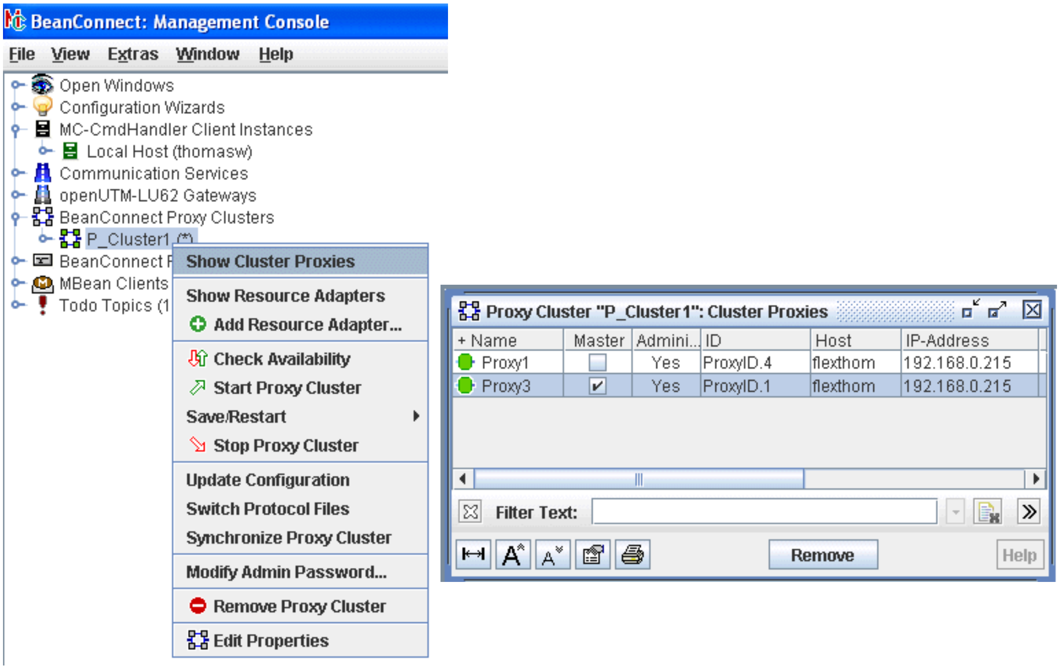
Stop a proxy by selecting the menu item **Stop Proxy** in the context menu of the relevant proxy node in the navigation tree.

In the case of a proxy that is configured for CICS partners, here again, in the same way as when starting the proxy, you can select single proxy components. The Management Console displays an action dialog box dialog box in which you can monitor the actions and results.

8.1.4 Special characteristics in cluster operation

The administration of a proxy cluster resembles that of an individual proxy. The Management Console provides the necessary administration functions via the proxy cluster's context menu. You can use the **Show Cluster Proxies** command to display all the proxies in the cluster.

Figure 46: Administering a BeanConnect proxy cluster



+ Name	Master	Admini...	ID	Host	IP-Address
Proxy1	□	Yes	ProxyID.4	flexthom	192.168.0.215
Proxy3	☑	Yes	ProxyID.1	flexthom	192.168.0.215

Start, stop and save operations always apply to all the proxies in the proxy cluster.

If you want to start or stop an individual proxy, select it in the **Cluster Proxies** panel and choose the corresponding command from the context menu.

You should use MBeans if you want to switch a running resource adapter to a different proxy, see section "Switching a resource adapter in the cluster to another proxy".

The administration of a proxy cluster differs from that of an individual proxy in the following ways:

- If the proxy cluster is made up of a number of proxies, one proxy is always the master proxy. This is indicated accordingly in the **Master** column of the **Cluster Proxies** list. This proxy is the (first) point of contact for the Management Console when fetching the configuration data for the cluster. If there are any changes, the Management Console ensures that these are made in all the proxies.
- If it is not possible to administer one of the proxies for a period then it is possible that this proxies data will not be consistent. In such cases, you can then use the **Synchronize Proxy Cluster** command in the proxy cluster's context menu to synchronize such proxies. When you do this, the other proxies take over the data from the master proxy.

8.2 Administering a BeanConnect proxy container on command level

BeanConnect provides some scripts and programs for administering the proxy container which you can use on command level or via the proxy container program group on Windows systems.

8.2.1 Starting a proxy container

You can start a local proxy container using a script.

On Windows systems you can also use the proxy container program group or start the proxy container as a Windows service.

8.2.1.1 Starting via a script

You can use the following scripts to start a local proxy container.

The procedure `startcontainer` in the subdirectory `shsc` of the proxy container home directory is available for starting a local proxy container.

Proceed as follows:

1. Open a shell or a DOS command prompt window.
2. Change to the proxy container home directory.
3. Call the script as follows:

   `shsc/startcontainer.sh` (under Solaris/Linux systems) or

   `shsc\startcontainer.cmd` (under Windows systems)

8.2.1.2 Starting using the proxy container program group under Windows

You can also start a local proxy container on a Windows system using the proxy container program group:

- From the program group **BeanConnect V3.0A00 - Proxy <proxy_cont_name>**, select the command **Proxy Container Startup - <proxy_cont_name>**.

8.2.1.3 Starting as a Windows service

If BeanConnect is installed under Windows, the proxy container is set up as a service with the name `BeanConnect30A00 <proxy_cont_name>`.
By default, the service has the startup type `manual`. If required, you can set the startup type to `automatic` to ensure that the proxy container is always available.

To start a local proxy container as a service:

1. From the program group **Start** - **Settings** - **Control Panel**, select the entry **Administrative Tools** - **Services**.
2. Select the **Start** command from the context menu of the service **BeanConnect3.0A00 <proxy_cont_name>**.

You can also start a proxy container as a service using the Management Console (see "Starting a proxy" on page 257).

When the proxy container executes as a service, its first output to stdout is written to the file `utmp.out` and its first output to stderr is written to the file `utmp.err`.
If the log files are switched then the files are named `utmp.err.<timestamp>` and `utmp.out.<timestamp>`. The log files for the last application run are automatically saved in the directory `out-err`.

If you start the proxy container as a service, no DOS window is opened for output to stdout or for error messages. You can, however, open windows specially for this purpose. Output is written during operation and is updated automatically (see Chapter 13, "Logging, diagnostics and troubleshooting" on page 459).

8.2.1.4 Starting after abnormal termination of a proxy container run

If the proxy container cannot be started, for instance because the previous proxy container run was terminated abnormally, proceed as follows:

- On Solaris/Linux systems, switch to the proxy container home directory and call the script `shsc/remove.sh`.
- On Windows systems, select **Forced Clear** from the program group **BeanConnect V3.0A00 - Proxy <proxy_cont_name>**.
  or
  switch to the proxy container home directory and call the script `shsc/remove.cmd`.
- If you are operating the Management Console in expert mode (in Solaris, Linux or Windows systems), choose **Forced Clear** from the proxy's context menu.

8.2.2 Restarting a proxy container

It may be necessary to restart the proxy container after certain configuration activities (see "Saving and activating the configuration of the BeanConnect proxy" on page 233).

8.2.2.1 Restarting using a script

You can use the following scripts to restart a local proxy container.

The procedure `change` in the subdirectory `shsc` of the proxy container home directory is available for restarting a local proxy container.

Proceed as follows:

1. Open a shell or a DOS command prompt window.
2. Switch to the proxy container home directory.
3. Call the script as follows:

   `shsc/change.sh` (under Solaris/Linux) or

   `shsc\change.cmd` (under Windows)

8.2.2.2 Restarting using the proxy container program group under Windows

You can restart a local proxy container on a Windows system using the proxy container program group:

- From the program group **BeanConnect VV3.0A00 - Proxy <proxy_cont_name>**, select the command **Proxy Container Restart**.

8.2.3 Stopping a proxy container

You can stop a local proxy container using a script or using the proxy container program group on a Windows system.

8.2.3.1 Stopping using a local script

You can use the following scripts for stopping the proxy container.

The procedure `shutcontainer` in the subdirectory `shsc` of the proxy container home directory is available for stopping a local proxy container.

Proceed as follows:

1. Open a shell or a DOS command prompt window.
2. Switch to the proxy container home directory
3. Call the script as follows:

   `shsc/shutcontainer.sh` (under Solaris/Linux) or

   `shsc\shutcontainer.cmd` (under Windows)

8.2.3.2 Stopping using the proxy container program group under Windows

You can also stop a local proxy container on a Windows system using the proxy container program group:

- From the program group **BeanConnect V3.0A00 - Proxy <proxy_cont_name>**, select the command **Proxy Container Shutdown**.

8.2.3.3 Stopping as a Windows service

If a proxy container was started as a service on a Windows system, you can stop it in the **Services** dialog box as follows.

1. From the program group **Start** - **Settings** - **Control Panel**, select the entry **Administrative Tools** - **Services**.
2. Select **Stop** from the context menu of the service **BeanConnect V3.0A00 <proxy_cont_name>**.

ℹ If the proxy container was started as service with the start type `automatic`, it can only be stopped in this way.

8.3 Starting an MC-CmdHandler as a service on Windows systems

If the MC-CmdHandler is to be started as a service on Windows systems then it must always first be configured as a service, see "Configuring an MC-CmdHandler as a service on Windows systems" on page 238. After this, the service has the autostart type `Manual`.

Use the Control Panel to set the autostart type to `Automatic`. This causes the service to be started automatically when the Windows system is started.

8.4 Administering the openUTM-LU62 Gateway

The openUTM-LU62 Gateway can be administered by the Management Console (see Section 8.1, "Administering a BeanConnect proxy via the Management Console") or directly using scripts. On Solaris and Linux systems you can call all administration commands from a shell as described below.

ℹ After installation of the openUTM-LU62 Gateway on Solaris or Linux systems, all users are authorized to administer it. If for security reasons you want to restrict the administration rights to certain users, proceed as follows:

In the openUTM-LU62 Gateway home directory (default: `/opt/lib/utmlu62`), you will find the file `u62_users`. In this file you can configure a list of users who are to have administration permission. User names must be separated by blank, tab, comma or new line.

If the list defined in the `u62_users` file is not empty, administration access will be denied for all users not contained in the list (except `root`, which always has administration permission).

Under Windows it is recommended that you open a DOS command prompt window with

Start - Programs - openUTM-LU62 - command prompt

From that DOS command prompt window, you can call the commands described below without the prefix `<LU62_home>/`.

8.4.1 Starting the openUTM-LU62 Gateway

The following command starts openUTM-LU62 as a background process, where `<LU62_home>` refers to the installation directory of the openUTM-LU62 Gateway:

```
<LU62_home>/u62_start
```

8.4.2 Stopping the openUTM-LU62 Gateway

The following command stops the openUTM-LU62 Gateway, where `<LU62_home>` refers to the installation directory of the openUTM-LU62 Gateway:

```
<LU62_home>/u62_adm -e
```

Under Windows you can select the following entry from the program group instead:

Start - Programs - openUTM-LU62 - Stop openUTM-LU62

8.4.3 Displaying status information on the openUTM-LU62

The following command displays information on the current status of the openUTM-LU62 Gateway:

```
<LU62_home>/u62_sta
```

Under Windows you can select the following entry from the program group instead:

Start - Programs - openUTM-LU62 - status information

8.5 Administering the communication service

This section describes how to start the SNA daemon and how you start the communication service (SNAP-IX in Solaris systems or IBM Communications Server in Linux and Windows systems) in a command line.

For further information on administering SNAP-IX or the IBM Communications Server please refer to the manufacturer's documentation.

8.5.1 Starting and stopping the SNA daemon (Linux and Solaris systems)

The SNA daemon must be running to administer the Communication Service. If not, an error message is output.

Starting the SNA daemon

- Switch to the directory `/opt/sna/bin` (Solaris) systems or `/opt/ibm/sna/bin` (Linux) systems in the system on which the communication service was installed.
- Enter the command `./sna start`.

Stopping the SNA daemon

- Switch to the directory `/opt/sna/bin` (Solaris systems) or `/opt/ibm/sna/bin` (Linux systems) in the system on which the communication service was installed.
- Enter the command`./sna stop`.

8.5.2 Starting and stopping a communication service in a command line (Linux and Solaris systems)

In Linux and Solaris systems, the BeanConnect Management Console generates the scripts `cs-start-all.sh` and `cs-stop.sh` for a BeanConnect proxy every time the communication service's configuration changes. These scripts are generated on saving and make it possible to start or stop the proxy component in a command line.

If all the proxy components are located on the same host then the scripts are located in the directory `<Proxy_Home>/shsc`, where `<Proxy_Home>` is the proxy container's home directory.

In contrast, if the openUTM-LU62 Gateway and communication service are running on separate hosts then the script is made available in the MC-CmdHandler which the Management Console uses for the administration of these proxy components. In this case, the script is located in the directory `<MC-CmdHandler_Home>/shsc`, where `<MC-CmdHandler_Home>` is the home directory of the MC-CmdHandler.

Starting the communication service

Proceed as follows to start the communication service in a command line:

- Open a shell on the host on which the communication service is installed.
- Go to the proxy container's home directory (installation on proxy host) or to the MC-CmdHandler's home directory (installation on separate host).
- Call the script:

```
shsc/cs-start-all.sh
```

Stopping the communication service

Proceed as follows to stop the communication service in a command line:

- Open a shell on the host on which the communication service is installed.
- Go to the proxy container's home directory (installation on proxy host) or to the MC-CmdHandler's home directory (installation on separate host).
- Call the script:

```
shsc/cs-stop.sh
```

8.6 Checking the availability of BeanConnect proxies

The availability of BeanConnect proxies refers to the availability of the proxy containers (including the components required for CICS) and their communication partners (EIS partners, resource adapters, MC-CmdHandlers). The availability check can take the form of a full check or an individual component or partner-specific check.

The following options are available:

- Checking the availability of a proxy
- Checking the availability of a BeanConnect resource adapter
- Checking the availability of an openUTM-LU62 Gateway and a communication service
- Checking the availability of an MC-CmdHandler
- Checking the availability of an EIS partner

> i You can also check the availability of a proxy cluster. For details, see "Special characteristics of proxy clusters" on page 273.

8.6.1 Checking the availability of a proxy

You can check the availability of a proxy (and its communication partners) in the Management Console by selecting **Check Availability** from the context menu of the relevant proxy node in the navigation tree.

You can also force an automatic availability check to be carried out at predefined time intervals by setting the parameter **Automatic Availability Check** in the proxy's property sheet.

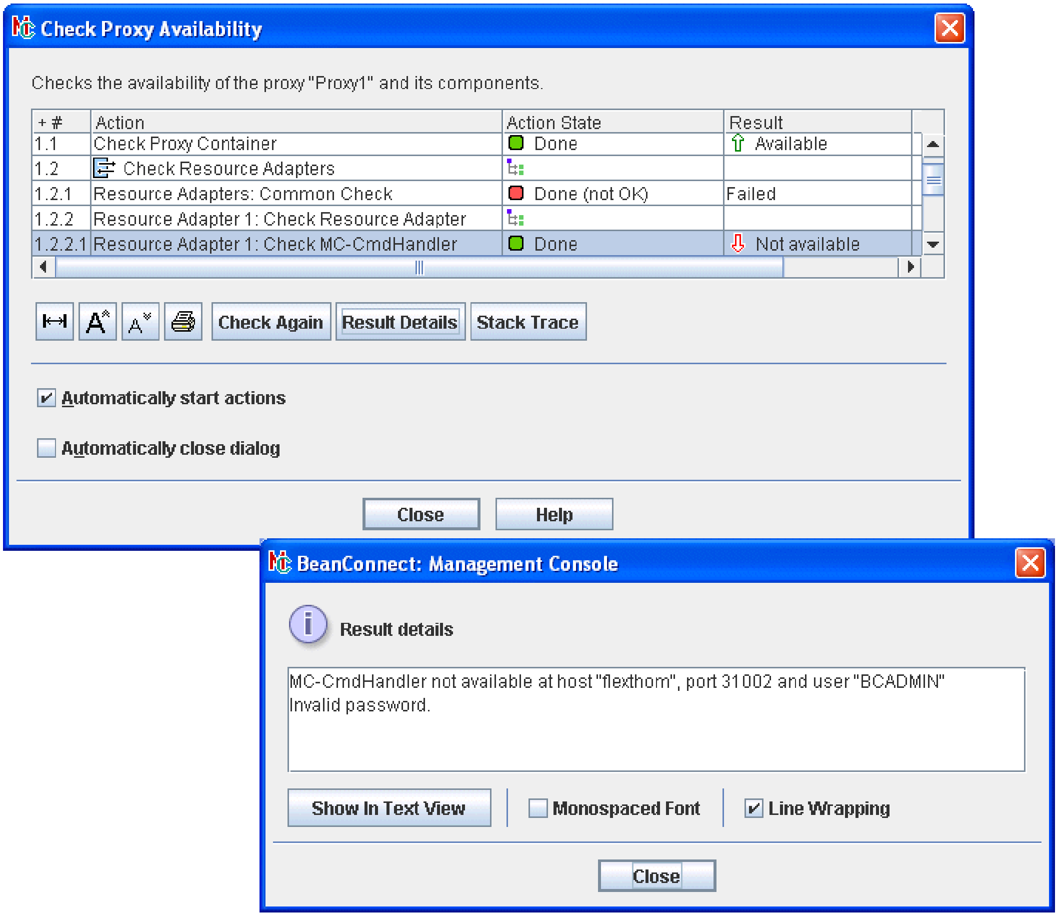
The Management Console checks the availability of:

- Proxy containers
- All the resource adapters assigned to the proxy
- All the MC-CmdHandlers
- The openUTM-LU62 Gateway and communication service if the proxy is configured for CICS partners
- All EIS partners

The proxy container must be running to check the availability of the resource adapter and the EIS partners. In the case of CICS partners, the openUTM-LU62 Gateway and the communication service must also be running.

The Management Console displays an action dialog box in which you can monitor the actions and results.

Figure 47: Checking the availability of a BeanConnect proxy



Checks the availability of the proxy "Proxy1" and its components.

+ #	Action	Action State	Result
1.1	Check Proxy Container	🟩 Done	⇧ Available
1.2	Check Resource Adapters		
1.2.1	Resource Adapters: Common Check	🟥 Done (not OK)	Failed
1.2.2	Resource Adapter 1: Check Resource Adapter		
1.2.2.1	Resource Adapter 1: Check MC-CmdHandler	🟩 Done	⇩ Not available

☑ Automatically start actions

□ Automatically close dialog

Result details

MC-CmdHandler not available at host "flexthom", port 31002 and user "BCADMIN"
Invalid password.

□ Monospaced Font ☑ Line Wrapping

If one of the components is unavailable, select the entry and click **Result Details** to output detailed information on the results of the check. This information can be useful for diagnosis.

When configuring a proxy, you can define an interval for regular availability checks.

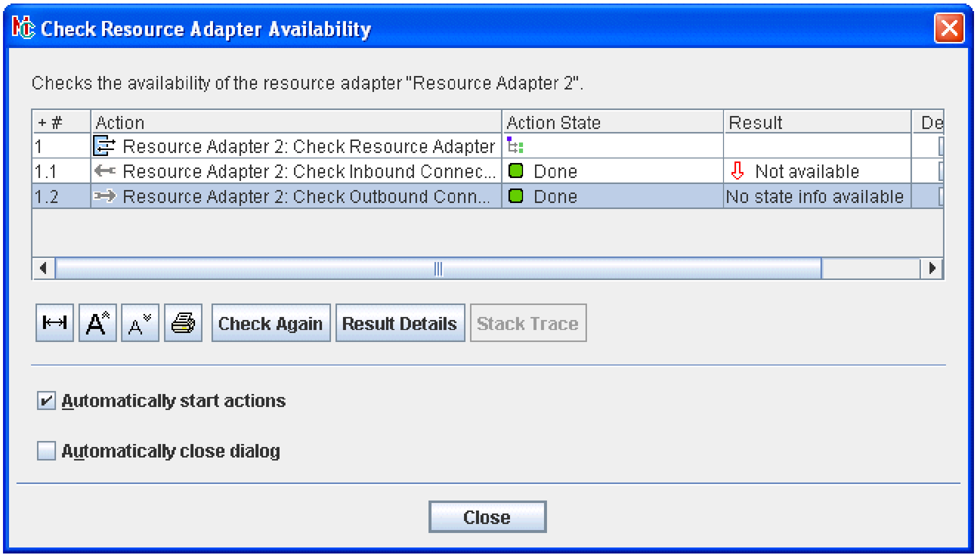
Special characteristics of proxy clusters

- To check the availability of all the components in a cluster, choose the **Check Availability** command from the proxy cluster's context menu.
- To check the availability of an individual proxy in the cluster, select the proxy in the **Cluster Proxies** panel and choose the **Check Availability** command in the context menu.

8.6.2 Checking the availability of a BeanConnect resource adapter

To check the availability of an individual resource adapter in the Management Console, open the resource adapter's context menu in the navigation tree and choose the **Check Availability** command. The associated proxy container must be running.

Figure 48: Checking the availability of a resource adapter



Check Resource Adapter Availability

Checks the availability of the resource adapter "Resource Adapter 2".

+ #	Action	Action State	Result	De
1	Resource Adapter 2: Check Resource Adapter			
1.1	Resource Adapter 2: Check Inbound Connec...	🟩 Done	Not available	
1.2	Resource Adapter 2: Check Outbound Conn...	🟩 Done	No state info available	

☑ Automatically start actions

□ Automatically close dialog

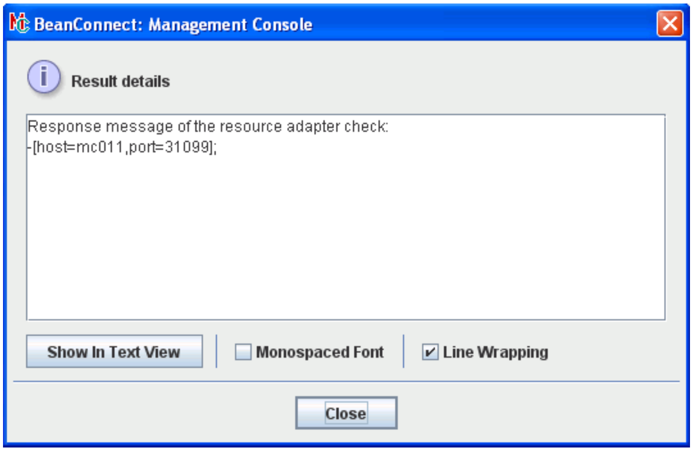
Select a line and click **Result Details** to obtain the detailed results of the check.

For each configured resource adapter, the detailed results of the check contain a result string in the following form:

-/+[host:hostname,port:portnumber]

The sign (+/-) indicates the availability:
+ means available
- means not available

Figure 49: Checking the availability of a resource adapter - result details

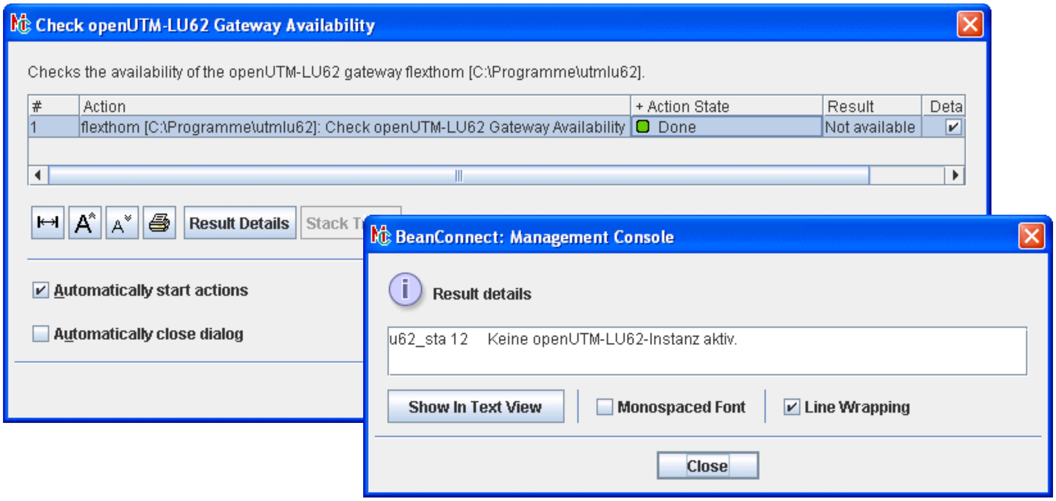


Response message of the resource adapter check:
-[host=mc011,port=31099];

8.6.3 Checking the availability of an openUTM-LU62 Gateway and a communication service

To check the availability of an openUTM-LU62 Gateway and a communication service in the Management Console, click the **openUTM-LU62 Gateway** or **Communication Services** at the topmost level and choose the **Check Availability** command in the context menu. The associated MC-CmdHandler must be running when you perform the check. This also applies if the component is running on the same host as the proxy.

Figure 50: Checking the availability of an openUTM-LU62 Gateway



Check openUTM-LU62 Gateway Availability

Checks the availability of the openUTM-LU62 gateway flexthom [C:\Programme\utmlu62].

#	Action	+ Action State	Result	Deta
1	flexthom [C:\Programme\utmlu62]: Check openUTM-LU62 Gateway Availability	🟩 Done	Not available	☑

☑ Automatically start actions

□ Automatically close dialog

BeanConnect: Management Console

Result details

u62_sta 12   Keine openUTM-LU62-Instanz aktiv.

□ Monospaced Font

☑ Line Wrapping

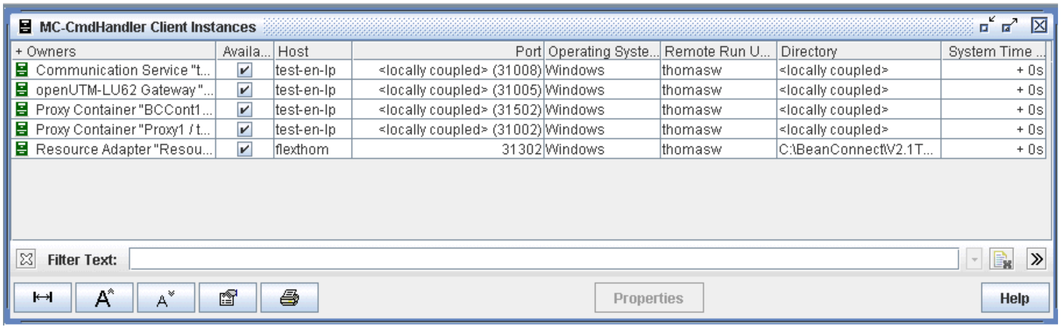
Click **Result Details** to obtain the detailed results of the check as illustrated in the figure.

8.6.4 Checking the availability of an MC-CmdHandler

8.6.4.1 Checking the availability of the MC-CmdHandler with the Management Console

You can use the Management Console to check the availability of an MC-CmdHandler client instance. To do this, click the corresponding node at the topmost level and choose the **Show MC-CmdHandler Client Instances** command in the context menu to open the **MC-CmdHandler Client Instances** panel.

Figure 51: Availability of an MC-CmdHandler instance



MC-CmdHandler Client Instances

+ Owners	Availa...	Host	Port	Operating Syste...	Remote Run U...	Directory	System Time ...
Communication Service "t...	☑	test-en-lp	<locally coupled> (31008)	Windows	thomasw	<locally coupled>	+ 0s
openUTM-LU62 Gateway "...	☑	test-en-lp	<locally coupled> (31005)	Windows	thomasw	<locally coupled>	+ 0s
Proxy Container "BCCont1...	☑	test-en-lp	<locally coupled> (31502)	Windows	thomasw	<locally coupled>	+ 0s
Proxy Container "Proxy1 / t...	☑	test-en-lp	<locally coupled> (31002)	Windows	thomasw	<locally coupled>	+ 0s
Resource Adapter "Resou...	☑	flexthom	31302	Windows	thomasw	C:\BeanConnectV2.1T...	+ 0s

Filter Text:

The **Availability** column in this table indicates whether or not the MC-CmdHandler is available.

To check the availability of a remote MC-CmdHandler, select the relevant line and choose the **Check Availability** command in the context menu.

"Internal" MC-CmdHandlers are identified as **<locally coupled>** in the list. Their availability cannot be checked since they are implicitly available.

8.6.4.2 Checking the availability of the MC-CmdHandler in the command line

Solaris and Linux systems

You use the following script in the proxy container's home directory to check whether the MC-CmdHandler is running:

- `shsc/checkmccmdhandler.sh`

In the case of a stand-alone MC-CmdHandler, the scripts are located in the directory `shsc` below the MC-CmdHandler's installation directory.

Windows systems

Check whether the MC-CmdHandler is running by choosing the command by opening the **Start** menu and choosing the command **MC-CmdHandler Check**
in the program group **BeanConnect <version> - Proxy <container> - MC-CmdHandler**.

You can also check the availability of the MC-CmdHandlers by running the following script located in the proxy container's home directory:

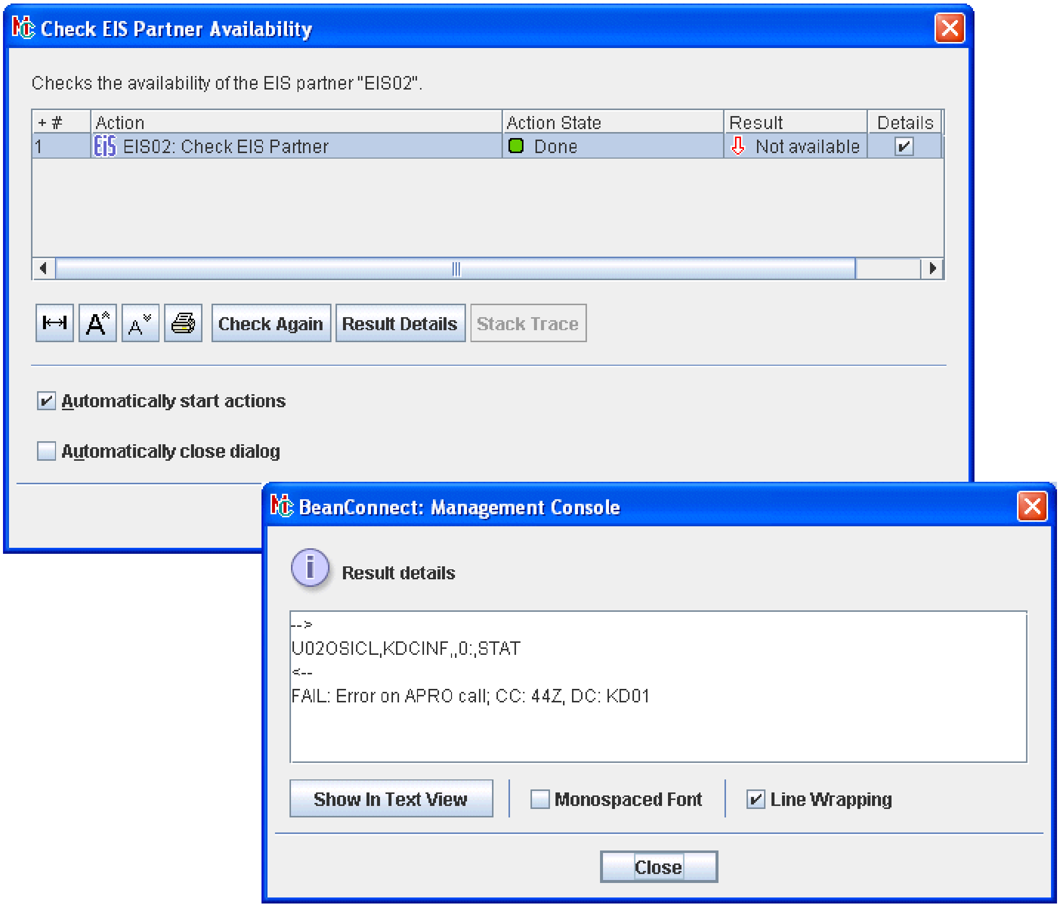
- `shsc\checkmccmdhandler.cmd`

8.6.5 Checking the availability of an EIS partner

You can only check the availability of an EIS partner if a service was specified in the EIS when the EIS partner was configured. This service is called when the availability check is performed and the service's output message is output by the Management Console when you choose the **Result Details** command.

To check the availability of an EIS partner in the Management Console, open the EIS partner's context menu in the navigation tree and choose the **Check Availability** command. The associated proxy container must be running. In the case of CICS partners, the openUTM-LU62 Gateway and the communication service must also be running.

Figure 52: Checking the availability of an EIS partner



Check EIS Partner Availability

Checks the availability of the EIS partner "EIS02".

+ #	Action	Action State	Result	Details
1	EIS02: Check EIS Partner	🟩 Done	Not available	☑

☑ Automatically start actions

□ Automatically close dialog

BeanConnect: Management Console

Result details

```
-->
U02OSICL,KDCINF,,0:,STAT
<--
FAIL: Error on APRO call; CC: 44Z, DC: KD01
```

□ Monospaced Font

☑ Line Wrapping

Select a line and click **Result Details** to view the EIS output message. If an error occurs, you will see diagnostic information as illustrated in the example.

8.7 Monitoring the resource adapter with the Management Console

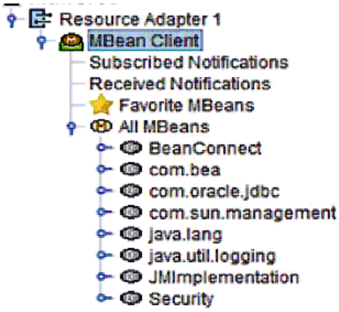
The resource adapter can be monitored via MBean clients. For this to be possible, an MBean client must be configured in the Management Console, see Section 6.11, "Configuring the Management Console as a JMX client".

The MBean client has a fixed assignment to a resource adapter and is displayed in the resource adapter's tree.

It is also possible to define a stand-alone MBean client which is not assigned to any resource adapter.

For further information, see section "Setting up free JMX clients" on page 244.

Figure 53: Navigation tree for an MBean



The following nodes are present below the **MBean Client** node:

Subscribed Notifications

Permits access to all subscribed notifications. You must perform subscription explicitly.

Received Notifications

Permits access to all received notifications. If notifications have been received then the node is displayed in bold. It is followed by the number of received notifications in brackets.

Statistics Collectors

Permits access to the statistics collectors. This node is only displayed if statistics collectors have been configured, see Section 8.7.4, "Collecting and displaying diagnostic values".

Favorite MBeans

Indicates the MBeans that have been included in the list of favorites. The list of favorites provides a clearer overview by providing a separate depiction of the frequently used MBeans. You can define favorites by choosing **Add to Favorites** in the relevant context menu.

All Beans

Permits access to all the MBeans including the MBeans listed under **Favorite MBeans**.

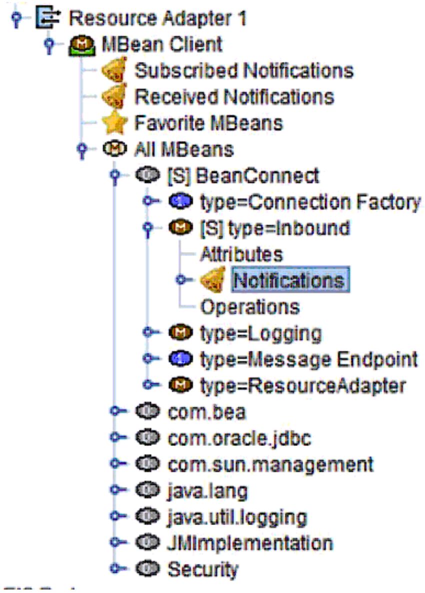
8.7.1 Establishing a connection to the MBean server

Most actions are only possible if there is a connection to the MBean server. If no such connection has so far been established, choose **Connect To MBean Server** from the context menu of the MBean client.

8.7.2 Displaying MBean object names

When you open the **All MBeans** or **Favorite MBeans** nodes, all the defined MBean domains or, alternatively, all the domains present in the favorites are displayed. You can then open the individual MBean domains to see the MBean object names. Alternatively, you can choose the **Show MBeans** command in the context menu.

Figure 54: MBean - object name subtree



- Resource Adapter 1
  - MBean Client
    - Subscribed Notifications
    - Received Notifications
    - Favorite MBeans
    - All MBeans
      - [S] BeanConnect
        - type=Connection Factory
        - [S] type=Inbound
          - Attributes
          - Notifications
          - Operations
        - type=Logging
        - type=Message Endpoint
        - type=ResourceAdapter
      - com.bea
      - com.oracle.jdbc
      - com.sun.management
      - java.lang
      - java.util.logging
      - JMImplementation
      - Security

For each MBean, you see a list containing the elements **Attributes**, **Notifications** and, if operations are possible for the MBean, **Operations**.

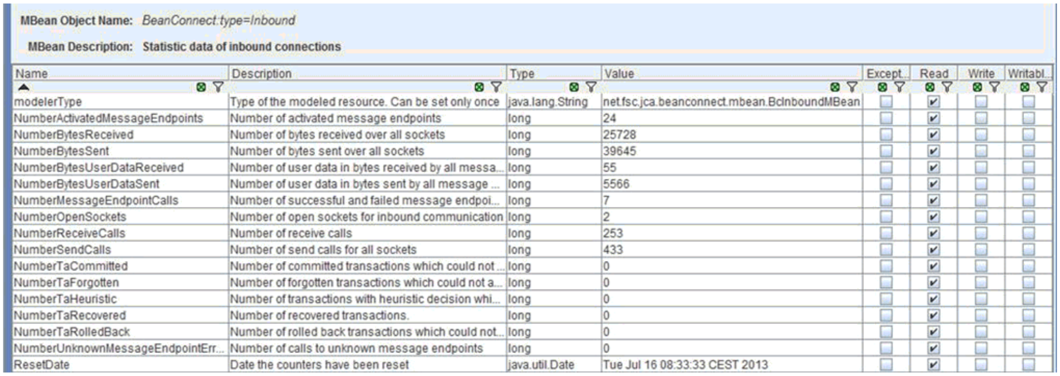
8.7.3 Displaying and modifying MBean attributes

MBeans usually possess extensive lists of attributes. You can output the current values of these attributes via the Management Console. You can also modify some of these attributes.

8.7.3.1 Displaying MBean attributes

To view an MBean's attributes, expand the MBean's node and click **Attributes**. Alternatively, you can choose **Show MBean Attributes** from the context menu.

Figure 55: MBean - attribute table



MBean Object Name: *BeanConnect:type=Inbound*

MBean Description: **Statistic data of inbound connections**

Name	Description	Type	Value	Except...	Read	Write	Writabl...
modelerType	Type of the modeled resource. Can be set only once	java.lang.String	net.fsc.jca.beanconnect.mbean.BcInboundMBean	□	☑	□	□
NumberActivatedMessageEndpoints	Number of activated message endpoints	long	24	□	☑	□	□
NumberBytesReceived	Number of bytes received over all sockets	long	25728	□	☑	□	□
NumberBytesSent	Number of bytes sent over all sockets	long	39645	□	☑	□	□
NumberBytesUserDataReceived	Number of user data in bytes received by all messa...	long	55	□	☑	□	□
NumberBytesUserDataSent	Number of user data in bytes sent by all message ...	long	5566	□	☑	□	□
NumberMessageEndpointCalls	Number of successful and failed message endpoi...	long	7	□	☑	□	□
NumberOpenSockets	Number of open sockets for inbound communication	long	2	□	☑	□	□
NumberReceiveCalls	Number of receive calls	long	253	□	☑	□	□
NumberSendCalls	Number of send calls for all sockets	long	433	□	☑	□	□
NumberTaCommitted	Number of committed transactions which could not ...	long	0	□	☑	□	□
NumberTaForgotten	Number of forgotten transactions which could not a...	long	0	□	☑	□	□
NumberTaHeuristic	Number of transactions with heuristic decision whi...	long	0	□	☑	□	□
NumberTaRecovered	Number of recovered transactions.	long	0	□	☑	□	□
NumberTaRolledBack	Number of rolled back transactions which could not...	long	0	□	☑	□	□
NumberUnknownMessageEndpointErr...	Number of calls to unknown message endpoints	long	0	□	☑	□	□
ResetDate	Date the counters have been reset	java.util.Date	Tue Jul 16 08:33:33 CEST 2013	□	☑	□	□

When you click an attribute in the table to select it, the details are displayed in the lower part of the window. This detailed view is intended for extensive attributes that cannot be displayed in full in the table.

The columns of the table have the following meanings:

Name / Description
Name and more detailed description of the attribute.

Type
Type of attribute value, e.g. string, integer, Boolean.

Value
Value of the attribute.

Exception
Specifies whether the JMX server delivered an exception when determining the value of the attribute If this is the case, you can display the exception in the attribute's detailed view.

Read
Specifies whether the attribute can be queried at the JMX server. This is the case for the majority of attributes.

Write
Specifies whether the attribute values can be modified.

Writable
Specifies whether the attribute value can be modified via the Management Console.

Only attributes with "simple" value types (string, integer, Boolean) can be modified via the Management Console. Attributes with complex (e.g. composite) values cannot usually be modified via the Management Console. These attributes are then identified as **Write** but not as **Writable**.

8.7.3.2 Modifying MBean attribute values

The Management Console allows you to modify all the attributes that are identified as **Writable** in the table. To do this, proceed as follows:

- In the attribute's context menu, choose **Set MBean Attribute Value**. Alternatively, select the attribute and click the **Set Value** button.
- You then edit the values in the following dialog.

The Management Console outputs a message after performing the action. This contains either a confirmation of the change or an error message if it was not possible to perform the modification.

8.7.4 Collecting and displaying diagnostic values

You can create statistics via the Management Console by configuring statistics collectors. The statistics collectors poll the values of the MBean attributes at regular intervals;

8.7.4.1 Configuring, displaying and modifying statistics collectors

You can configure a statistics collector for every attribute of an MBean. The Management Console then generates a node with the name **Statistics Collectors** below the MBean client. All the statistics collectors for the MBean client are displayed below this node.

You set up a statistics collector as follows:

- Display the attributes for the required MBean, see Section 8.7.3.1, "Displaying MBean attributes".
- Select the attribute in the table and choose the command **Collect Attribute Values** in the attribute's context menu.

When you click the **Statistics Collectors** node, a table with all the statistics collectors available for the MBean is output. Alternatively, you can choose **Show Statistics Collectors** from the corresponding node's context menu. You will find more details on the meaning of the table columns in the online Help system.

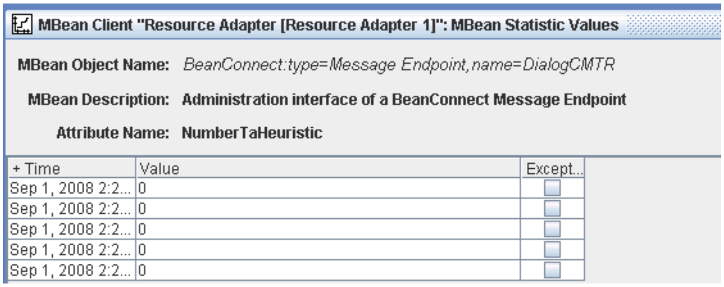
You can modify the data collection interval for each statistics collector. To do this, select the required statistics collector in the table, choose **Edit Properties** in the context menu and then modify the interval in the following dialog. Alternatively, you can click the **Edit** button below the table.

To remove a statistics collector from the table, choose the command **Remove MBean Statistics Collector** from the statistics collector's context menu (alternatively: **Remove** button below the table). Removing a statistics collector also deletes all the data that has been collected by this collector up to the time of deletion.

8.7.4.2 Displaying statistical values

You can view the values for a statistics collector by displaying the table of statistics collectors and choosing the command **Show Statistic Values** in the required collector's context menu. Alternatively, you can select the collector and click the **Show Values** button below the table.

Figure 56: MBean - statistical values



MBean Client "Resource Adapter [Resource Adapter 1]": MBean Statistic Values

MBean Object Name: *BeanConnect:type=Message Endpoint,name=DialogCMTR*

MBean Description: Administration interface of a BeanConnect Message Endpoint

Attribute Name: NumberTaHeuristic

+ Time	Value	Except...
Sep 1, 2008 2:2...	0	☐
Sep 1, 2008 2:2...	0	☐
Sep 1, 2008 2:2...	0	☐
Sep 1, 2008 2:2...	0	☐
Sep 1, 2008 2:2...	0	☐

The table with the collected statistical values contains the three columns **Time**, **Value** and **Exception**, see the online Help system. The **Exception** column outputs any exception which the JMX server may have output when determining the statistical value. When you click a statistical value with the mouse, details such as the complete text of an exception are displayed in the lower part of the window.

8.7.5 Subscribing to and displaying MBean notifications

The MBeans of type **ResourceAdapter**, **ManagedConnectionFactory**, **Inbound** and **MessageEndpoint** issue notifications. These are messages which the resource adapter generates when certain events occur and which can be displayed in the Management Console. Before you can display notifications, you must have explicitly subscribed to these in the Management Console (Subscribe procedure).

The following table indicates the notifications that you can subscribe to:

Type of MBean	Name and meaning of the notification

Resource adapter	`BeanConnect.Started`
	The resource adapter has been started.
	`BeanConnect.Stopped`
	The resource adapter has been stopped.

ManagedConnectionFactory	`BeanConnect.Connection.Error`
	The resource adapter has thrown an exception for a connection.
	`BeanConnect.Transaction.Rollback`
	The connection is involved in a transaction that has been rolled back.
	`BeanConnect.Transaction.Heuristic`
	A heuristic decision has been made for the transaction branch in which the connection is involved.

Inbound	`BeanConnect.MessageEndpoint.Activation` The application server has activated a message endpoint in BeanConnect.
	`BeanConnect.MessageEndpoint.Deactivation` The application server has deactivated a message endpoint in BeanConnect.
	`BeanConnect.MessageEndpoint.Unknown` A message has arrived for an unknown message endpoint.
	`BeanConnect.Transaction.Rollback` A transaction has been rolled back during recovery.
	`BeanConnect.Transaction.Heuristic` A heuristic decision has been made for a transaction during recovery.

Type of MBean	Name and meaning of the notification
MessageEndpoint	`BeanConnect.MessageEndpoint.Error` An error occurred when calling the message endpoint.
	`BeanConnect.MessageEndpoint.Exception` An exception was thrown when calling the message endpoint.
	`BeanConnect.Transaction.Rollback` The message endpoint is involved in a transaction that has been rolled back.
	`BeanConnect.Transaction.Heuristic` A heuristic decision has been made for the transaction branch in which the message endpoint is involved.

8.7.5.1 Subscribing to MBean notifications

You can subscribe to the MBean Notifications either all at once or individually:

- If you want to subscribe to all an MBean's notifications at once, choose the command **Subscribe MBean Notifications** in the MBean node's context menu.
- If you want to subscribe to an individual notification, expand the **Notifications** node in an MBean's subtree. All the notifications to which you can subscribe are then displayed. You should now choose **Subscribe MBean Notifications of this type** in the context menu of the required notification.

If you want to terminate your subscription to a notification, use the command **Unsubscribe MBean Notification** in the context menu of an MBean's **Notifications** node or choose **Unsubscribe MBean Notifications of this type** from the context menu of a specific notification type node.

The settings for the notifications are retained even after the Management Console is shut down. As a result, it is not necessary to resubscribe to notifications for which you already have a subscription each time the Management Console is started. Instead, these notifications are supplied automatically once the connection to the JMX server has been established. However, no notifications generated during the period when the Management Console was not logged in at the MBean server are supplied.

8.7.5.2 Displaying MBean notifications

The Management Console indicates that notifications have been received by displaying the corresponding nodes in bold. The number of notifications is indicated in parentheses next to the node. You can view the MBean notifications in the following ways.

- To view all the notifications from all the MBean clients, click the **Received Notifications** node below the **MBean Client** node or choose the command **Show Received Notifications** in this node's context menu.
- To view all the notifications for a specific MBean, expand the relevant MBean's subtree and click the **Notifications** node. Alternatively: Choose the command **Show Received Notifications** in this node's context menu.
- To view all the notifications of a given type, click a notification type node or choose **Show Received Notifications** in its context menu.

The notifications and associated attributes are listed in table form in a new window. To display detailed information for a notification, click to select the notification or choose the command **Show MBean Attributes** from the context menu. The details are displayed in the lower window area. You will find more information on the meaning of the table columns in the online Help system.

You can delete a notification using the command **Remove MBean Notification** from the context menu. Alternatively, you can select the notification and click the **Remove** button below the table.

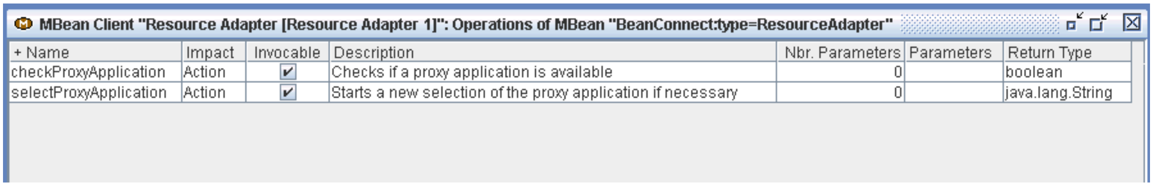
i Received notifications are not saved when the Management Console is shut down and are therefore not displayed in the next session.

8.7.6 Displaying and executing MBean operations

The Management Console allows you to perform operations on MBeans. These are specific actions that are performed in the resource adapter, e.g. resetting statistics counters or checking availability.

Every MBean possesses an **Operations** node. Clicking this node or choosing **Show MBean Operations** from the context menu outputs a table which lists all the operations for the corresponding MBean.

Figure 57: MBean operations



MBean Client "Resource Adapter [Resource Adapter 1]": Operations of MBean "BeanConnect:type=ResourceAdapter"

+ Name	Impact	Invocable	Description	Nbr. Parameters	Parameters	Return Type
checkProxyApplication	Action	☑	Checks if a proxy application is available	0		boolean
selectProxyApplication	Action	☑	Starts a new selection of the proxy application if necessary	0		java.lang.String

When you click an operation, the detailed information is displayed in the window at the bottom.

Operations that you can perform using the Management Console are identified accordingly in the column **Invocable**. In a similar way to when modifying attribute values, the Management Console can only execute operations which have no or only simple parameter types. Proceed as follows to execute this type of operation:

- In the operation's context menu, choose the command **Invoke MBean Operation...**. Alternatively, you can select the operation and click the **Invoke** button below the table.
- Enter the call (invocation) parameters in the subsequent dialog (if necessary). The dialog may consist of several sheets. The operation is not performed until you confirm with **OK**.

The Management Console outputs a message after execution. This contains either the result of the operation or an error message if it was not possible to perform the operation.

Switching a resource adapter in the cluster to another proxy

You can use the MBean operation `selectProxyApplication` to stop a proxy while the cluster is running and simultaneously switch the resource adapter(s) to a different proxy ("soft" switchover). This has the advantage of minimizing negative impacts on operation. In contrast, if you stop a running proxy in a proxy cluster directly via the context menu then the connections administered by this proxy are immediately cleared, with the result, for example, that open transactions are interrupted and may have to be rolled back.

You can avoid this by performing a "soft" switchover. Proceed as follows:

- Shut down the proxy using the openUTM tool `kdcshut` and specify a wait time.

  To do this, open a shell or DOS command window and enter the following command (with "\" in Windows systems):

  ```
  <openUTM-Server_home>/ex/kdcshut  <Proxy_home>  time
  ```

  Here, `<openUTM-Server_home>` is the openUTM installation directory, `<Proxy_home>` is the fully qualified path name of the proxy and `time` is the wait time in minutes (recommended value: at least 10 minutes). On the one hand, this call prevents any new connections from being established and, on the other, it stops the proxy from being shut down immediately.

- Immediately after this in the Management Console, call the resource adapter MBean in a resource adapter instance that is operating with the proxy that you want to shut down. In this MBean, call the operation `selectProxyApplication` in order to assign a different proxy to the resource adapter instance. BeanConnect selects the new proxy automatically using internal algorithms.

  If the application server is configured as a cluster, you must perform this operation for every resource adapter instance in the application server cluster that is assigned to the proxy that you want to shut down. The resource adapter MBean's `CurrentProxyUrl` attribute indicates the proxy to which a resource adapter instance is assigned.

- Once you have switched the resource adapter to the new proxy, you can shut down the earlier proxy by calling `kdcshut` again and specifying a short wait time (e.g. 5 minutes):

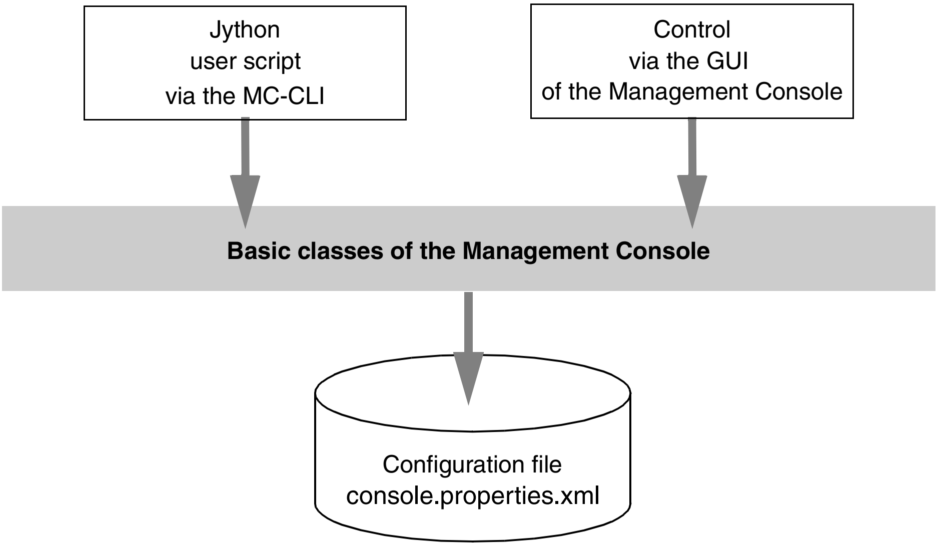
  ```
  <openUTM-Server_home>/ex/kdcshut  <Proxy_home> 5 G
  ```

  The parameters `5` and `G` cause the proxy to be shut down once all the connections have been cleared but at the latest at the end of the 5 minute wait time.

9 Command Line Interface of the BeanConnect Management Console (MC-CLI)

With the **M**anagement **C**onsole **C**ommand **L**ine **I**nterface (abbreviated to MC-CLI in the following), BeanConnect provides a set of Jython functions which allow you to start BeanConnect Management Console functions from a Jython script. It makes sense to use the MC-CLI, for example, whenever large volumes of data have to be configured or frequently recurring administration tasks have to be performed (see also Section 9.5, "Application scenarios (examples)").

Figure 58: Command Line Interface (CLI) and the Management Console graphical user interface (GUI)



You can process the same configuration sequentially with both the BeanConnect Management Console GUI and with MC-CLI scripts. In both cases, the configuration is represented by one and the same configuration file named `console.properties.xml`. It is not possible to configure and administer multiple Management Console settings simultaneously (both via the GUI and using MC-CLI).

9.1 Overview of MC-CLI

All proxy objects and proxy cluster objects, i.e. proxies, proxy cluster, resource adapters, EIS partners, inbound users, inbound services, outbound services, inbound message endpoints and outbound communication endpoints can be configured and administered via CLI user scripts.

You can create the above-mentioned objects (except for the proxies), read and modify the properties of the objects, and remove objects from the configuration. You can read the properties of the todo topics and delete todo topics.

You can also run administration functions for the objects. You can, for example, check availability and administrability or start and stop proxies or proxy clusters.

MC-CLI modules and functions

The MC-CLI interface consists of a number of modules each of which contains a group of functions:

- The module `BcAdminMain` contains functions that are used to start and exit a Management Console session at the MC-CLI.

  The parameter `console_home` is used to access the central file `console.properties.xml` which contains the configuration data.

- The module `BcAdminAction` contains functions which analyze the results of an administration call and return information about the event and all the subactions.

- A module for each object type that can be administered and configured via the MC-CLI. Each of these modules contains the functions required to administer and configure the corresponding object type. The available modules are listed in the following table.

BcAdminProxy	Functions for configuring and administering a proxy
BcAdminProxyCluster	Functions for configuring and administering a proxy cluster
BcAdminRa	Functions for configuring and administering a resource adapter
BcAdminEisPartner	Functions for configuring and administering an EIS partner
BcAdminInboundService	Functions for configuring an inbound service
BcAdminInboundMsgEndpoint	Functions for configuring an inbound message endpoint
BcAdminInboundUser	Functions for configuring an inbound user
BcAdminOutboundService	Functions for configuring an outbound service
BcAdminOutboundCommEndpoint	Functions for configuring an outbound communication endpoint
BcAdminTodo	Information functions and functions for handling a todo topic

The modules contain various functions depending on the object type. You will find a list of all the functions present in these modules in the following table.

Section 9.4, "Functions" describes which functions are available for the corresponding object types and which object properties can be read and/or modified.

create()	Add a new object to the configuration
getObject()	Read an existing object from the configuration
authenticate()	Authenticate caller
remove()	Remove an object from the configuration
getProperties()	Read the properties of the object
modifyProperties()	Modify the properties of the object
getList()	Read a list of objects of a given type that are linked to this object
perform()	Start administrative actions, e.g. checkAvailability, start, stop
addProxy()	Add a proxy to the proxy cluster
removeProxy()	Remove a proxy from the proxy cluster
getMasterProxy()	Read master proxy of a proxy cluster
setMasterProxy()	Change the master proxy of a proxy cluster

The exact meanings of the functions and object properties are described in the Management Console's online help system. For more information on the online help system, see Section 5.1.2, "Starting the Management Console's online Help system".

The names of the object properties are mostly identical to those used in the graphical user interface and even when they differ, they are nevertheless unambiguous.

9.2 Creating and calling MC-CLI user scripts

Following installation of the BeanConnect Management Console, the `lib` subdirectory of the BeanConnect installation directory contains the jar file `BeanConnectMcCli.jar` which contains the MC-CLI's Jython modules. This jar file must be present in the `CLASSPATH` when the MC-CLI functions are called.

The product Jython must also be installed on your computer and the installation directory must be specified in `PATH/JYTHONPATH`.

For further information on the product Jython, see the BeanConnect Release Notice.

9.2.1 Prerequisites when calling an MC-CLI user script

To use the Jython modules provided in the MC-CLI, the following conditions must be fulfilled when an MC-CLI user script is started.

The scripts `startBcAdmin.cmd` (Windows) and `startBcAdmin.sh` (Linux and Solaris systems) in the subdirectory `cli-sample` of the Management Console installation directory (see Section 9.5, "Application scenarios (examples)") are available as examples.

- `CLASSPATH` must contain the BeanConnect jar files from the `lib` directory. The `CLASSPATH` is set by the script `javaenv.cmd` (Windows systems) or `javaenv.sh` (Linux and Solaris systems). Call this script with:

  `<console_home>\bin\javaenv.cmd` (Windows) or

  `<console_home>/bin/javaenv.sh` (Unix systems)

  `<console-home>` stands for the Management Console installation directory.

- `JYTHONPATH` must contain the Jython installation directory:

  `set JYTHONPATH=<jython_home>` (Windows) or
  `JYTHONPATH=<jython_home>` (Unix systems)

- PATH must be extended to include the Jython installation directory:

  `set PATH=<jython_home>;%PATH%` (Windows) or
  `PATH=<jython_home>:$PATH` (Linux and Solaris systems)

- You must specify the following VM arguments in the `Jython` command used to start the CLI user script:
  - `-DBEANCONNECTPATH=<beanConnect_lib>`
    `<beanConnect_lib>` is the directory containing the BeanConnect jar files.
  - `-DBEANCONNECT_JDK_HOME=<jdk_home>`
    `<jdk_home>` is the JDK installation directory
  - `-DBEANCONNECT_USERCONS=<Console_home>`
    `<Console_home>` is the Management Console installation directory
  - `-Dlog4jCfgFile=<log4j_properties_file>`
    `<log4j_properties_file>` is the file containing the log4j properties. In most cases, this is the file `log4j.properties.xml` in the `config` subdirectory of the Management Console installation directory.

9.2.2 Preparing the configuration

You cannot use the MC-CLI functions to add proxies to the configuration. The Management Console recognizes newly installed local proxies on start-up and adds these to the configuration automatically. They are therefore available in the CLI.

Remote proxies must first be added to the configuration via the graphical user interface before they can then be configured and administered via the MC-CLI.

9.2.3 Structure of the user script

To use the Jython modules provided in the MC-CLI, you must take account of the following requirements in your MC-CLI script.

- Every MC-CLI script that addresses Java classes, must contain the corresponding `import` statements for the Java classes of the MC-CLI (see Section 9.3, "Java classes"):

```
import com.fujitsu.ts.jca.tools.mc.cli.BcDef     as BcDef
import com.fujitsu.ts.jca.tools.mc.cli.BcParameterException
                                                 as BcParameterException
import com.fujitsu.ts.jca.tools.mc.cli.BcObjectException
                                                 as BcObjectException
import com.fujitsu.ts.jca.tools.mc.cli.BcToolException
                                                  as BcToolException
```

- You must issue an `import` statement `import <mc-cli-module>` for each MC-CLI module whose functions are called in the script
  (e.g. `import BcAdminMain`).

- At the start of the script, you must call the function `BcAdminMain.init()` which starts the Management Console session and reads the configuration file.
  After that, you can call other MC-CLI functions.

- If a proxy or proxy cluster is protected by an administration password then you must perform a successful authentication of this object before calling a function that uses the proxy or proxy cluster as a parameter (see the function "authenticate() – Authenticate for proxy").

- In order for changes to become effective, they must be explicitly saved and integrated in the configuration of the associated component. This is done via the parameter `action` (= "`save`", "`update-config`", "`update-ra-xml`") in the `perform()` function or the parameter `save_all=True` in `BcAdminMain.close()`.

- At the end of the script, it is necessary to call the function `BcAdminMain.close()` in order to exit the Management Console session. Only then it is possible to start other Management Console sessions using the same configuration file.

  If a Management Console session is not exited correctly then the synchronization file `ConsoleInUse.txt` in the Management Console installation directory may not be deleted and all subsequent attempts to start a Management Console session will be aborted. In such cases, this file must be deleted manually.

9.2.4 Specifying call parameters

The call parameters of the MC-CLI functions are usually positional parameters. Some optional parameters are defined as keyword parameters (identified with (kw) below). If the corresponding keyword is not specified when a function is called then the position of the parameter is the determining factor.

When passing the parameters on MC-CLI function calls, you must observe some additional rules depending on the parameter type (objects, properties of the objects):

Objects

At the MC-CLI, the following rules apply when specifying objects:

- In the case of the functions `create()` and `getObject()`, it may be necessary to specify the higher-level `BcObject` of type `BcObjectType.PROXY` or `BcObjectType.PROXY_CLUSTER`.
- The functions `create()` and `getObject()` return a created or read `BcObject` object of type `BcObjectType` as the result.

  This object can then be used to call all the other functions for this object (parameter: `bc_object`).
- The function `getList()` returns a (Jython) dictionary with

  `key`
  Name of the object (property **name**)

  `value`
  Object of type `BcObject`

  Each of these objects can then be used to call all the other functions for this object (parameter: bc_object).

Properties

In the MC-CLI, the following rules apply to properties (parameter `props` in `create()` or `modifyProperties()`, return in `getProperties()`):

- All properties are passed in Jython dictionaries.
- The property names always consist of lowercase letters, numbers, hyphens ('-') and periods ('.').
- If multiple tabs with properties exist in the Management Console graphical user interface, then the property name is prefixed by the name of the relevant tab, e.g. "`utm.`", "`timer.`". Exception: The properties of the **General** tab. These have no prefix.
- Properties for time values have the unit as postfix ("`.sec`" or "`.min`").
- The values of the properties are always specified as strings, also in the case of integer values (e.g. for time values or port numbers).
- If a property is only able to assume certain values then these are defined in the Java class `BcDef`. This is described in greater detail in the sections relating to the individual functions.
- If an MC-CmdHandler is assigned to an object (proxy, resource adapter, etc.) via the graphical user interface during configuration, then only the corresponding properties `admin-port` and possibly also `admin-pw` are specified in the CLI.

9.3 Java classes

The MC-CLI interface uses the Java classes `BcDef`, `BcObjectType`, `BcObject` for the call parameters and return values. For error handling, MC-CLI uses exceptions of the classes `BcObjectException`, `BcParameterException` and `BcToolException`. These Java classes are contained in the package `com.fujitsu.ts.jca.tools.mc.cli`.

This section gives you a brief overview of these classes. A full description of the Java classes can be found in JavaDoc which is stored in the `JavaDoc` subdirectory of the Management Console installation directory.

9.3.1 Class: BcDef

The class `BcDef` describes values of object properties, parameters and returns (action) that are only able to assume specific values. As a result, the values do not have to be set or checked on the basis of user-defined strings.

9.3.2 Class: BcObjectType

The type of an object of class `BcObject` is described via the class `BcObjectType`. The value can be read using the method `toString()`. This value is also the input value for the parameter `list_type` in the `getList()` functions.

The following values are available:

Objects	**toString()**
`BcObjectType.ACTION`	`action`
`BcObjectType.EIS_PARTNER`	`eis-partner`
`BcObjectType.INBOUND_MSG_ENDPOINT`	`inbound-msg-endpoint`
`BcObjectType.INBOUND_SERVICE`	`inbound-service`
`BcObjectType.INBOUND_USER`	`inbound-user`
`BcObjectType.OUTBOUND_COMM_ENDPOINT`	`outbound-comm-endpoint`
`BcObjectType.OUTBOUND_SERVICE`	`outbound-service`
`BcObjectType.PROXY`	`proxy`

Objects	toString()
`BcObjectType.PROXY_CLUSTER`	`proxy-cluster`
`BcObjectType.RESOURCE_ADAPTER`	`resource-adapter`
`BcObjectType.TODO`	`todo`

9.3.3 Class: BcObject

The class `BcObject` represents Management Console objects in the MC-CLI.

The functions `create()` and `getObject()` return the created or read objects of type `BcObject` as their result. In the case of all other functions, the object of type `BcObject` that is to be addressed can be specified as a parameter.

The function `getList()` returns a (Jython) dictionary which in turn contains objects of type `BcObject` as a value.

The following methods are available:

9.3.3.1 getName()

Function: `getName()`

Reads the name of the object

Parameters: None

Return: (String)

Name of the object. Corresponds to the property `name` of the object.

9.3.3.2 getObjectType()

Function: `getObjectType()`

Reads the type of the object

Parameters: None

Return: (`BcObjectType`)

Type of the object, see Section 9.3.2, "Class: BcObjectType".

9.3.4 Exceptions

The MC-CLI functions usually do not return a return code specifying whether the function was executed successfully. If a function cannot be executed then an exception from the package `com.fujitsu.ts.jca.tools.mc.cli` is thrown. If necessary, such exceptions can be caught in the script and processed appropriately.

The following exceptions are thrown:

9.3.4.1 Class: BcObjectException

Meaning:
An object necessary for this function cannot be found or the specified object cannot be used.

Possible causes:

- The Management Console session has not been started or has already been terminated.
- In all functions in which the parameter `proxy_object` was specified:
  - The specified proxy object is invalid.
    Reason: incorrect object type (BcObjectType) or the session in which the object was generated has terminated.

    Or:
  - The proxy object requires an authentication which has not yet been conducted or was conducted unsuccessfully.
- In all functions in which the parameter `object_name/bc_object` has to be specified: The configuration does not contain an object of the required type with the specified name or an invalid object was specified.

Message text:
`"object not given"` or

`"no object <type> <object_name> given"`

or similar.

Solution:

Depending on the message text, you should check

- whether a Management Console session has been started,
- whether the object belongs to the current session,
- whether authentication has been performed for the associated proxy object,
- whether the object is of the right type (BcObject with correct `BcObjectType`) or
- whether the specified object is present in the configuration.

9.3.4.2 Class: BcParameterException

Meaning:

An error occurred during the specification of function parameters or object properties.

Possible causes:

- The specification of this parameter or this property for this function or object type is not permitted. This means that this property does not exist for this object type or cannot be modified.
- The specified value is not permitted for this parameter or for the specified property (not numerical or not in the permitted range of values) or is not permitted in combination with other properties.

Message text:

`"invalid / unchangeable property given"` or

`"invalid value <value> for property <key> given"` or

`"invalid parameter <key> given"`

or similar.

Solution:

Depending on the message text, please check whether the property specified for this object type exists or can be modified or whether the specified value is permitted for this parameter.

9.3.4.3 Class: BcToolException

Meaning:
An error occurred in one of the Management Console basic classes.

Possible causes:

- File access error
- Class access error
- Other errors depending on the type of exception

Message text:
Differs depending on exception type

Solution:
Depending on the message text, please check an error has occurred when accessing a file or a Java class or there is any other error that can be corrected by the user. If this is not the case, please contact the service/diagnostics department.

9.4 Functions

9.4.1 General

The MC-CLI interface consists of a number of modules, each of which contains the functions for an object type that can be configured and administered using the MC-CLI, see also Section 9.1, "Overview of MC-CLI".

Each module together with its functions is described in this section. The descriptions are presented in alphabetical order.

The exact meanings of the functions and properties are described in the Management Console's online help system. For more information on the online help system, see Section 5.1.2, "Starting the Management Console's online Help system".

9.4.1.1 Parameters

The parameters specified for each function must be entered in the sequence indicated (positional parameters). The keyword parameters are optional and depend on the function. The description indicates whether or not they have to be specified. They are identified by the presence of (kw) (= key word parameter).

In the following function descriptions, the data type that is expected for each parameter is specified (e.g. **Parameter:** `object_name` (String)).

The Jython data type "Dictionary" is used in some functions. In the parameter descriptions, this term is used as follows: "Dictionary with the key-value pairs..."

9.4.1.2 Properties

In the following sections, there is a subsection entitled "Properties of an <object>" for all modules that contain functions for a specific object type. The table in this subsection contains all the properties of objects of the named object type. The meanings of the columns are as follows:

- The column "Keyword (key) at the MC-CLI" contains the names by which the properties are identified in the MC-CLI.
- The column "Field name in the GUI" contains the property names used in the Management Console GUI and online help system.
- The "Funct." column has the following meaning:

c	(create)	Property can be specified in `create()`
cd	(create/default)	Property can be specified in `create()`. If it is not specified, a default value is set.
cf	(create/forced)	Property must be specified in `create()`. If it is not specified, a `BcParameterException` is thrown.
g	(get)	Property is returned by `getProperties()`
m	(modify)	Property can be modified in `modifyProperties()`
m(s)	(modify/synchronize)	(only for resource adapters in clusters) Property can be modified in `modifyProperties()`; all the resource adapters in the cluster are synchronized on saving.

- The "Property value" column contains the values permitted for a property. The variables have the following meanings:

(String)	String
(String numeric)	String that consists only of numbers
`BcDef.<prop>_xxx`	Values that are defined in the Java class `BcDef` and start with "*<prop>*_", (e.g. `BcDef.BOOL_xxx` for `BcDef.BOOL_TRUE` and `BcDef.BOOL_FALSE`)

The specification `None` is not permitted for a property value. Specifying an empty string deletes the previous values of a property.

9.4.1.3 Messages

When many actions are executed, the Management Console generates messages which are output in a separate protocol window. In the MC-CLI, these protocol messages are output at `stdout` with the prefix `MC-CLI:ProtocolMessage` (asynchronous).

9.4.1.4 Returns

The returns described under **Return** are only returned if the function was executed successfully.

The MC-CLI functions do not return a return code specifying whether the function was executed successfully. If a function cannot be executed then an exception from the package `com.fujitsu.ts.jca.tools.mc.cli` is thrown (see Section 9.3.4, "Exceptions"). Such exceptions can be caught in the script and processed appropriately.

Actions started using the function `perform()` usually correspond to the actions that open an action dialog box in the graphical user interface. The action dialog box outputs a table with the description, status and results of all the subactions. In the MC-CLI, the function `perform()` normally returns an object of class `BcObject` with object type `BcObjectType.ACTION`. The functions of the module `BcAdminAction` can be used to extract this information from this object.

9.4.2 BcAdminAction

The module `BcAdminAction` contains functions that analyze the result of a `perform()`call, i.e. a `BcObject` object of type `BcObjectType.ACTION`, and return certain information.

`BcAdminAction` contains the functions:

- getCheckResults() – Show results of check actions
- getResults() – Show results of all subactions of an action
- isFinishedSuccessfully() – Show success/failure of an action

9.4.2.1 getCheckResults() – Show results of check actions

Function: `BcAdminAction.getCheckResults()`

Returns a dictionary that contains the results of all check proxy container subactions of the specified type.

Parameters: `bc_object`

(`BcObject` of type `BcObjectType.ACTION`)

The result of a `perform()` call.

`check_type`

Type of check, possible specifications are:
`BcDef.ACTION_CHECK_ADM`
`BcDef.ACTION_CHECK_AVAIL`

Returns: Dictionary with key-value pairs consisting of the proxy name and check result (value=`result`) of all check proxy container subactions.

Examples of the return

```
{"BCCnt1": "Available", "BcCnt2": "Not available"}
{"BCCnt1": "Administrable", "BcCnt2": "Not administrable"}
```

Exceptions: `BcObjectException, BcToolException`

Note: This function returns a selection of the results of the function `getResults()`, namely information about the administrability (specification `BcDef.ACTION_CHECK_ADM)` or the status (specification `BcDef.ACTION_CHECK_AVAIL`) of the proxy container(s)). The return information contains key-value pairs with:

- key: Name of the proxy container (object_name)
- value: Result of the subaction (result)

where `object_name` and `result` are the properties of the subaction as described in `getResult()`. If the specified action was the return value of a check call for a proxy then the dictionary contains only one element. If it was the return value of a check call for a proxy cluster then it contains one element for each proxy in the cluster.

Example:

```
...
import BcAdminAction
import BcAdminProxy
...
proxy_obj=BcAdminProxy.getObject("BCProxy")
bcaction=BcAdminProxy.perform(proxy_obj,action="check-avail")
checkProxy=BcAdminAction.getCheckResults(bcaction,\
                                      BcDef.ACTION_CHECK_AVAIL)
...
```

9.4.2.2 getResults() – Show results of all subactions of an action

Function: `BcAdminAction.getResults()`

Returns a list of dictionaries containing the information for all the subactions of this action.

Parameters: `bc_object` (`BcObject` of type `BcObjectType.ACTION`)

The result of a `perform()` call.

Returns: List of dictionaries with the key-value pairs for all the properties of the subactions.

A dictionary containing the information in the form of key-value pairs is returned for each subaction. This information broadly corresponds to the columns of the action dialog box in the graphical user interface, see the Layout of the Management Console section of the Management Console online help system – Action dialog box – Layout. There is some additional information that simplifies processing at program level. The value (value) is always of type String, and can also be empty.

Exceptions: `BcObjectException`, `BcToolException`.

Example:

```
...
import BcAdminAction
import BcAdminProxy
...
proxy_obj=BcAdminProxy.getObject("BCProxy")
bcaction=BcAdminProxy.perform(proxy_obj, "save")
iresDicn=BcAdminAction.getResults(bcaction)
...
```

Keyword (key) at the MC-CLI	Column name in GUI	Meaning
`action`	**Action**	Text of the subaction
`action-type`	Partly present in **Action**	Type of subaction (`BcDef.ACTION_ xxx` or `BcDef.ACTION_SA_xxx`)
`details`	**Result Details**	Detailed information on the result, if available
`id`	**#**	Number of the subaction
`object-name`	Partly present in **Action**	Name of the object affected by this subaction
`object-type`	Partly present in **Action**	Type of object (`BcObjectType.xxx.toString()`)

Keyword (key) at the MC-CLI	Column name in GUI	Meaning
result	**Result**	Result of the subaction (BcDef.ACTION_RESULT_xxx)
stack	**StackTrace**	Detailed information about the stack if an exception has occurred.
state	**Action State**	Status of the subaction (BcDef.ACTION_STATE_xxx)

Note:

- The most important values of object-type and action-type are defined in BcObjectType and BcDef. However, it is also possible for non-predefined values to be returned.
- In order to filter certain events out of this volume of information, it may be useful to create your own functions that read individual items of information from the dictionary returned by getResults() and prepare this as required. One example is available in the form of getCheckResults() and another is present in the form of the function printResults() in the sample script sampleAdminProxyCluster.py.
- The id numbers of the subactions are hierarchically structured to show which (sub)actions belong to which higher-level actions.

9.4.2.3 isFinishedSuccessfully() – Show success/failure of an action

Function: `BcAdminAction.isFinishedSuccessfully()`

Specifies whether it was possible to complete the initiated action successfully.

Parameters: `bc_object`

(`BcObject` of type `BcObjectType.ACTION`)

The result of a `perform()` call.

Returns: `True` if the action could be completed successfully together with all its subactions.

`False` if at least one of the subactions could not be completed successfully.

Exceptions: `BcObjectException`

Example:

```
...
import BcAdminAction
import BcAdminProxy
...
proxy_obj=BcAdminProxy.getObject("BCProxy")
...
bcaction=BcAdminProxy.perform(proxy_obj, "save")
isSucc=BcAdminAction.isFinishedSuccessfully(bcaction)
...
```

9.4.3 BcAdminEisPartner

The module `BcAdminEisPartner` contains all the functions available for configuring and administering a Management Console EIS partner object.

`BcAdminEisPartner` contains the functions:

- create() – Add EIS partner to the configuration
- getObject() – Read EIS partner object from the configuration
- getProperties() – Read properties of an EIS partner
- modifyProperties() – Modify properties of an EIS partner
- perform() – Start administrative actions
- remove() – Remove EIS partner

9.4.3.1 create() – Add EIS partner to the configuration

Function: `BcAdminEisPartner.create()`

An EIS partner, whose properties you must pass to the MC-CLI in a dictionary, is added to the configuration.

Parameters: `object_name (String)`

Name of the BeanConnect EIS partner

`proxy_object`
(`BcObject` of type `BcObjectType.PROXY` / `BcObjectType.PROXY_CLUSTER`)

Proxy or proxy cluster to which the EIS partner is to be assigned.

The specification of a `proxy_object` of type `BcObjectType.PROXY` is not permitted if the proxy belongs to a proxy cluster. In this case, the proxy cluster object must be specified as a parameter here.

`props` (kw)

Dictionary with the key-value pairs for the properties that are to be assigned to the EIS partner. You can find the possible values for key in Section 9.4.3.7, "Properties of an EIS partner".

The properties `name`, `type`, `utm.hosts`, `utm.listener-port` and `utm.partner-lpap` must be specified. All the other properties are either optional or set to the default values.

Returns: `(BcObject` of type `BcObjectType.EIS_PARTNER)`

The EIS partner newly added to the configuration

Exceptions: `BcParameterException, BcObjectException, BcToolException`

Example:

```
...
import BcAdminEisPartner
import BcAdminProxy
...
proxy_obj=BcAdminProxy.getObject("BCProxy")
creProps= {"type":BcDef.PTYPE_UTM,"desc": "myEIS",\
     "utm.hosts":"xyz","utm.partner-lpap":"LPAP",\
     "utm.listener-port": "1234"}
newEis=BcAdminEisPartner.create("TestEIS",proxy_obj,creProps)

...
```

9.4.3.2 getObject() – Read EIS partner object from the configuration

Function: `BcAdminEisPartner.getObject()`

Reads the EIS partner with the specified name from the configuration

Parameters: `object_name` (string)

Name of the BeanConnect EIS partner that is to be read.

`proxy_object`
(`BcObject` of type `BcObjectType.PROXY` / `BcObjectType.PROXY_CLUSTER`)

Proxy or proxy cluster to which the EIS partner is assigned.

Returns: `(BcObject` of type `BcObjectType.EIS_PARTNER)`

The read EIS partner or `None` if no EIS partner with a corresponding name exists.

Exceptions: `BcObjectException, BcParameterException, BcToolException`

Note: The specification of a `proxy_object` of type `BcObjectType.PROXY` is only permitted if the proxy is not present in a proxy cluster. In this case, the proxy cluster object must be specified as a parameter here.

Example:

```
...
import BcAdminEisPartner
import BcAdminProxy
...
proxy_obj=BcAdminProxy.getObject("BCProxy")
eis_obj=BcAdminEisPartner.getObject("testEIS",proxy_obj)

...
```

9.4.3.3 getProperties() – Read properties of an EIS partner

Function: `BcAdminEisPartner.getProperties()`

Reads all the properties of the specified EIS partner and returns a dictionary with key-value pairs for the properties.

Parameters: `bc_object` (`BcObject` of type BcObjectType.EIS_PARTNER)

EIS partner whose properties are to be read.

Returns: Dictionary with the key-value pairs for all the properties of the EIS partner. You can find the possible values for key in Section 9.4.3.7, "Properties of an EIS partner".

In the case of EIS partners with the property `type`="`cics`", no properties with the prefix "`utm.`" are output.

Exceptions: `BcObjectException, BcToolException`

Example:

```
...
import BcAdminEisPartner
import BcAdminProxy
...
proxy_obj=BcAdminProxy.getObject("BCProxy")
eis_obj=BcAdminEisPartner.getObject("testEIS",proxy_obj)
eisProps=BcAdminEisPartner.getProperties(eis_obj)

...
```

9.4.3.4 modifyProperties() – Modify properties of an EIS partner

Function: `BcAdminEisPartner.modifyProperties()`

Modifies all the properties of the specified EIS partner that are present in the specified dictionary.

Parameters: `bc_object` (`BcObject` of type `BcObjectType.EIS_PARTNER`)

EIS partner whose properties are to be modified.

`props`

Dictionary with the key-value pairs for the properties that are to be modified. You can find the possible values for key in Section 9.4.3.7, "Properties of an EIS partner".

In the case of EIS partners with the property `type`="`cics`", no properties with the prefix "`utm.`" may be specified.

Returns: None

Exceptions: `BcObjectException, BcParameterException`, `BcToolException`

Example:

```
...
import BcAdminEisPartner
import BcAdminProxy
...
proxy_obj=BcAdminProxy.getObject("BCProxy")
eis_obj=BcAdminEisPartner.getObject("testEIS",proxy_obj)
modProps={"desc":"modified"}
BcAdminEisPartner.modifyProperties(eis_obj, modProps)

...
```

9.4.3.5 perform() – Start administrative actions

Function: `BcAdminEisPartner.perform()`

Starts the specified action for the EIS partner.

Parameters: `bc_object` (`BcObject` of type `BcObjectType.EIS_PARTNER`)

EIS partner for which the action is to be started.

`action (String)`

Action that is to be started for the specified EIS partner. Possible values are (see `BcDef.ACTION_xxx` in the MC-CLI-JavaDoc)

`check-avail`	Checks the availability of the EIS partner.
`gen-config`	Generates a configuration file that must be integrated in the configuration of the EIS partner.

Returns: For `action="gen-config"`:

- `True` (boolean) if the configuration files have been created.
- `False` (boolean) if the configuration files could not be created.

For `action="check-avail"`:

- (String)
  Check service output message if proxy and EIS partner have been started and the service could be called successfully.
- (String)
  Message from MC-CLI or the proxy or the EIS partner if the service could not be called successfully.

Exceptions: `BcObjectException, BcParameterException, BcToolException`

Example:

```
...
import BcAdminEisPartner
import BcAdminProxy
...
proxy_obj=BcAdminProxy.getObject("BCProxy")
eis_obj=BcAdminEisPartner.getObject("testEIS",proxy_obj)
isSucc=BcAdminEisPartner.perform(eis_obj,action="gen-config")

...
```

9.4.3.6 remove() – Remove EIS partner

Function: `BcAdminEisPartner.remove()`

Removes the specified EIS partner from the configuration.

Parameters: `bc_object` (`BcObject` of type `BcObjectType.EIS_PARTNER`).

EIS partner that is to be removed.

Returns: None

Exceptions: `BcObjectException, BcToolException`

Example:

```
...
import BcAdminEisPartner
import BcAdminProxy
...
proxy_obj=BcAdminProxy.getObject("BCProxy")
eis_obj=BcAdminEisPartner.getObject("testEIS",proxy_obj)
BcAdminEisPartner.remove (eis_obj)

...
```

9.4.3.7 Properties of an EIS partner

The following table contains all the properties of an EIS partner.

The meanings and values permitted for the various properties can be found in the Management Console online help system under
Configuring BeanConnect Proxies – EIS Partners – Editing an EIS Partner. Tabs named General, UTM Partner, and Availability Check are available for this.

Keyword (key) at the MC-CLI [1]	**Field name in the GUI** [1]	**Funct.**[1]	**Property value**[1]
`active`	**Active**	cd / g / m	`BcDef.BOOL_xxx`[2]
`avail.char-code`	**Character Code**	cd / g / m	`BcDef.CHAR_CODE_xxx`[2]
`avail.check-service`	**Check Service**	c / g / m	`BcDef.EIS_AVAIL_NO_CHECK_SERVICE` or (String)
`avail.message`	**Message**	c / g / m	(String)
`avail.password`	**Password**	c / m	(String)
`avail.user`	**Users**	c / g / m	(String)
`connections`	**Connections**	cd / g / m	(String numeric)
`desc`	**Description**	c / g / m	(String)
`name`	**Name**	cf / g / m	(String)
`id`	**ID**	g	(String)
`prefix`	**Prefix**	cd / g / m	(String)
`type`	**Type**	cf / g	`BcDef.PTYPE_UTM` / `BcDef.PTYPE_CICS`
`utm.access-point`[3]	**Access Point**	cd / g / m	`BcDef.ACCESS_POINT_xxx`[2]
`utm.access-point-name`[3]	**Access Point Name**	c / g / m	(String)
`utm.admin-permission`[3]	**Admin Permission**	cd / g / m	`BcDef.BOOL_xxx`[2]
`utm.api-mode`[3]	**Application Program Interface Mode of EIS Partner**	cd / g / m	`BcDef.EIS_API_MODE_xxx`[2]
`utm.appl-entity-qualifier`[3]	**Application Entity Qualifier**	cd / g / m	(String)
`utm.appl-process-title`[3]	**Application Process Title**	cd / g / m	(String)
`utm.hosts`[3]	**Hosts**	cf / g / m	(String)
`utm.is-bs2000`[3]	**Is BS2000**	cd / g / m	`BcDef.BOOL_xxx`[2]
`utm.listener-port`[3]	**Listener Port**	cf / g / m	(String numeric)

Keyword (key) at the MC-CLI [1]	Field name in the GUI [1]	Funct.[1]	Property value[1]
`utm.partner-idletimer.sec`[3]	**Partner Idletimer**	cd / g / m	(String numeric)
`utm.partner-lpap`[3]	**Partner LPAP**	cf / g / m	(String)
`utm.proxy-auto-conn`[3]	**Proxy AutoConnect**	cd / g / m	(String numeric)
`utm.proxy-cont-winners`[3]	**Proxy ContentionWinners**	cd / g / m	(String numeric)
`utm.proxy-host`[3]	**Proxy Hostname**	cd / g / w	(String)
`utm.proxy-idletimer.sec`[3]	**Proxy Idletimer**	cd / g / m	(String numeric)
`utm.transport-selector`[3]	**Transport Selector**	c / g / m	(String)
`utm.transport-selector-format`[3]	**Transport Selector Format**	cd / g / m	`BcDef.TSEL_FORMAT_xxx`[2]

[1] For the meanings of the columns and abbreviations, see Section 9.4.1.2, "Properties"

[2] The possible values of xxx can be found in the JavaDoc (Java class `BcDef`) tn the `JavaDoc` subdirectory of the Management Console installation directory

[3] Properties with the prefix "`utm.`" can only be read and modified in the case of EIS partners of type `utm`.

Notes:

- Not all combinations of properties can be modified. In some cases, properties cannot be modified if other properties have certain values. Thus, for example
  - `avail.user`, `avail.password`, `avail.char-code` cannot be modified if `avail.check-service=BcDef.EIS_AVAIL_NO_CHECK_SERVICE`
  - `utm.access-point-name`, `transport-selector`, `transport-selector-format` cannot be modified if `utm-access-point=BcDef.ACCESS_POINT_CREATE_GENERIC`.

    For more information, see the description (online help system) of the Management Console (e.g. for utm.api-mode).
- The property **Availability Check – Perform Check** is not available here because no automatic checks are performed in the MC-CLI.

9.4.4 BcAdminInboundMsgEndpoint

The module `BcAdminInboundMsgEndpoint` contains all the functions available for configuring and administering a Management Console inbound message object.

`BcAdminInboundMsgEndpoint` contains the functions:

- create() – Add inbound message endpoint to the configuration
- getObject() – Read inbound message endpoint object from the configuration
- getProperties() – Read properties of an inbound message endpoint
- getProperties() – Modify properties of an inbound message endpoint
- remove() – Remove inbound message endpoint

9.4.4.1 create() – Add inbound message endpoint to the configuration

Function: `BcAdminInboundMsgEndpoint.create()`

An inbound message endpoint, whose properties you must pass to the MC-CLI in a dictionary, is added to the configuration.

Parameters: `object_name` (String)

Name of the BeanConnect inbound message endpoint

`proxy_object`
(`BcObject` of type `BcObjectType.PROXY` or `BcObjectType.PROXY_CLUSTER`)

Proxy or proxy cluster to which the inbound message endpoint is to be assigned.

The specification of a `proxy_object` of type `BcObjectType.PROXY` is not permitted if the proxy belongs to a proxy cluster. In this case, the proxy cluster object must be specified as a parameter here.

`props`

Dictionary with the key-value pairs for the properties that are to be assigned to the inbound message endpoint. You can find the possible values for key in Section 9.4.4.6, "Properties of an inbound message endpoint".

Returns: (`BcObject` of type `BcObjectType.INBOUND_MSG_ENDPOINT`)

The inbound message endpoint added to the configuration.

Exceptions: `BcParameterException, BcObjectException, BcToolException`

Example:

```
...
import BcAdminInboundMsgEndpoint
import BcAdminProxy
...
proxy_obj=BcAdminProxy.getObject("BCProxy")
creProps={"desc":"my msg endpoint",\
"dial-type":BcDef.TYPE_DIALOG,\
"resource-adapter":"myRA","service-names":"TESTIS1, TESTIS2"}

me_obj=BcAdminInboundMsgEndpoint.create("myMEP", proxy_obj,\
props=creProps)
...
```

9.4.4.2 getObject() – Read inbound message endpoint object from the configuration

Function: `BcAdminInboundMsgEndpoint.getObject()`

Reads the inbound message endpoint with the specified name from the configuration

Parameters: `object_name` (String)

Name of the BeanConnect inbound message endpoint that is to be read.

`proxy_object`
(`BcObject` of type `BcObjectType.PROXY` / `ObjectType.PROXY_CLUSTER`)

The proxy or proxy cluster to which the inbound message endpoint is to be assigned.

The specification of a `proxy_object` of type `BcObjectType.PROXY` is not permitted if the proxy belongs to a proxy cluster. In this case, the proxy cluster object must be specified as a parameter here.

Returns: (`BcObject` of type `BcObjectType.INBOUND_MSG_ENDPOINT`)

The read inbound message endpoint or `None` if no inbound message endpoint with a corresponding name exists.

Exceptions: `BcObjectException, BcParameterException, BcToolException`

Example:

```
...
import BcAdminInboundMsgEndpoint
import BcAdminProxy
...
proxy_obj=BcAdminProxy.getObject("BCProxy")
me_obj=BcAdminInboundMsgEndpoint.getObject("myMEP",proxy_obj)
...
```

9.4.4.3 getProperties() – Read properties of an inbound message endpoint

Function: `BcAdminInboundMsgEndpoint.getProperties()`

Reads all the properties of the specified inbound message endpoint and returns a dictionary with key-value pairs for the properties.

Parameters: `bc_object` (`BcObject` of type `BcObjectType.INBOUND_SERVICE`)

The inbound message endpoint whose properties are to be read.

Returns: Dictionary with the key-value pairs for all the properties of the inbound message endpoint. You can find the possible values for key in Section 9.4.4.6, "Properties of an inbound message endpoint".

Exceptions: `BcObjectException, BcToolException`

Example:

```
...
import BcAdminInboundMsgEndpoint
import BcAdminProxy
...
proxy_obj=BcAdminProxy.getObject("BCProxy")
me_obj=BcAdminInboundMsgEndpoint.getObject("myMEP",proxy_obj)
meProps=BcAdminInboundMsgEndpoint.getProperties(me_obj)
...
```

9.4.4.4 getProperties() – Modify properties of an inbound message endpoint

Function: `BcAdminInboundMsgEndpoint.modifyProperties()`

Modifies all the properties of the specified inbound message endpoint that are present in the specified dictionary.

Parameters: `bc_object` (`BcObject` of type `BcObjectType.INBOUND_MSG_ENDPOINT`)

The inbound message endpoint whose properties are to be modified.

`props`

Dictionary with the key-value pairs for the properties that are to be modified. You can find the possible values for key in Section 9.4.4.6, "Properties of an inbound message endpoint".

Returns: None

Exceptions: `BcObjectException, BcParameterException, BcToolException`

Example:

```
...
import BcAdminInboundMsgEndpoint
import BcAdminProxy
...
proxy_obj=BcAdminProxy.getObject("BCProxy")
me_obj=BcAdminInboundMsgEndpoint.getObject("myMEP",proxy_obj)
modProps={"desc":"created","service-names":"TESTIS1, TESTIS2"}
BcAdminInboundMsgEndpoint.modifyProperties(me_obj,modProps)
...
```

9.4.4.5 remove() – Remove inbound message endpoint

Function: `BcAdminInboundMsgEndpoint.remove()`

Removes the specified inbound message endpoint from the configuration.

Parameters: `bc_object` (`BcObject` of type `BcObjectType.INBOUND_MSG_ENDPOINT`)

Inbound message endpoint that is to be removed.

Returns: None

Exceptions: `BcObjectException, BcToolException`

Example:

```
...
import BcAdminInboundMsgEndpoint
import BcAdminProxy
...
proxy_obj=BcAdminProxy.getObject("BCProxy")
me_obj=BcAdminInboundMsgEndpoint.getObject("myMEP",proxy_obj)
BcAdminInboundMsgEndpoint.remove(me_obj)
...
```

9.4.4.6 Properties of an inbound message endpoint

The following table contains all the properties of an inbound message endpoint.

The meanings and values permitted for the various properties can be found in the Management Console online help system under
Configuring BeanConnect Proxies – Inbound – Inbound message endpoints – Inbound message endpoint, properties.

Keyword (key) at the MC-CLI [1]	Field name in the GUI [1]	Funct.[1]	Property value[1]
`desc`	**Description**	c / g / m	(String)
`dial-type`	**Type (dial / asyn)**	cd / g / m	`BcDef.TYPE_xxx`[2]
`dlq-msgs`	**Dead letter queue Messages**	g	(String numeric)
`name`	**Name**	cf / g / m	(String)
`reply-timer.sec`	**ReplyTimer**	cd / g / m	(String numeric)
`resource-adapter`[3]	**Resource Adapter**	cf / g / m	(String)
`service-names`	**Service Names**	cf / g / m	(String)
`state`	**State**	g	`BcDef.STATE_xxx`[2]
`ta-timer.sec`	**Transaction Timer**	cd / g / m	(String numeric)
`waiting-msgs`	**Waiting Messages**	g	(String numeric)

[1] For the meanings of the columns and abbreviations, see Section 9.4.1.2, "Properties"

[2] The possible values of xxx can be found in the JavaDoc (Java class `BcDef`) tn the `JavaDoc` subdirectory of the Management Console installation directory

[3] If the inbound message endpoint belongs to a proxy cluster then the property is not output by `getProperties()` and may not be specified in `modifyProperties()`.

9.4.5 BcAdminInboundService

The module `BcAdminInboundService` contains all the functions available for configuring and administering a Management Console inbound service object. Objects of type "inbound service" are implicitly generated and removed via the configuration of the inbound message endpoint. There are therefore no `create()` and `remove()` functions for inbound services.

`BcAdminInboundService` contains the functions:

- getObject() – Read inbound service object from the configuration
- getProperties() – Read properties of an inbound service
- modifyProperties() – Modify properties of an inbound service

9.4.5.1 getObject() – Read inbound service object from the configuration

Function: `BcAdminInboundService.getObject()`

Reads the inbound service with the specified name from the configuration

Parameters: `object_name` (String)

Name of the BeanConnect inbound service that is to be read.

`proxy_object`
(`BcObject` of type `BcObjectType.PROXY` / `BcObjectType.PROXY_CLUSTER`)

Proxy or proxy cluster to which the inbound service is assigned.

The specification of a `proxy_object` of type `BcObjectType.PROXY` is not permitted if the proxy belongs to a proxy cluster. In this case, the proxy cluster object must be specified as a parameter here.

Returns: `(BcObject` of type `BcObjectType.INBOUND_SERVICE)`

The read inbound service or `None` if no inbound service with a corresponding name exists.

Exceptions: `BcObjectException, BcParameterException, BcToolException`

Example:

```
...
import BcAdminInboundService
import BcAdminProxy
...
proxy_obj=BcAdminProxy.getObject("BCProxy")
serv_obj=BcAdminInboundService.getObject("SRV",proxy_obj)
...
```

9.4.5.2 getProperties() – Read properties of an inbound service

Function: `BcAdminInboundService.getProperties()`

Reads all the properties of the specified inbound service and returns a dictionary with key-value pairs for the properties.

Parameters: `bc_object` (`BcObject` of type `BcObjectType.INBOUND_SERVICE`)

Inbound service whose properties are to be read.

Returns: Dictionary with the key-value pairs for all the properties of the inbound service. You can find the possible values for `key` in Section 9.4.5.4, "Properties of an inbound service".

Exceptions: `BcObjectException, BcToolException`

Example:

```
...
import BcAdminInboundService
import BcAdminProxy
...
proxy_obj=BcAdminProxy.getObject("BCProxy")
is_obj=BcAdminInboundService.getObject("SRV",proxy_obj)
isProps=BcAdminInboundService.getProperties(is_obj)
...
```

9.4.5.3 modifyProperties() – Modify properties of an inbound service

Function: `BcAdminInboundService.modifyProperties()`

Modifies all the properties of the specified inbound service that are present in the specified dictionary.

Parameters: `bc_object`
(`BcObject` of type `BcObjectType.INBOUND_SERVICE`)

Inbound service whose properties are to be modified.

`props`

Dictionary with the key-value pairs for the properties that are to be modified. You can find the possible values for key in Section 9.4.5.4, "Properties of an inbound service".

Returns: None

Exceptions: `BcObjectException, BcParameterException, BcToolException`

Example:

```
...
import BcAdminInboundService
import BcAdminProxy
...
proxy_obj=BcAdminProxy.getObject("BCProxy")
is_obj=BcAdminInboundService.getObject("SRV",proxy_obj)
modProps{"desc":"modified"}
BcAdminInboundService.modifyProperties(is_obj,modProps)
...
```

9.4.5.4 Properties of an inbound service

The following table contains all the properties of an inbound service.

The meanings and values permitted for the various properties can be found in the Management Console online help system under
Configuring BeanConnect Proxies – Inbound – Inbound services – Editing inbound service.

Keyword (key) at the MC-CLI [1]	Field name in the GUI [1]	Funct.[1]	Property value[1]
desc	**Description**	g / m	(String)
dial-type	**Type**	g	BcDef.TYPE_xxx[2]
inbound-msg-endpoint	**Inbound Message Endpoint**	g	(String)
name	**Service Name**	g	(String)
partner-char-code	**Partner Character Code**	g / m	BcDef.CHAR_ CODE_xxx[2]
partner-encoding	**Partner Encoding**	g / m	(String)
xatmi	**XATMI**	g / m	BcDef.BOOL_xxx[2]

[1] For the meanings of the columns and abbreviations, see Section 9.4.1.2, "Properties"

[2] The possible values of xxx can be found in the JavaDoc (Java class BcDef) tn the JavaDoc subdirectory of the Management Console installation directory

9.4.6 BcAdminInboundUser

The module `BcAdminInboundUser` contains all the functions available for configuring and administering a Management Console inbound user object.

`BcAdminInboundUser` contains the functions:

- create() – Add inbound user to the configuration
- getObject() – Read inbound user object from the configuration
- getProperties() – Read properties of an inbound user
- modifyProperties() – Modify properties of an inbound user
- remove() – Remove inbound user

9.4.6.1 create() – Add inbound user to the configuration

Function: `BcAdminInboundUser.create()`

An inbound user, whose properties you must pass to the MC-CLI in a dictionary, is added to the configuration.

Parameters: `object_name` (String)

Name of the BeanConnect inbound user

`proxy_object`
(`BcObject` of type `BcObjectType.PROXY` / `BcObjectType.PROXY_CLUSTER`)

Proxy or proxy cluster to which the inbound user is to be assigned.

The specification of a `proxy_object` of type `BcObjectType.PROXY` is not permitted if the proxy belongs to a proxy cluster. In this case, the proxy cluster object must be specified as a parameter here.

`props (kw)`

Dictionary with the key-value pairs for the properties that are to be assigned to the inbound user. You can find the possible values for key in Section 9.4.6.6, "Properties of an inbound user".

Returns: (`BcObject` of type `BcObjectType.INBOUND_USER`)

The inbound user added to the configuration.

Exceptions: `BcParameterException, BcObjectException, BcToolException`

Example:

```
...
import BcAdminInboundUser
import BcAdminProxy
...
proxy_obj=BcAdminProxy.getObject("BCProxy")
creProps={"desc":"created","password":"ADMIN"}
usr_obj=BcAdminInboundUser.create("USR",proxy_obj,props=creProps)
...
```

9.4.6.2 getObject() – Read inbound user object from the configuration

Function: `BcAdminInboundUser.getObject()`

Reads the specified inbound user from the configuration.

Parameters: `object_name` (string)

Name of the BeanConnect inbound user that is to be read.

`proxy_object` (`BcObject` of type `BcObjectType.PROXY` / `BcObjectType.PROXY_CLUSTER`)

Proxy or proxy cluster to which the inbound user is assigned.

The specification of a `proxy_object` of type `BcObjectType.PROXY` is only permitted if the proxy is not present in a proxy cluster. In this case, the proxy cluster object must be specified as a parameter here.

Returns: (`BcObject` of type `BcObjectType.INBOUND_USER`)

The read inbound user or `None` if no inbound user with a corresponding name exists.

Exceptions: `BcObjectException, BcParameterException, BcToolException`

Example:

```
...
import BcAdminInboundUser
import BcAdminProxy
...
proxy_obj=BcAdminProxy.getObject("BCProxy")
user_obj =BcAdminInboundUser.getObject("USR",proxy_obj)
...
```

9.4.6.3 getProperties() – Read properties of an inbound user

Function: `BcAdminInboundUser.getProperties()`

Reads all the properties of the specified inbound user and returns a dictionary with key-value pairs for the properties.

Parameters: `bc_object` (**`BcObject`** of type `BcObjectType.INBOUND_USER`)

Inbound user whose properties are to be read.

Returns: Dictionary with key-value pairs for the properties of the inbound user. You can find the possible values for key in Section 9.4.6.6, "Properties of an inbound user".

Exceptions: `BcObjectException, BcToolException`

Example:

```
...
import BcAdminInboundUser
import BcAdminProxy
...
proxy_obj=BcAdminProxy.getObject("BCProxy")
user_obj=BcAdminInboundUser.getObject("USR",proxy_obj)
userProps=BcAdminInboundUser.getProperties(user_obj)
...
```

9.4.6.4 modifyProperties() – Modify properties of an inbound user

Function: `BcAdminInboundUser.modifyProperties()`

Modifies all the properties of the specified inbound user that are present in the specified dictionary.

Parameters: `bc_object` (`BcObject` of type `BcObjectType.INBOUND_USER`)

Inbound user whose properties are to be modified.

`props`

Dictionary with key-value pairs for the properties that are to be modified. You can find the possible values for key in Section 9.4.6.6, "Properties of an inbound user".

Returns: None

Exceptions: `BcObjectException, BcParameterException, BcToolException`

Example:

```
...
import BcAdminInboundUser
import BcAdminProxy
...
proxy_obj=BcAdminProxy.getObject("BCProxy")
user_obj=BcAdminInboundUser.getObject("USR",proxy_obj)
modProps={"desc":"modified"}
BcAdminInboundUser.modifyProperties(user_obj,modProps)
...
```

9.4.6.5 remove() – Remove inbound user

Function: `BcAdminInboundUser.remove()`

Removes the specified inbound user from the configuration.

Parameters: `bc_object` (`BcObject` of type `BcObjectType.INBOUND_USER`)

Inbound user that is to be removed.

Returns: None

Exceptions: `BcObjectException, BcParameterException, BcToolException`

Example:

```
...
import BcAdminInboundUser
import BcAdminProxy
...
proxy_obj=BcAdminProxy.getObject("BCProxy")
user_obj=BcAdminInboundUser.getObject("USR",proxy_obj)
BcAdminInboundUser.remove (user_obj)
...
```

9.4.6.6 Properties of an inbound user

The following table contains all the properties of an inbound user.

The meanings and values permitted for the various properties can be found in the Management Console online help system under
Configuring BeanConnect Proxies – Inbound – Inbound Users – Editing an inbound user, General.

Keyword (key) at the MC-CLI [1]	Field name in the GUI [1]	Funct.[1]	Property value[1]
`desc`	**Description**	c / g / m	(String)
`name`	**Name**	cf / g	(String)
`password`	**Password**	c / m	(String)

[1] For the meanings of the columns and abbreviations, see Section 9.4.1.2, "Properties"

9.4.7 BcAdminMain

You can use the functions available in the module `BcAdminMain` to start or close a Management Console session for the MC-CLI as well as to output a list of all the proxies, proxy clusters and todo topics for the current BeanConnect configuration.

`BcAdminMain` contains the functions:

- close() – Close Management Console session
- getList() – Output list of all configured objects of an object type
- getVersion() – Read Management Console version
- init() – Start Management Console session for MC-CLI

9.4.7.1 close() – Close Management Console session

Function: `BcAdminMain.close()`

Closes the Management Console session and, depending on the parameter, saves the as yet unsaved changes.

Parameters: `save_all` (boolean)

`True`	All the changes should be saved.
`False`	All the changes should be rejected (default).

Return: None

Note:

- The `close()` call must be issued at the end of the function calls to the Management Console in order to make it possible to conduct other Management Console sessions. (end of serialization).
- All Java objects of type `BcObject` that have been generated in the current MC-CLI session become invalid when `close()` is called and can no longer be used (e.g. as parameters). You will have to regenerate these objects in a new MC-CLI session if they are required.

Example:

```
...
import BcAdminMain
...

BcAdminMain.close(True)
...
```

9.4.7.2 getList() – Output list of all configured objects of an object type

Function: `BcAdminMain.getlist()`

Reads the names of all the objects of a type from the current BeanConnect configuration and returns a dictionary with names and objects.

Parameters: `list_type` (String)

Type of object of which a list of all objects defined in the current configuration is to be created. You can make the following specifications for `list_type` (see also Section 9.3.2, "Class: BcObjectType")

String	or the value of one of the following `toString()` methods
`"proxy"`	`BcObjectType.PROXY.toString()`
`"proxy-cluster"`	`BcObjectType.PROXY_CLUSTER.toString()`
`"todo"`	`BcObjectType.TODO.toString()`

Return: Dictionary with all the objects of the specified type and the associated names as key.

Exceptions: `BcParameterException`, `BcObjectException`, `BcToolException`.

Example:

```
...
import BcAdminMain
...
''' get list of all proxys configured '''
bcproxys=BcAdminMain.getList("proxy")
...
```

The list of proxies (`list_type= "proxy"`) contains only individual (stand-alone) proxies, i.e. no proxies that belong to a proxy cluster. You can read the proxies that are present in a proxy cluster using the function `getList()` in the module `BcAdminProxyCluster`.

9.4.7.3 getVersion() – Read Management Console version

Function: `BcAdminMain.getVersion()`

Returns the version of the Management Console and returns this as a string.

Parameters: None

Return: String containing the version of the Management Console

Exceptions: `BcObjectException`

Example:

```
...
import BcAdminMain
...
admver=BcAdminMain.getVersion()
...
```

9.4.7.4 init() – Start Management Console session for MC-CLI

Function: `BcAdminMain.init()`

Starts a Management Console session in which you can administer the BeanConnect configuration via the MC-CLI. The configuration file `console.properties.xml` is read from the `config` subdirectory of the specified console subdirectory `console_home` in BeanConnect. If it does not exist, the file `console.properties.xml` is generated.

Parameters: `console_home` (String)

Management Console subdirectory in the BeanConnect installation directory.

Returns: None

Exceptions: `BcParameterException, BcObjectException, BcToolException`

Note: The thrown `BcToolException` may be due to the fact that another Management Console session already exists (starter from the graphical user interface or via the MC-CLI). Close this session.

If no other session exists, you may need to delete the serialization file `ConsoleInUse.txt` in the Management Console home directory.

Example:

```
...
import sys
import BcAdminMain
...
consoleHome=sys.argv[1]
BcAdminMain.init(consoleHome)

...
```

9.4.8 BcAdminOutboundCommEndpoint

The module `BcAdminOutboundCommEndpoint` contains all the functions available for configuring and administering a Management Console outbound communication endpoint object.

`BcAdminOutboundCommEndpoint` contains the functions:

- create() – Add outbound communication endpoint to the configuration
- getObject() – Read outbound communication endpoint object from the configuration
- getProperties() – Read properties of an outbound configuration endpoint
- modifyProperties() – Modify properties of an outbound communication endpoint
- remove() – Remove outbound communication endpoint

9.4.8.1 create() – Add outbound communication endpoint to the configuration

Function: `BcAdminOutboundCommEndpoint.create()`

An outbound communication endpoint, whose properties you must pass to the MC-CLI in a dictionary, is added to the configuration.

Parameters: `object_name`

Name of the BeanConnect outbound communication endpoint.

`proxy_object`
(`BcObject` of type `BcObjectType.PROXY` / `BcObjectType.PROXY_CLUSTER`)

Proxy or proxy cluster to which the outbound communication endpoint is to be assigned.

The specification of a `proxy_object` of type `BcObjectType.PROXY` is not permitted if the proxy belongs to a proxy cluster. In this case, the proxy cluster object must be specified as a parameter here.

`props` (kw)

Dictionary with the key-value pairs for the properties that are to be assigned to the outbound communication endpoint. You can find the possible values for key in Section 9.4.8.6, "Properties of an outbound communication endpoint".

Returns: `(BcObject` of type `BcObjectType.OUTBOUND_COMM_ENDPOINT)`

The outbound communication endpoint added to the configuration.

Exceptions: `BcParameterException, BcObjectException, BcToolException`

Example:

```
...
import BcAdminOutboundCommEndpoint
import BcAdminProxy
...
proxy_obj=BcAdminProxy.getObject("BCProxy")
crPr={"desc":"created","eis-partner":"EP","partner-service":"TOS"}
ce_obj=BcAdminOutboundCommEndpoint.create("CEND",\
                                            proxy_obj,props=crPr)
...
```

9.4.8.2 getObject() – Read outbound communication endpoint object from the configuration

Function: `BcAdminOutboundCommEndpoint.getObject()`

Reads the outbound communication endpoint with the specified name from the configuration

Parameters: `object_name` (String)

Name of the BeanConnect outbound communication endpoint that is to be read.

`proxy_object`
(`BcObject` of type `BcObjectType.PROXY` / `BcObjectType.PROXY_CLUSTER`)

Proxy or proxy cluster to which the outbound communication endpoint is assigned.

The specification of a `proxy_object` of type `BcObjectType.PROXY` is not permitted if the proxy belongs to a proxy cluster. In this case, the proxy cluster object must be specified as a parameter here.

Returns: `(BcObject` of type `BcObjectType.OUTBOUND_COMM_ENDPOINT)`

The read outbound communication endpoint or `None` if no outbound communication endpoint with a corresponding name exists.

Exceptions: `BcObjectException, BcParameterException, BcToolException`

Example:

```
...
import BcAdminOutboundCommEndpoint
import BcAdminProxy
...
proxy_obj=BcAdminProxy.getObject("BCProxy")
ce_obj=BcAdminOutboundCommEndpoint.getObject("CEND",proxy_obj)
...
```

9.4.8.3 getProperties() – Read properties of an outbound configuration endpoint

Function: `BcAdminOutboundCommEndpoint.getProperties()`

Reads all the properties of the specified outbound communication endpoint and returns a dictionary with key-value pairs for the properties.

Parameters: `bc_object`
(`BcObject` of type `BcObjectType.OUTBOUND_COMM_ENDPOINT`)

The outbound communication endpoint whose properties are to be read.

Returns: Dictionary with key-value pairs for the properties of the outbound communication endpoint. You can find the possible values for key in Section 9.4.8.6, "Properties of an outbound communication endpoint".

Exceptions: `BcObjectException, BcToolException`

Example:

```
...
import BcAdminOutboundCommEndpoint
import BcAdminProxy
...
proxy_obj=BcAdminProxy.getObject("BCProxy")
ce_obj=BcAdminOutboundCommEndpoint.getObject("CEND",proxy_obj)
ceProps=BcAdminOutboundCommEndpoint.getProperties(ce_obj)
...
```

9.4.8.4 modifyProperties() – Modify properties of an outbound communication endpoint

Function: `BcAdminOutboundCommEndpoint.modifyProperties()`

Modifies all the properties of the specified outbound communication endpoint that are present in the specified dictionary.

Parameters: `bc_object`
(`BcObject` of type `BcObjectType.OUTBOUND_COMM_ENDPOINT`)

The outbound communication endpoint whose properties are to be read.

`props`

Dictionary with the key-value pairs for the properties that are to be modified. You can find the possible values for key in Section 9.4.8.6, "Properties of an outbound communication endpoint".

Returns: None

Exceptions: `BcObjectException, BcParameterException, BcToolException`

Example:

```
...
import BcAdminOutboundCommEndpoint
import BcAdminProxy
...
proxy_obj=BcAdminProxy.getObject("BCProxy")
ce_obj=BcAdminOutboundCommEndpoint.getObject("CEND",proxy_obj)
modPr={"desc":"modified"}
BcAdminOutboundCommEndpoint.modifyProperties(ce_obj,modPr)
...
```

9.4.8.5 remove() – Remove outbound communication endpoint

Function: `BcAdminOutboundCommEndpoint.remove()`

Removes the specified outbound communication endpoint from the configuration.

Parameters: `bc_object`
(`BcObject` of type `BcObjectType.OUTBOUND_COMM_ENDPOINT`)

The outbound communication endpoint that is to be removed.

Returns: None

Exceptions: `BcObjectException, BcToolException`

Example:

```
...
import BcAdminOutboundCommEndpoint
import BcAdminProxy
...
proxy_obj=BcAdminProxy.getObject("BCProxy")
ce_obj=BcAdminOutboundCommEndpoint.getObject("CEND",proxy_obj)
BcAdminOutboundCommEndpoint.remove(ce_obj)
...
```

9.4.8.6 Properties of an outbound communication endpoint

The following table contains all the properties of an outbound communication endpoint.

The meanings and values permitted for the various properties can be found in the Management Console online help system under
Configuring BeanConnect Proxies – Outbound – Outbound Communication Endpoints – Outbound communication endpoints, Properties.

Keyword (key) at the MC-CLI [1]	**Field name in the GUI** [1]	**Funct.**[1]	**Property value**[1]
`desc`	**Description**	c / g / m	(String)
`eis-partner`	**EIS Partner**	cf / g / m	(String)
`name`	**Name**	cf / g / m	(String)
`partner-service`	**Partner Service**	cf / g / m	(String)

[1] For the meanings of the columns and abbreviations, see Section 9.4.1.2, "Properties"

9.4.9 BcAdminOutboundService

The module `BcAdminOutboundService` contains all the functions available for configuring and administering a Management Console outbound service object.

`BcAdminOutboundService` contains the functions:

- create() – Add outbound service to the configuration
- getObject() – Read outbound service object from the configuration
- getProperties() – Read properties of an outbound service
- modifyProperties() – Modify properties of an outbound service
- remove() – Remove outbound service

9.4.9.1 create() – Add outbound service to the configuration

Function: `BcAdminOutboundService.create()`

An outbound service, whose properties you must pass to the MC-CLI in a dictionary, is added to the configuration.

Parameters: `object_name`

Name of the BeanConnect outbound service.

`proxy_object`
(`BcObject of type BcObjectType.PROXY` / `BcObjectType.PROXY_CLUSTER`)

Proxy or proxy cluster to which the outbound service is to be assigned.

The specification of a `proxy_object` of type `BcObjectType.PROXY` is not permitted if the proxy belongs to a proxy cluster. In this case, the proxy cluster object must be specified as a parameter here.

`props` (kw)

Dictionary with the key-value pairs for the properties that are to be assigned to the outbound service. You can find the possible values for key in Section 9.4.9.6, "Properties of an outbound service".

Returns: (`BcObject` of type `BcObjectType.OUTBOUND_SERVICE`)

The outbound service added to the configuration.

Exceptions: `BcParameterException, BcObjectException, BcToolException`

Example:

```
...
import BcAdminOutboundService
import BcAdminProxy
...
proxy_obj=BcAdminProxy.getObject("BCProxy")
crPr={"desc":"created","dial-type":BcDef.TYPE DIALOG,\
                                         "reply-timer.sec": "90"}
os_obj=BcAdminOutboundService.create("OSRV",proxy_obj,props=crPr)
...
```

9.4.9.2 getObject() – Read outbound service object from the configuration

Function: `BcAdminOutboundService.getObject()`

Reads the specified outbound service from the configuration

Parameters: `object_name`

Name of the BeanConnect outbound service that is to be read.

`proxy_object`
(`BcObject` of type `BcObjectType.PROXY` / `BcObjectType.PROXY_CLUSTER`)

Proxy or proxy cluster to which the outbound service is assigned.

The specification of a `proxy_object` of type `BcObjectType.PROXY` is not permitted if the proxy belongs to a proxy cluster. In this case, the proxy cluster object must be specified as a parameter here.

Returns: `(BcObject` of type `BcObjectType.OUTBOUND_SERVICE)`

The read outbound service or `None` if no outbound service with a corresponding name exists.

Exceptions: `BcObjectException, BcParameterException, BcToolException`

Example:

```
...
import BcAdminOutboundService
import BcAdminProxy
...
proxy_obj=BcAdminProxy.getObject("BCProxy")
os_obj=BcAdminOutboundService.getObject("OSRV",proxy_obj)
...
```

9.4.9.3 getProperties() – Read properties of an outbound service

Function: `BcAdminOutboundService.getProperties()`

Reads all the properties of the specified outbound service and returns a dictionary with key-value pairs for the properties.

Parameters: `bc_object`
(`BcObject` of type `BcObjectType.OUTBOUND_SERVICE`)

Outbound service whose properties are to be read.

Returns: Dictionary with key-value pairs for the properties of the outbound service. You can find the possible values for key in Section 9.4.9.6, "Properties of an outbound service".

Exceptions: `BcObjectException, BcToolException`

Example:

```
...
import BcAdminOutboundService
import BcAdminProxy
...
proxy_obj=BcAdminProxy.getObject("BCProxy")
os_obj=BcAdminOutboundService.getObject("OSRV",proxy_obj)
os_props=BcAdminOutboundService.getProperties(os_obj)
...
```

9.4.9.4 modifyProperties() – Modify properties of an outbound service

Function: `BcAdminOutboundService.modifyProperties()`

Modifies all the properties of the specified outbound service that is present in the specified dictionary.

Parameters: `bc_object`
(`BcObject` of type `BcObjectType.OUTBOUND_SERVICE`)

Outbound service whose properties are to be modified.

`props`

Dictionary with key-value pairs for the properties that are to be modified. You can find the possible values for key in Section 9.4.9.6, "Properties of an outbound service".

Returns: None

Exceptions: `BcObjectException, BcParameterException, BcToolException`

Example:

```
...
import BcAdminOutboundService
import BcAdminProxy
...
proxy_obj=BcAdminProxy.getObject("BCProxy")
os_obj=BcAdminOutboundService.getObject("OSRV",proxy_obj)
modProps={"reply-timer.sec": "180"}
BcAdminOutboundService.modifyProperties(os_obj,modProps)
...
```

9.4.9.5 remove() – Remove outbound service

Function: `BcAdminOutboundService.remove()`

Removes the specified outbound service from the configuration

Parameters: `bc_object`
(`BcObject` of type `BcObjectType.OUTBOUND_SERVICE`)

The outbound service that is to be removed.

Returns: None

Exceptions: `BcObjectException, BcToolException`

Example:

```
...
import BcAdminOutboundService
import BcAdminProxy
...
proxy_obj=BcAdminProxy.getObject("BCProxy")
os_obj=BcAdminOutboundService.getObject("OSRV",proxy_obj)
BcAdminOutboundService.remove (os_obj)
...
```

9.4.9.6 Properties of an outbound service

The following table contains all the properties of an outbound service.

The meanings and values permitted for the various properties can be found in the Management Console online help system under
Configuring BeanConnect Proxies – Outbound – Outbound Services – Editing an outbound service, General.

Keyword (key) at the MC-CLI [1]	Field name in the GUI [1]	Funct.[1]	Property value[1]
`desc`	**Description**	c / g / m	(String)
`dial-type`	**Type**	cd / g	`BcDef.TYPE_xxx`[2]
`name`	**Partner Service Name**	cf / g	(String)
`reply-timer.sec`	**Reply Timer (sec)**	cd / g / m	(String numeric)

[1] For the meanings of the columns and abbreviations, see Section 9.4.1.2, "Properties"

[2] The possible values of xxx can be found in the JavaDoc (Java class `BcDef`) tn the `JavaDoc` subdirectory of the Management Console installation directory

9.4.10 BcAdminProxy

The module `BcAdminProxy` contains all the functions available for configuring and administering a Management Console proxy.

The module does not contain a create() function with which you can add new proxies to the configuration. The Management Console recognizes newly installed local proxies on start-up and adds these to the configuration automatically. Remote proxies must first be added to the configuration via the graphical user interface.

`BcAdminProxy` contains the functions:

- authenticate() – Authenticate for proxy
- getList() – List all objects of an object type present in a proxy
- getObject() – Read proxy object from the configuration
- getProperties() – Read properties of a proxy
- modifyProperties() – Modify properties of a proxy
- perform() – Start administrative actions for a proxy
- remove() – Remove proxy object from the configuration

9.4.10.1 authenticate() – Authenticate for proxy

Function: `BcAdminProxy.authenticate()`

You use this function to authenticate yourself in order to administer and configure the specified proxy.

Parameters: `bc_object (BcObject` of type `BcObjectType.PROXY`)

Proxy object with which you want to authenticate yourself.

`password (String)`

Proxy's administration password.

Return: `True`

If authentication was successful:

`BcParameterException`

otherwise

Exceptions: `BcObjectException`, `BcParameterException`

Note:

- If authentication is not successfully performed for a `BcObject` of type `BcObjectType.PROXY` then no other function in which this object is specified as a parameter can be executed. This also applies to the functions of other modules, e.g. `BcAdminInboundUser.create()`.
- If you save the proxy's administration password in the graphical user interface then authentication is not required (see the Management Console's online help system: Adding BeanConnect Proxies to the Management Console – Adding a Proxy – Proxy Properties, General: Management Console Access, Admin User Password, Use).

Example:

```
...
import BcAdminProxy
...
BcAdminProxy.authenticate(proxy_obj, admin_pw)
...
```

9.4.10.2 getList() – List all objects of an object type present in a proxy

Function: `BcAdminProxy.getList()`

Reads all the objects of a certain type that are assigned to the proxy and returns a dictionary with the names and objects of this type.

Parameters: `bc_object (BcObject` of type `BcObjectType.PROXY)`

Proxy whose objects of type `list_type` are to be listed.

`list_type` (String)

Type of object of which a list of all objects in the current configuration is to be created. The following values can be specified for `list_type` (see also Section 9.3.2, "Class: BcObjectType"):

String	or the value of one of the following `toString()` methods
`"eis-partner"`	`BcObjectType.EIS_PARTNER.toString()`
`"inbound-msg-endpoint"`	`BcObjectType.INBOUND_MSG_ENDPOINT.toString()`
`"inbound-service"`	`BcObjectType.INBOUND_SERVICE.toString()`
`"inbound-user"`	`BcObjectType.INBOUND_USER.toString()`
`"outbound-comm-endpoint"`	`BcObjectType.OUTBOUND_COMM_ENDPOINT.toString()`
`"outbound-service"`	`BcObjectType.OUTBOUND_SERVICE.toString()`
`"resource-adapter"`	`BcObjectType.RESOURCE_ADAPTER.toString()`

Returns: Dictionary with all objects of this type that are assigned to this proxy together with the name as key.

Exceptions: `BcParameterException, BcObjectException, BcToolException`

Note: If the specified proxy belongs to a proxy cluster then you cannot use `BcAdminProxy.getList()` to read the objects configured in the proxy. They must be read using `BcAdminProxyCluster.getList()`.

Example:

```
...
import BcAdminProxy
...
proxy_obj=BcAdminProxy.getObject("proxy")
bcDicn=BcAdminProxy.getList(proxy_obj,"eis-partner")

...
```

9.4.10.3 getObject() – Read proxy object from the configuration

Function: `BcAdminProxy.getObject()`

Reads the specified proxy from the configuration

Parameters: `proxy_name (String)`

Name of the BeanConnect proxy that is to be read.

Return: `(BcObject` of type `BcObjectType.PROXY)`

The read proxy or `None` if no proxy with a corresponding name exists.

Exceptions: `BcObjectException, BcToolException`

Note: Before you can use the returned object as an input parameter for other functions, you must first perform an authentication (see the function "authenticate() – Authenticate for proxy").

Example:

```
...
import BcAdminProxy
...
proxy_obj=BcAdminProxy.getObject("BCProxy")

...
```

9.4.10.4 getProperties() – Read properties of a proxy

Function: `BcAdminProxy.getProperties()`

Reads all the properties of the specified proxy and returns a dictionary with key-value pairs for the properties.

Parameters: `bc_object (BcObject` of type `BcObjectType.PROXY)`

Proxy whose properties you want to read.

Return: Dictionary with the key-value pairs for all the properties of the proxy (see Section 9.4.10.8, "Properties of a proxy").

Exceptions: `BcObjectException, BcToolException`

Example:

```
...
import BcAdminMain
import BcAdminProxy
...
bcproxies=BcAdminMain.getList("proxy")
proxy_obj=bcproxies["BCProxy"]
proxyProps=BcAdminProxy.getProperties(proxy_obj)
...
```

9.4.10.5 modifyProperties() – Modify properties of a proxy

Function: `BcAdminProxy.modifyProperties()`

Modifies all the properties of the specified property that are present in the specified dictionary.

Parameters: `bc_object (BcObject` of type `BcObjectType.PROXY)`

Proxy whose properties you want to modify.

`props` (Dictionary)

Dictionary with the key-value pairs for the properties that are to be modified. You can find the possible values for key in Section 9.4.10.8, "Properties of a proxy".

Return: None

Exceptions: `BcObjectException, BcParameterException, BcToolException`

Example:

```
...
import BcAdminProxy
...
proxy_obj=BcAdminProxy.getObject("BCProxy")
modProps={"timer.shutdown-time.min": "2"}
BcAdminProxy.modifyProperties(proxy_obj,modProps)
...
```

9.4.10.6 perform() – Start administrative actions for a proxy

Function: BcAdminProxy.perform()

Starts the action specified in action for the proxy.

Parameters: bc_object (BcObject of type BcObjectType.PROXY)

Proxy that is to be administered.

action (String)

Action that is to be started for the specified proxy. Possible values are (see BcDef.ACTION_xxx in the MC-CLI-JavaDoc)

check-adm	Checks the proxy's administration status.
check-avail	Checks the proxy's availability. First of all the availability of the proxy container is checked. This is followed by a check of the proxy's components and communication partners.
restart	Terminates and restarts the proxy.
save	Saves the changes that have been made for this proxy in this session.
start	Starts the proxy.
stop	Terminates the proxy.
update-config	Enters the saved changes in the configuration of the proxy.

params (kw)

Dictionary with the key-value pairs for the parameters that are to be passed to the specified action.

If action=check-avail then the following key-value pair can be specified in the dictionary params:

key="all-components“

value=BcDef.BOOL_TRUE
if the availability of all the components of the proxy is to be checked.
value=BcDef.BOOL_FALSE
if only the availability of the proxy container is to be checked (default).

Returns: (BcObject of type BcObjectType.ACTION)

Contains all information about the started action and all its subactions. To obtain more detailed information, you can call a function of the BcAdminAction module with this object as parameter.

Exceptions: BcParameterException, BcObjectException, BcToolException

Note:

- If the proxy object belongs to a proxy cluster then the "save" action cannot be initiated with BcAdminProxy.perform(). The modified properties are saved directly in the modifyProperties() call or via BcAdminProxyCluster.perform().
- If the proxy object belongs to a proxy cluster then the "update-config" action cannot be initiated with BcAdminProxy.perform(). It must be initiated using BcAdminProxyCluster.perform().

Example:

```
...
import BcAdminProxy
...
proxy_obj=BcAdminProxy.getObject("proxy")
bcaction=BcAdminProxy.perform(proxy_obj,"check-avail")

...
```

9.4.10.7 remove() – Remove proxy object from the configuration

Function: `BcAdminProxy.remove()`

Removes the specified proxy from the configuration.

Parameters: `bc_object (BcObject` of type `BcObjectType.PROXY`)

Proxy object that is to be removed from the configuration.

Return: None

Exceptions: `BcObjectException`, `BcParameterException`.

Example:

```
...
import BcAdminProxy
...
proxy_obj=BcAdminProxy.getObject("BCProxy")
BcAdminProxy.remove(proxy_obj)
...
```

9.4.10.8 Properties of a proxy

The following table contains all the properties of a proxy.

The meanings and values permitted for the various properties can be found in the Management Console online help system under
Managing BeanConnect Proxies – Managing a Proxy via context menu – Edit Properties and under:
Adding BeanConnect Proxies to the Management Console - Expert mode - Adding a Proxy, Timer Settings/ Performance Settings.

Keyword (key) at the MC-CLI [1]	Field name in the GUI [1]	Funct.[1]	Property value[1]
`admin-port`	**MC-CmdHandler Listener Port**	g/ m	(String numeric)
`api-mode`	**Application Program Interface Mode/API Mode**	g/ m c)	`BcDef.API_MODE_xxx` [2]
`appl-process-title`	**Container Application Process Title**	g/ m	(String)
`cluster.cluster-name`	(Node in navigation tree via proxy)	g b)	(String)
`cluster.ip-address`	**IP-Address**	g b)	(String)
`cluster.is-administrable`	**Administrable**	g b)	`BcDef.ADM_STATE_xxx` [2]
`cluster.is-master`	**Master**	g b)	`BcDef.BOOL_xxx` [2]
`dir`	**Container Directory**	g	(String)
`host`	**Host**	g a)	(String)
`id`	**ID**	g	(String)
`name`	**Name**	g/ m	(String)
`partner-type`	**Possible EIS Partner Type**	g/ m c)	`BcDef.PTYPE_xxx` [2]

Keyword (key) at the MC-CLI [1]	Field name in the GUI [1]	Funct.[1]	Property value[1]
`performance.modify-start-par`	**Modify Standard Start Parameter**	g/ m c)	`BcDef.BOOL_xxx` [2]
`performance.nbr-par-inb-rfc1006-conns`	**Number of Parallel Connections: InboundRfc1006**	g/ m c)	(String numeric)
`performance.nbr-par-inb-sock-conns`	**Number of Parallel Connections: InboundSocket**	g/ m c)	(String numeric)
`performance.nbr-par-inb-upic-conns`	**Number of Parallel Connections: InboundUpic**	g/m c)	(String numeric)
`performance.nbr-par-outb-conns`	**Number of Parallel Connections: Outbound**	g c)	(String numeric)
`performance.proxy-cache-size`	**Proxy Container Storage Area Size: Cache**	g/ m c)	(String numeric)
`performance.proxy-mode`	**Proxy Container Mode F(ast)/S(ecure)**	g/ m c)	`BcDef.APPLI_MODE_xxx` [2]
`performance.proxy-nbr-asyn-tasks`	**Number of Proxy Container Processes: asynTasks**	g/ m c)	(String numeric)
`performance.proxy-nbr-tasks`	**Number of Proxy Container Processes: tasks**	g/ m c)	(String numeric)
`performance.proxy-pagepool-size`	**Proxy Container Storage Area Size: page pool**	g/ m c)	(String numeric)
`port`	**ContainerPort**	g	(String numeric)
`run-user-id`	**ContainerRunUserID**	g/ m e)	(String)
`start-as-win-service`	**Start as a service**	g/ m d)	`BcDef.BOOL_xxx` [2]
`timer.asyn-conf.sec`	**Asynch. Confirmation**	g/ m c)	(String numeric)
`timer.eis-partner-reconnect.min`	**EisPartnerReconnect**	g/ m c)	(String numeric)
`timer.prep-to-commit.sec`	**Prepare To Commit**	g/ m c)	(String numeric)

Keyword (key) at the MC-CLI [1]	Field name in the GUI [1]	Funct.[1]	Property value[1]
`timer.ra-check -interval.sec`	**RA Check Interval**	g/ m c)	(String numeric)
`timer.ra-connect-time.sec`	**RA Connection Time**	g/ m c)	(String numeric)
`timer.shutdown-time.min`	**ShutdownTime**	g/ m c)	(String numeric)
`timer.ta-comm.sec`	**Transaction Communication**	g/ m c)	(String numeric)
`win-service-name`	**Windows Service Name**	g d)	(String)

[1] For the meanings of the columns and abbreviations, see Section 9.4.1.2, "Properties"

[2] The possible values of xxx can be found in the JavaDoc (Java class `BcDef`) tn the `JavaDoc` subdirectory of the Management Console installation directory

Notes (general):

- The password and security level can only be modified via the Management Console's graphical user interface (e.g. it is not possible to read and modify the property **Admin User Password** at the MC-CLI). To ensure secure access to the configuration, it is advisable not to store the password permanently but to specify it each time a proxy object is accessed. To do this, you use the function `authenticate()`.
- The property **General/Automatic Availability Check – Time Interval (sec)** is not available in MC-CLI since no automatic checks are performed in MC-CLI.

Notes on the indices in the tables

a) The property `host` cannot be modified.

b) The properties with the prefix "`cluster.`" are only output if the proxy object belongs to a proxy cluster.

c) If the proxy object belongs to a proxy cluster then neither the properties with the prefix "`timer.`" and "`performance.`" nor the properties `partner-type` and `api-mode` can be modified with `BcAdminProxy.modifyProperties()`. These properties must be modified using `BcAdminProxyCluster.modifyProperties()`.

d) The properties `start-as-win-service` and `win-service-name` are not available for proxies on Unix systems (they cannot be read or modified).

e) When a new proxy is added or a reinstallation is performed, the property `run-user-id` must be set. Otherwise it is not possible to execute the perform() function.

9.4.11 BcAdminProxyCluster

The module `BcAdminProxyCluster` contains all the functions available for configuring and administering a Management Console proxy cluster object.

`BcAdminProxyCluster` contains the functions:

- addProxy() – Add proxy to the proxy cluster
- authenticate() – Authenticate for proxy cluster
- create() – Add proxy cluster to the configuration
- getList() – List all objects of a type in the proxy cluster
- getMasterProxy() – Read master proxy of a proxy cluster
- getObject() – Read proxy cluster object from the configuration
- getProperties() – Read properties of a proxy cluster
- modifyProperties() – Modify properties of a proxy cluster
- perform() – Start administrative actions
- remove() – Remove proxy cluster
- removeProxy() – Remove proxy from proxy cluster
- setMasterProxy() – Change master proxy of a proxy cluster

9.4.11.1 addProxy() – Add proxy to the proxy cluster

Function: `BcAdminProxyCluster.addProxy()`

Add a standalone proxy to the specified cluster.

Parameters: `bc_object` (`BcObject` of type `BcObjectType.PROXY_CLUSTER`).

Proxy cluster that is to be extended.

`proxy_object` (`BcObject` of type `BcObjectType.PROXY`).

Proxy that is to be added to the cluster.

Returns: None

Exceptions: `BcObjectException, BcParameterException BcToolException`

Example:

```
...
import BcAdminProxyCluster
import BcAdminProxy
...
clstr_obj=BcAdminProxyCluster.getObject("BCCluster")
proxy_obj=BcAdminProxy.getObject("BCProxy")
proxyList=BcAdminProxyCluster.addProxy(clstr_obj,proxy_obj)
...
```

9.4.11.2 authenticate() – Authenticate for proxy cluster

Function: `BcAdminProxyCluster.authenticate()`

You can use this function to authenticate yourself in order to administer and configure the specified proxy cluster.

Parameters: `bc_object` (`BcObject` of type `BcObjectType.PROXY_CLUSTER`).

Proxy cluster with which you want to authenticate yourself.

`password` (String)

Administration password of the proxy cluster

Returns: `True` if authentication was successful,

`BcParameterException` otherwise

Exceptions: `BcParameterException, BcObjectException`

Note:

- The master proxy's Admin User Password must be specified for the proxy cluster.
- If authentication is not successfully performed for a `BcObject` of type `BcObjectType.PROXY_CLUSTER` then no other function in which this object is specified as a parameter can be executed. This also applies to the functions of other modules, e.g. BcAdminInboundUser.create().
- If you save the master proxy's administration password in the proxy cluster then authentication is not required (see the Management Console's online help system: Adding BeanConnect Proxies to the Management Console – Adding a Proxy – Proxy Properties, General: Management Console Access, Admin User Password, Use.

Example:

```
...
import BcAdminProxyCluster
...
clstr_obj=BcAdminProxyCluster.getObject("BCCluster")
clstr_aut=BcAdminProxyCluster.authenticate(clstr_obj,"BCpass")
...
```

9.4.11.3 create() – Add proxy cluster to the configuration

Function: `BcAdminProxyCluster.create()`

Adds a new proxy cluster, which contains the specified proxy as master proxy, to the configuration.

Parameters: `object_name` (String)

Name of the new BeanConnect proxy cluster

`proxy_object` (`BcObject` of type `BcObjectType.PROXY`)

Proxy that is to be the master proxy of the proxy cluster.

`props`

Dictionary with the key-value pairs for the properties that are to be assigned to the proxy cluster. You can find the possible values for key in Section 9.4.11.13, "Properties of a proxy cluster".

Returns: (`BcObject` of type `BcObjectType.PROXY_CLUSTER`)

The proxy cluster added to the configuration.

Exceptions: `BcParameterException, BcObjectException, BcToolException`

Note: Before you can use the returned object as an input parameter for other functions, you must first perform an authentication (see the function `authenticate()`).

Example:

```
...
import BcAdminProxyCluster
import BcAdminProxy
...
proxy_obj=BcAdminProxy.getObject("BCProxy")
clstr_obj=BcAdminProxyCluster.create("BCCluster",proxy_obj)
...
```

9.4.11.4 getList() – List all objects of a type in the proxy cluster

Function: `BcAdminProxyCluster.getList()`

Reads all the objects of a certain type that are assigned to the proxy cluster and returns a dictionary with the names and objects of this type.

Parameters: `bc_object` (`BcObject` of type `BcObjectType.PROXY_CLUSTER`).

Proxy cluster for which the objects of the specified cluster are to be listed.

`list_type (String)`

Type of object of which a list of all objects in the current configuration is to be created. The following values can be specified for `list_type` (see also Section 9.3.2, "Class: BcObjectType"):

String	or the value of one of the following `toString()` methods
`"eis-partner"`	`BcObjectType.EIS_PARTNER.toString()`
`"inbound-msg -endpoint"`	`BcObjectType.INBOUND_MSG_ ENDPOINT.toString()`
`"inbound-service"`	`BcObjectType.INBOUND_SERVICE.toString()`
`"inbound-user"`	`BcObjectType.INBOUND_USER.toString()`
`"outbound-comm -endpoint"`	`BcObjectType.OUTBOUND_COMM_ENDPOINT. toString()`
`"outbound-service"`	`BcObjectType.OUTBOUND_SERVICE.toString()`
`"proxy"`	`BcObjectType.PROXY.toString()`
`"resource -adapter"`	`BcObjectType.RESOURCE_ADAPTER.toString()`

Returns: Dictionary with all the objects of this type that are assigned to this proxy cluster and the names as key.

Exceptions: `BcObjectException, BcParameterException, BcToolException`

Example:

```
...
import BcAdminProxyCluster
...
clstr_obj=BcAdminProxyCluster.getObject("BCCluster")
proxyList=BcAdminProxyCluster.getList(clstr_obj,"proxy")

...
```

9.4.11.5 getMasterProxy() – Read master proxy of a proxy cluster

Function: `BcAdminProxyCluster.getMasterProxy()`

Returns the master proxy of the proxy cluster.

Parameters: `bc_object` (`BcObject` of type `BcObjectType.PROXY_CLUSTER`)

Proxy cluster whose master proxy is to be read.

Returns: (`BcObject` of type `BcObjectType.PROXY`).

Master proxy of the proxy cluster

Exceptions: `BcObjectException, BcParameterException, BcToolException`

Example:

```
...
import BcAdminProxyCluster
...
clstr_obj=BcAdminProxyCluster.getObject("BCCluster")
m_proxy_obj=BcAdminProxyCluster.getMasterProxy(clstr_obj)

...
```

9.4.11.6 getObject() – Read proxy cluster object from the configuration

Function: `BcAdminProxyCluster.getObject()`

Reads the proxy cluster with the specified name from the configuration

Parameters: `cluster_name` (String)

Name of the BeanConnect proxy cluster that is to be read.

Returns: (`BcObject` of type `BcObjectType.PROXY_CLUSTER`).

The read proxy cluster or `None` if no proxy cluster with a corresponding name exists.

Exceptions: `BcObjectException, BcToolException`

Note: Before you can use the returned object as a call parameter for other functions, you must first perform an authentication (see the function `authenticate()`).

Example:

```
...
import BcAdminProxyCluster
...
clstr_obj=BcAdminProxyCluster.getObject("BCCluster")

...
```

9.4.11.7 getProperties() – Read properties of a proxy cluster

Function: `BcAdminProxyCluster.getProperties()`

Reads all the properties of the specified proxy cluster and returns a dictionary with key-value pairs for the properties. You can find the possible values for key in Section 9.4.11.13, "Properties of a proxy cluster".

Parameters: `bc_object` (`BcObject` of type `BcObjectType.PROXY_CLUSTER`).

Proxy cluster whose properties are to be read.

Returns: Dictionary with the key-value pairs for all the properties of the proxy cluster.

Exceptions: `BcObjectException, BcToolException`

Example:

```
...
import BcAdminProxyCluster
...
clstr_obj=BcAdminProxyCluster.getObject("BCCluster")
clusterProps=BcAdminProxyCluster.getProperties(clstr_obj)

...
```

9.4.11.8 modifyProperties() – Modify properties of a proxy cluster

Function: `BcAdminProxyCluster.modifyProperties()`

Modifies the properties of the specified proxy cluster. You must pass the new values of the properties that are to be modified to the function in a dictionary.

Parameters: `bc_object` (`BcObject` of type `BcObjectType.PROXY_CLUSTER`).

Proxy cluster whose properties are to be modified.

`props`

Dictionary with the key-value pairs for the properties that are to be modified. You can find the possible values for key in Section 9.4.11.13, "Properties of a proxy cluster".

Returns: None

Exceptions: `BcObjectException, BcParameterException BcToolException`

Example:

```
...
import BcAdminProxyCluster
...
modProps={"timer.shutdown-time.min": "2"}
clstr_obj=BcAdminProxyCluster.getObject("BCCluster")
BcAdminProxyCluster.modifyProperties(clstr_obj,modProps)

...
```

9.4.11.9 perform() – Start administrative actions

Function: `BcAdminProxyCluster.perform()`

Starts a given action for the proxy cluster.

Parameters: `bc_object` (`BcObject` of type `BcObjectType.PROXY_CLUSTER`).

Proxy cluster for which the action is to be started.

`action (String)`

Action that is to be started for the specified proxy. Possible values are (see `BcDef.ACTION_xxx` in the MC-CLI-JavaDoc)

`check-adm`	Checks the proxy cluster's administration status.
`check-avail`	Checks the proxy cluster's execution status.
`restart`	Terminates and restarts the proxy cluster.
`save`	Saves the changes that have been made for this proxy cluster in the current session.
`start`	Starts the proxy cluster.
`stop`	Terminates the proxy cluster.
`update-config`	Enters the saved changes in the configuration of the proxy cluster.

Returns: (`BcObject` of type `BcObjectType.ACTION`).

Contains all information about the started action and all its subactions. To obtain more detailed information, you can call a function of the `BcAdminAction` module with this object as parameter.

Exceptions: `BcParameterException, BcObjectException, BcToolException`

Example:

```
...
import BcAdminProxyCluster
...
clstr_obj=BcAdminProxyCluster.getObject("BCCluster")
bcaction=BcAdminProxyCluster.perform(clstr_obj,"stop")

...
```

9.4.11.10 remove() – Remove proxy cluster

Function: `BcAdminProxyCluster.remove()`

Removes the specified proxy cluster from the configuration.

Parameters: `bc_object` (`BcObject` of type `BcObjectType.PROXY_CLUSTER`).

Proxy cluster that is to be removed.

Returns: None

Exceptions: `BcObjectException, BcToolException`

Example:

```
...
import BcAdminProxyCluster
...
clstr_obj=BcAdminProxyCluster.getObject("BCCluster")
BcAdminProxyCluster.remove(clstr_obj)

...
```

9.4.11.11 removeProxy() – Remove proxy from proxy cluster

Function: `BcAdminProxyCluster.removeProxy()`

Removes the specified proxy from the proxy cluster and enters it in the list of individual (standalone) proxies.

Parameters: `bc_object` (`BcObject` of type `BcObjectType.PROXY_CLUSTER`).

Proxy cluster in which the proxy is to be removed.

`proxy_object` (`BcObject` of type `BcObjectType.PROXY`).

Proxy that is to be removed from the cluster.

Returns: None

Exceptions: `BcObjectException, BcParameterException, BcToolException`

Note:

- If the specified proxy is the master proxy and the proxy cluster contains further proxies then the function call is rejected.
- When the last proxy is removed from a cluster then the proxy cluster is also automatically removed.

Example:

```
...
import BcAdminProxyCluster
import BcAdminProxy
...
clstr_obj=BcAdminProxyCluster.getObject("BCCluster")
proxy_dicn=BcAdminProxyCluster.getList(clstr_obj, "proxy")
proxy_obj=proxy_dicn["BCProxy"]
BcAdminProxyCluster.removeProxy(clstr_obj,proxy_obj)

...
```

9.4.11.12 setMasterProxy() – Change master proxy of a proxy cluster

Function: `BcAdminProxyCluster.setMasterProxy()`

Sets another proxy as the master proxy of the proxy cluster.

Parameters: `bc_object` (`BcObject` of type `BcObjectType.PROXY_CLUSTER`)

Proxy cluster in which the proxy is to become the new master proxy.

`proxy_object` (`BcObject` of type `BcObjectType.PROXY`).

Proxy that is to be the new master proxy of the proxy cluster.

Returns: None

Exceptions: `BcObjectException, BcParameterException, BcToolException`

Example:

```
...
import BcAdminProxyCluster
import BcAdminProxy
...
clstr_obj=BcAdminProxyCluster.getObject("BCCluster")
proxy_dicn=BcAdminProxyCluster.getList(clstr_obj, "proxy")
new_mproxy_obj=proxy_dicn["BCProxy"]
BcAdminProxyCluster.setMasterProxy(clstr_obj, new_mproxy_obj)

...
```

9.4.11.13 Properties of a proxy cluster

The following table contains all the properties of a proxy cluster.

The meanings and values permitted for the various properties can be found in the Management Console online help system under
BeanConnect Proxy Clusters - Configuring BeanConnect Proxy Clusters - Proxy cluster, Properties
and under:
Adding BeanConnect Proxies to the Management Console - Expert mode - Adding a Proxy, Timer Settings/ Performance Settings.

Keyword (key) at the MC-CLI [1]	Field name in the GUI [1]	Funct.[1]	Property value[1]
`api-mode`	**Application Interface Mode/API Mode**	cd / g / m	`BcDef.API_MODE_ xxx`[2]
`id`	**ID**	g	(String)
`name`	**Name**	cf / g / m	(String)
`partner-type`	**Possible EIS Partner Types**	cd / g / m	`BcDef.PTYPE_xxx`[2]
`performance. modify-start-par`	**Modify Standard Start Parameter**	cd / g / m	`BcDef.BOOL_xxx`[2]
`performance. nbr-par -inb-rfc1006-conns`	**Number of Parallel Connections: InboundRfc1006**	cd / g / m	(String numeric)
`performance. nbr-par-inb-sock-conns`	**Number of Parallel Connections: InboundSocket**	cd / g / m	(String numeric)
`performance. nbr-par-inb-upic-conns`	**Number of Parallel Connections: InboundUpic**	cd / g / m	(String numeric)
`performance. nbr-par-outb-conns`	**Number of Parallel Connections: Outbound**	g	(String numeric)
`performance. proxy-cache-size`	**Proxy Container Storage Area Size: Cache**	cd / g / m	(String numeric)

Keyword (key) at the MC-CLI [1]	Field name in the GUI [1]	Funct.[1]	Property value[1]
`performance.proxy-mode`	**Proxy Container Mode F(ast)/S(ecure)**	cd / g / m	`BcDef.APPLI_MODE_xxx`[2]
`performance.proxy-nbr-asyn-tasks`	**Number of Proxy Container Processes: asynTasks**	cd / g / m	(String numeric)
`performance.proxy-nbr-tasks`	**Number of Proxy Container Processes: tasks**	cd / g / m	(String numeric)
`performance.proxy-pagepool-size`	**Proxy Container Storage Area Size: page pool**	cd / g / m	(String numeric)
`timer.asyn-conf.sec`	**Asynch. Confirmation**	cd / g / m	(String numeric)
`timer.eis-partner-reconnect.min`	**EisPartnerReconnect**	cd / g / m	(String numeric)
`timer.prep-to-commit.sec`	**Prepare To Commit**	cd / g / m	(String numeric)
`timer.ra-check-interval.sec`	**RA Check Interval**	cd / g / m	(String numeric)
`timer.ra-connect-time.sec`	**RAConnectionTime**	cd / g / m	(String numeric)
`timer.shutdown-time.min`	**ShutdownTime**	cd / g / m	(String numeric)
`timer.ta-comm.sec`	**Transaction Communication**	cd / g / m	(String numeric)

[1] For the meanings of the columns and abbreviations, see Section 9.4.1.2, "Properties"

[2] The possible values of xxx can be found in the JavaDoc (Java class `BcDef`) tn the `JavaDoc` subdirectory of the Management Console installation directory

i The property "General/Automatic Availability Check – Time Interval (sec)" is not available in MC-CLI since no automatic checks are performed in the MC-CLI.

9.4.12 BcAdminRA

The module `BcAdminRa` contains all the functions available for configuring and administering a Management Console resource adapter object.

`BcAdminRA` contains the functions:

- create() – Add resource adapter to the configuration
- getObject() – Read resource adapter object from the configuration
- getProperties() – Read properties of a resource adapter
- modifyProperties() – Modify properties of a resource adapter
- perform() – Start administrative actions
- remove() – Remove resource adapter

9.4.12.1 create() – Add resource adapter to the configuration

Function: `BcAdminRA.create()`

A resource adapter is added to the configuration. You must pass the properties of the resource adapter to the MC-CLI in a dictionary.

Parameters: `object_name` (String)

Name of the BeanConnect resource adapter that is to be added to the configuration.

`proxy_object`
(`BcObject` of type `BcObjectType.PROXY` / `BcObjectType.PROXY_CLUSTER`)

Proxy or proxy cluster to which the resource adapter is to be assigned.

`props` (kw)

Dictionary with the key-value pairs for the properties that are to be assigned to the resource adapter. You can find the possible values for key in Section 9.4.12.7, "Properties of a resource adapter". The property `host` must be specified.

Returns: `(BcObject` of type `BcObjectType.RESOURCE_ADAPTER)`

The resource adapter added to the configuration.

Exceptions: `BcParameterException, BcObjectException, BcToolException`

Note: The specification of a `proxy_object` of type `BcObjectType.PROXY` is only permitted if the proxy does not belong to a proxy cluster. In the case of proxies belonging to a cluster, the proxy cluster object must be specified as a parameter here.

Example:

```
...
import BcAdminRA
import BcAdminProxy
...
proxy_obj=BcAdminProxy.getObject("BCProxy")
creProps ={"host":"xyz","port-inbound":"1234","desc": "created",\
                            "ta-logging": BcDef.TA_LOGGING_NONE}

newRA=BcAdminRA.create("test-RA",proxy_obj,creProps)

...
```

9.4.12.2 getObject() – Read resource adapter object from the configuration

Function: `BcAdminRA.getObject()`

Reads the resource adapter with the specified name from the configuration

Parameters: `object_name` (String)

Name of the BeanConnect resource adapter that is to be read.

`proxy_object`
(`BcObject` of type `BcObjectType.PROXY` / `BcObjectType.PROXY_CLUSTER`)

Proxy or proxy cluster to which the resource adapter is assigned.

The specification of a `proxy_object` of type `BcObjectType.PROXY` is only permitted if the proxy does not belong to a proxy cluster. In the case of proxies belonging to a cluster, the proxy cluster object must be specified as a parameter here.

Returns: (`BcObject` of type `BcObjectType.RESOURCE_ADAPTER`).

The read resource adapter or `None` if no resource adapter with a corresponding name exists.

Exceptions: `BcObjectException, BcParameterException, BcToolException`

Example:

```
...
import BcAdminRA
import BcAdminProxy
...
proxy_obj=BcAdminProxy.getObject("BCProxy")
ra_obj=BcAdminRA.getObject("test-RA".proxy_obj)

...
```

9.4.12.3 getProperties() – Read properties of a resource adapter

Function: `BcAdminRa.getProperties()`

Reads all the properties of the specified resource adapter and returns a dictionary with key-value pairs for the properties. You can find the possible values for key in Section 9.4.12.7, "Properties of a resource adapter".

Parameters: `bc_object` (`BcObject` of type `BcObjectType.RESOURCE_ADAPTER`).

Resource adapter whose properties are to be read.

Returns: Dictionary with the key-value pairs for all the properties of the resource adapter.

Exceptions: `BcObjectException, BcToolException`

Example:

```
...
import BcAdminRA
import BcAdminProxy
...
proxy_obj=BcAdminProxy.getObject("BCProxy")
ra_obj=BcAdminRA.getObject("test-RA", proxy_obj)
raProps=BcAdminRA.getProperties(ra_obj)

...
```

9.4.12.4 modifyProperties() – Modify properties of a resource adapter

Function: `BcAdminRa.modifyProperties()`

Modifies all the properties of the specified resource adapter that are present in the specified dictionary.

Parameters: `bc_object` (`BcObject` of type `BcObjectType.RESOURCE_ADAPTER`).

Resource adapter whose properties are to be modified.

`props`

Dictionary with the key-value pairs for the properties that are to be modified. For possible values of key, see Section 9.4.12.7, "Properties of a resource adapter"

Returns: None

Exceptions: `BcObjectException, BcParameterException`, `BcToolException`

Example:

```
...
import BcAdminRA
import BcAdminProxy
...
proxy_obj=BcAdminProxy.getObject("BCProxy")
ra_obj=BcAdminRA.getObject("test-RA", proxy_obj)
modProps={"desc":"modified"}
BcAdminRA.modifyProperties(ra_obj,modProps)

...
```

9.4.12.5 perform() – Start administrative actions

Function: `BcAdminRA.perform()`

Starts the specified action for the resource adapter.

Parameters: `bc_object` (`BcObject` of type `BcObjectType.RESOURCE_ADAPTER`).

Resource adapter for which the action is to be started.

`action (String)`

Action that is to be started for the specified resource adapter. Possible values are (see `BcDef.ACTION_xxx`)

`check-avail`	Checks the resource adapter's execution status.
`update-ra-xml`	Enters the saved changes in the configuration of the resource adapter.

`params (kw)` (Dictionary)

Dictionary with the key-value pairs for the parameters that are to be passed to the specified action.

If `action="update-ra-xml"` then the following key-value pair must be specified in the dictionary `params`:

`key="rar-file-path"`
`value=` Name of the resource adapter rar file in which the changes of the configuration are to be entered.

Returns: For `action="update-ra-xml"`:
True (boolean) if update has been initiated, otherwise BcToolException.

For `action="check-avail"`:
`bcaction` (`BcObject` of type `BcObjectType.ACTION`)
containing all the information about the started action and all its subactions. To obtain more detailed information, you can call a function of the `BcAdminAction` module with this object as parameter.

Exceptions: `BcObjectException, BcParameterException, BcToolException`

Example:

```
...
import BcAdminRA
import BcAdminProxy
...
proxy_obj=BcAdminProxy.getObject("BCProxy")
ra_obj=BcAdminRA.getObject("test-RA", proxy_obj)
bcaction=BcAdminRA.perform(ra_obj,"check-avail")

...
```

9.4.12.6 remove() – Remove resource adapter

Function: `BcAdminRA.remove()`

Removes the specified resource adapter from the configuration.

Parameters: `bc_object` (`BcObject` of type `BcObjectType.RESOURCE_ADAPTER`).

Resource adapter that is to be removed.

Returns: None

Exceptions: `BcObjectException, BcToolException`

Example:

```
...
import BcAdminRA
import BcAdminProxy
...
proxy_obj=BcAdminProxy.getObject("BCProxy")
ra_obj=BcAdminRA.getObject("test-RA", proxy_obj)
BcAdminRA.remove(ra_obj)

...
```

9.4.12.7 Properties of a resource adapter

The following table contains all the properties of a resource adapter.

The meanings and values permitted for the various properties can be found in the Management Console online help system under
Configuring BeanConnect Proxies - Resource Adapters - Resource Adapter, Properties.

Keyword (key) at the MC-CLI[1]	Field name in the GUI[1]	Funct. in standalone proxy[1]	Funct. in proxy cluster [1]	Property value[1]
`admin-port`	**McCmdHandlerPort**	c / g / m	c / g / m	(String numeric)
`admin-pw`	**McCmdHandler Password**	c / m	c / m	(String)
`desc`	**Description**	c / g / m	-	(String)
`host`	**Host**	cf / g / m	cf / g / m	(String)
`id`	**ID**	g	-	(String numeric)
`index`	**Index**	g	-	(String numeric)
`log4j-config-file-path`	**Log4J Configuration File Path**	c / g / m	c / g / m	(String)
`name`	**Name**	cd / g / m	c	(String)
`port-inbound`	**Listener Port**	c / g / m	c / g / m	(String numeric)
`proxy-reconnect-count`	**ProxyReconnectCount**	–	c / g / m(s)	(String numeric)
`proxy-reconnect-interval.min`	**ProxyReconnect Interval**	–	c / g / m(s)	(String numeric)
`proxy-url-outbound`	**Proxy URL for OLTP Outbound Communication**	g	-	(String)
`ta-log-dir`	**TransactionLoggingDir**	cd / g / m	c / g / m(s)	(String)
`ta-logging`	**TransactionLogging**	cd / g / m	c / g / m(s)	`BcDef.TA_LOGGING_xxx` [2]

[1] For the meanings of the columns and abbreviations, see Section 9.4.1.2, "Properties"

[2] The possible values of xxx can be found in the JavaDoc (Java class `BcDef`) tn the `JavaDoc` subdirectory of the Management Console installation directory

If the properties `proxy-reconnect-count`, `proxy-reconnect-interval.min`, or `ta-log-dir, ta-logging` are modified for a resource adapter that belongs to a proxy cluster then all the resource adapters in this cluster are synchronized automatically when the change is saved.

9.4.13 BcAdminTodo

The module `BcAdminTodo` contains all the functions available for configuring and administering a Management Console todo topic object. Because todo topics do not have a name via which they can be addressed, they can only be read via the list function `BcAdminMain.getList(BcObjectType.TODO.toString())`.

`BcAdminTodo` contains the functions:

- getProperties() – Read properties of a todo topic
- remove() – Delete todo topic

9.4.13.1 getProperties() – Read properties of a todo topic

Function: `BcAdminTodo.getProperties()`

Reads all the properties of the specified todo topic and returns a dictionary with key-value pairs for the properties.

Parameters: `bc_object` (`BcObject` of type `BcObjectType.TODO`)

The todo topic whose properties are to be read.

Returns: Dictionary with the key-value pairs for all the properties of the todo topic. You can find the possible values for key in Section 9.4.13.3, "Properties of a todo topic".

Exceptions: `BcObjectException, BcToolException`

Example:

```
...
import BcAdminTodo
...
todoProps=BcAdminTodo.getProperties(elem)
...
```

9.4.13.2 remove() – Delete todo topic

Function: `BcAdminTodo.remove()`

Deletes the specified todo topic from the configuration.

Parameters: `bc_object` (`BcObject` of type `BcObjectType.TODO`)

The todo topic that is to be deleted.

Returns: None

Exceptions: `BcObjectException, BcToolException`

Example:

```
...
import BcAdminTodo
...
BcAdminTodo.remove(elem)
...
```

9.4.13.3 Properties of a todo topic

The following table contains all the properties of a todo topic.

The meanings and values permitted for the various properties can be found in the Management Console online help system under
Todo Topics - Todo topic, Properties.

Keyword (key) at the MC-CLI [1]	Field name in the GUI [1]	Funct. [1]	Property value[1]
`comm-service`	**Communication Service**	g	(String)
`components`	**Component(s)**	g	(String)
`eis-partner`	**EIS Partner**	g	(String)
`internal`	**Internal**	g	`BcDef.BOOL_xxx`[2]
`lu62gateway`	**openUTM-LU62 Gateway**	g	(String)
`proxy`	**Proxy**	g	(String)
`proxy-cluster`	**Proxy cluster**	g	(String)
`related-file-name`	**Related Configuration File Name**	g	(String)
`text`	**Text**	g	(String)
`time`	**Time**	g	(String)

[1] For the meanings of the columns and abbreviations, see Section 9.4.1.2, "Properties"

[2] The possible values of xxx can be found in the JavaDoc (Java class `BcDef`) tn the `JavaDoc` subdirectory of the Management Console installation directory

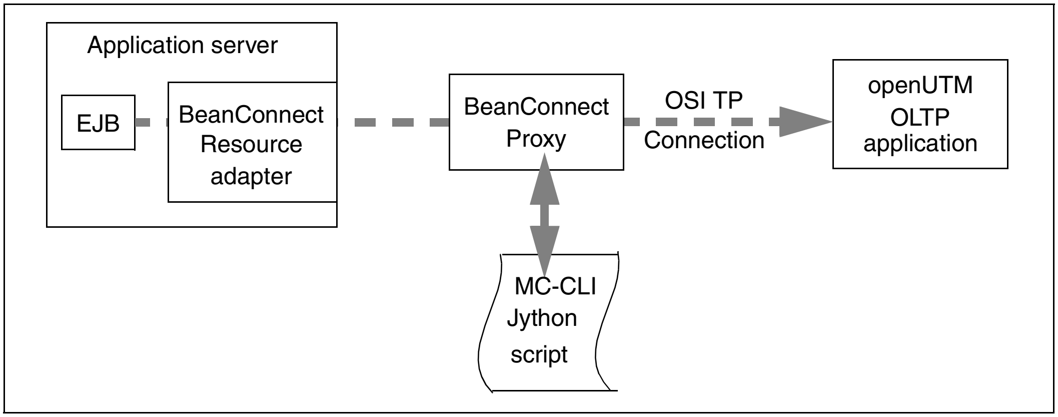
9.5 Application scenarios (examples)

This section describes the procedure to be followed for different application scenarios. It describes the configuration for both outbound and inbound communication with an openUTM application. In addition, you will also find examples of administrative activities that you can automate by means of a script and a list of sample Jython scripts that are supplied together with BeanConnect.

i Before configuration can be performed via the MC-CLI, either a local proxy must be installed or a remote proxy must be added via the Management Console's graphical user interface.

9.5.1 Configuring outbound communication with an openUTM application

Figure 59: Configuring outbound communication with an openUTM application



To configure outbound communication with an openUTM application, you must call the following MC-CLI functions in the specified order in a Jython script. You will find an example of the configuration of outbound communication in the supplied Jython sample script `sampleAdminOutbound.py` located in the directory `cli-sample`.

1. `BcAdminMain.init()`
   Call the function `BcAdminMain.init(<console_home>)` to start a Management Console session.

   The Management Console automatically recognizes an installed local proxy and opens the corresponding configuration file `console.properties.xml` or creates this if it does not already exist.

2. **`BcAdminMain.getlist()`** / **`BcAdminProxy.getObject()`**
   Read the list of all configured proxies using `BcAdminMain.getList(“proxy“)` or use `BcAdminProxy.getObject(<proxy_name>)` to read a specific proxy object.

   You can use the values returned by the calls as the call parameter `<bcproxy>` in the calls listed in points 3 to 8.

3. **`BcAdminProxy.authenticate()`**
   Authenticate yourself for the proxy by calling `BcAdminProxy.authenticate(<bcproxy>, admin_pw)`.

4. **`BcAdminProxy.modifyProperties()`**
   Configure the proxy by calling the function `BcAdminProxy.modifyProperties(<bcproxy>, properties)` with the desired property values in your script.

5. **`BcAdminEisPartner.create()`**
   Define one (or more) EIS system(s) by calling the function `BcAdminEisPartner.create(<eisPartnerName>,<bcproxy>, properties)` (multiple times if necessary).

6. **`BcAdminOutboundService.create()`**
   Define one (or more) outbound service(s) by calling the function `BcAdminOutboundService.create(<remoteTacName>,<bcproxy>, properties)` (multiple times if necessary).

7. **`BcAdminOutboundCommEndpoint.create()`**
   Define one (or more) outbound communication endpoint(s) by calling the function `BcAdminOutboundCommEndpoint.create(<communicationEndpointName>, <bcproxy>, properties)` (multiple times if necessary).

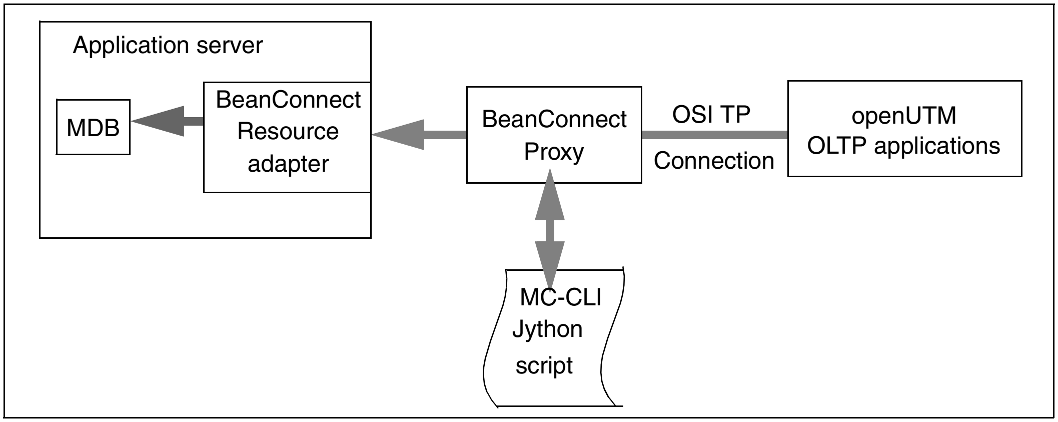
8. **`BcAdminProxy.perform()`**
   Save and update the proxy configuration using the functions `BcAdminProxy.perform(<bcproxy>, "save")` and `BcAdminProxy.perform(<bcproxy>, "update-config")`.

9. **`BcAdminEisPartner.perform()`**
   Call the function `BcAdminEisPartner.perform(<eisPartner>, "gen-config")` to save the configuration of the EIS partner(s).

10. **`BcAdminMain.close()`**
    Call the function `BcAdminMain.close()` to terminate the Management Console session.

9.5.2 Configuring inbound communication with an openUTM application

Figure 60: Configuring inbound communication with an openUTM application



For the existing proxy configuration from Section 9.5.1, "Configuring outbound communication with an openUTM application", it is also possible to call a script to configure inbound communication. You will find an example of the configuration of inbound communication in the supplied Jython sample script `sampleAdminInbound.py` located in the directory `cli-sample`.

1. **`BcAdminMain.init()`**
   Call the function `BcAdminMain.init(<console_home>)` to start a Management Console session.

   The Management Console reads the configuration file `console.properties.xml` and uses the configuration described above

2. **`BcAdminMain.getlist()`** / **`BcAdminProxy.getObject()`**
   Read the list of all configured proxies using `BcAdminMain.getList(“proxy“)` or read a specific proxy object using `BcAdminProxy.getObject(<proxy_name>)`.

3. **`BcAdminProxy.authenticate()`**
   Authenticate yourself for the proxy by calling `BcAdminProxy.authenticate(<bcproxy>, admin_pw)`.

4. **`BcAdminRA.create()`**
   Define one (or more) resource adapter(s) by calling the function `BcAdminRA.create(<resourceAdapterName>,<bcproxy>, properties)` (multiple times if necessary).

5. `BcAdminInboundMsgEndpoint.create()`
   Set up one (or more) inbound message endpoint(s) by calling the function `BcAdminInboundMsgEndpoint.create(<messageEndpointName>, <bcproxy>, properties)` (multiple times if necessary).

6. `BcAdminInboundUser.create()`
   If the EIS partner makes use of users, define one (or more) inbound user(s) by calling the function `BcAdminInboundUser.create(<userName>, <bcproxy>, properties)` (multiple times if necessary).

7. `BcAdminInboundService.modifyProperties()`
   Define the properties of the corresponding UTM transaction code by calling the function `BcAdminInboundService.modifyProperties (<InboundService>, properties)` (multiple times if necessary).

8. `BcAdminProxy.perform()`
   Save and update the proxy configuration using the functions `BcAdminProxy.perform(<bcproxy>, "save")` and `BcAdminProxy.perform(<bcproxy>, "update-config")`.

9. `BcAdminRA.perform()`
   Call the function `BcAdminRa.perform(<resourceAdapter>,"update-ra-xml", params)` to save the configuration of the resource adapter(s).

10. `BcAdminMain.close()`
    Call the function `BcAdminMain.close()` to terminate the Management Console session.

9.5.3 Administer proxies

When administering proxy components, it is possible to imagine various application scenarios in which the use of scripts is more convenient than administration via the Management Console GUI. Examples of this are:

- All the proxies, or all the proxies running on a given host, are to be shut down or restarted, for example in order to update the configuration.
- The reply timer is to be deactivated for all inbound message endpoints of `type=dialog`, for example due to network problems.
- Once the network problems have been resolved, the reply timer is to be set to the default value once again.

9.5.4 Jython sample scripts

Following installation of the Management Console, the Management Console installation directory contains the subdirectory `cli-sample` in which you will find a start script and various Jython sample scripts (`xxx.py`):

- **`startBcAdmin.cmd`** (for Windows) or **`startBcAdmin.sh`** (for Unix platforms) – start script
- **`sampleAdminMain.py`** – Jython script with functions for the module `BcAdminMain`.
- **`sampleAdminProxy.py`** – Jython script with functions for the module `BcAdminProxy`.
- **`sampleAdminProxyCluster.py`** – Jython script with functions for the module `BcAdminProxyCluster`
- **`sampleAdminRa.py`** – Jython script with functions for the module `BcAdminRA`
- **`sampleAdminEisPartner`**.py – Jython script with functions for the module `BcAdminEisPartner`
- **`sampleAdminInbound.py`** – Jython script with functions for the modules `BcAdminInboundUser`, `BcAdminInboundService`, `BcAdminInboundMsgEndpoint`
- **`sampleAdminOutbound.py`** – Jython script with functions for the modules `BcAdminOutboundService`, `BcAdminOutboundCommEndpoint`

Notes on using the Jython sample scripts:

- Every Jython sample script can be started as a main program. It requires at a minimum the console directory (console_home) as the first argument.
- Every Jython sample script (with the exception of `sampleAdminMain.py`) can be inserted (once only) in another script (as a "subroutine" using an `import` statement.
- The start script `startBcAdmin.cmd`/`startBcAdmin.sh` calls the Jython sample script `sampleAdminMain`. The required function scope can be specified as the second argument:
  - "proxy"
    starts `sampleAdminProxy.py` (as import).
  - "all"
    starts all other imported sample scripts `sampleAdmin`Xxx (as import).

- In all other Jython scripts it is possible to enable or disable the functionality via switches.
  - `bCreDel=True/False`
    Generate and remove test objects (the object names start with "test"/"TEST")
  - `bModObjs=True/False`
    Modify test objects.
  - `bSaveMod=True/False`
    Save configuration changes.
  - Other script-specific switches (see also the comments in the script files).
- In most cases, `bCreDel=bModObjs=bSaveMod=True` creates an object in a first pass through the scripts. This object is then modified in a second pass and removed in a third. The sequence of the functions is controlled by the value of the property `desc` (`"created"` -> `"modified"` -> remove object):
  - If a test object does not exist then it is created with `desc="created"`.
  - If a test object exists and `desc= created"`, then the object is modified (desc="`modified`").
  - If a test object exists and `desc="modified"`, then the object is removed from the configuration.
- To run a sample script, you must first replace all the placeholders("`****`") in the script file by valid values. In particular, you must enter the following in all scripts:
  - `consoleHome`
    Management Console installation directory if the script is to be started as a main program and no argument is passed.
  - `proxy_name`
    Name of the proxy for which the functions are to be executed (not in the case of `sampleAdminMain.py`).
- Before a script is able to run, you must at least modify the environment variable JYTHONPATH.

10 Interfaces and programming

This chapter provides information on the following topics:

- BeanConnect-specific interfaces and Common Client Interface (CCI)
- Programming outbound communication
- Programming inbound communication

Before any communication between an EJB and an EIS application can take place, both the configuration data of BeanConnect and the configuration of the EIS application have to be set up properly.

- On outbound communication, an EJB that is deployed in a Java EE application server calls a service in the EIS partner.
- On inbound communication, an EIS partner sends messages to an OLTP message-driven bean that is deployed in a Java EE application server.

> ℹ The JavaDoc of BeanConnect, which is often referred to in the course of this chapter is supplied with the resource adapter JAR file `BC30A00_RA.jar` and is available after the installation of the resource adapter.
>
> The JavaDoc is located in the resource adapter's installation directory:
>
> - In Windows systems, under `JavaDoc\api\index.html`
> - In Unix/Linux systems under `JavaDoc/api/index.html`

10.1 BeanConnect-specific interfaces and Common Client Interface (CCI)

BeanConnect offers two different sets of interfaces:

- BeanConnect-specific interfaces
- Common Client Interface (CCI)

Recommendations: BeanConnect-specific interfaces or CCI

The Common Client Interface (CCI) is defined in the JCA specification of Sun Microsystems^{TM}. The CCI defines a standard API and addresses primarily the needs of application development tools and EAI Frameworks (Enterprise Application Integration). If you are familiar with the CCI in the JCA specification, it makes sense to use the CCI.

In any other case it is advisable to use the BeanConnect-specific interfaces, because the associated programming effort is considerably reduced.

Differences between the BeanConnect-specific interfaces and CCI

For outbound communication, BeanConnect-specific interfaces and the CCI offer virtually the same functionality.

For OLTP communication with openUTM partners and for CICS partners, the BeanConnect-specific `EISOltpConnection` interface and the CCI offer the identical functionality.
The BeanConnect-specific `EISUpicConnection` interface merely provides additional functionality for UPIC connections to openUTM partners (see "Additional functionality provided by the EISUpicConnection interface" on page 385).

For inbound communication, the BeanConnect-specific `OltpMessageListener` interface and the CCI offer the same functionality for dialog communication. Additionally, BeanConnect provides the `AsyncOltpMessageListener` interface for asynchronous communication.

Selecting the interfaces to be used

For outbound communication, you specify the interface to be used as follows:

- In the application server-specific deployment descriptor file `weblogic-ra.xml` of the resource adapter, you specify the connection factory interface in the `<connection-factory-interface>` entry:
  - For the BeanConnect-specific interfaces:

    `net.fsc.jca.communication.EISOltpConnectionFactory`
    or
    `net.fsc.jca.communication.EISUpicConnectionFactory` (only if the EIS partner is an openUTM application)
  - For CCI:

    `net.fsc.jca.communication.cci.BCOltpConnectionFactory`
    or
    `net.fsc.jca.communication.cci.BCUpicConnectionFactory` (only if the EIS partner is an openUTM application)
- In the deployment descriptor file `ejb-jar.xml` of an EJB, you specify the connection factory interface in the `<res-type>` entry:
  - For the BeanConnect-specific interfaces:

    `net.fsc.jca.communication.EISOltpConnectionFactory` or
    `net.fsc.jca.communication.EISUpicConnectionFactory`
  - For CCI:

    `net.fsc.jca.communication.cci.BCOltpConnectionFactory` or
    `net.fsc.jca.communication.cci.BCUpicConnectionFactory`

For inbound communication, you specify your favored interface in the deployment descriptor file `ejb-jar.xml` of the message-driven bean.

You specify the interface in the `<messaging-type>` entry:

- For the BeanConnect-specific interfaces:

  `net.fsc.jca.communication.AsyncOltpMessageListener` or
  `net.fsc.jca.communication.OltpMessageListener`
- For CCI:

  `javax.resource.cci.MessageListener`

10.2 Programming outbound communication

For outbound communication with an openUTM partner application, BeanConnect supports the communication protocol OSI-TP, which can be used for distributed transaction processing as well as for non-transactional communication, and the proprietary, non-transactional protocol UPIC for access to openUTM applications.

For outbound communication with an CICS partner application, BeanConnect supports the communication protocol LU6.2, which can be used for distributed transaction processing as well as for non-transactional communication.

This section provides you with information on the following topics:

- BeanConnect-specific interfaces for outbound communication
- Common Client Interface (CCI) for outbound communication
- Programming information on outbound communication
- Program framework for outbound communication
- Outbound communication with XATMI partners
- Code samples for outbound communication

10.2.1 BeanConnect-specific interfaces for outbound communication

The BeanConnect-specific interfaces for outbound communication are contained in the package `net.fsc.jca.communication`.

10.2.1.1 Connection factory interfaces

You use a connection factory to set up a connection. BeanConnect provides the following connection factory interfaces:

- `EISConnectionFactory`
- `EISOltpConnectionFactory`
- `EISUpicConnectionFactory` (only if the EIS partner is an openUTM application)

The `EISOltpConnectionFactory` interface and the `EISUpicConnectionFactory` interface extend the `EisConnectionFactory` interface without providing additional functionality.

Using the `EISOltpConnectionFactory` or `EISUpicConnectionFactory` interface makes sense if you want to make sure that communication is processed by means of the OSI-TP protocol or the UPIC protocol, respectively.
The use of the `EISConnectionFactory` interface is recommended.

Example 11 Verifying that the OSI-TP protocol is used for communication (openUTM partner)

To verify that communication with the EIS application is processed by means of the OSI TP protocol specify the following code sequence:

```
...
net.fsc.jca.communication.EISConnectionFactory cf =
   (EISConnectionFactory)ic.lookup
      ("java:comp/env/<resourceRefName>");
...
if (! (cf instanceof EISOltpConnectionFactory))
   throw new Exception("EISOltpConnectionFactory was
                       expected!");
...
```

An exception is thrown if not the OSI-TP protocol is used for communication.

10.2.1.2 Connection interfaces (overview)

BeanConnect provides the following connection interfaces:

- `EISConnection`
- `EISOltpConnection`
- `EISUpicConnection` (only if the EIS partner is an openUTM application)

The `EISOltpConnection` interface extends the `EISConnection` interface by providing additional features such as data structure characteristics, for details see "Additional functionality provided by the EISOltpConnection interface" on page 385).

The `EISUpicConnection` interface extends the `EISConnection` interface by providing additional features such as data structure features (for details see Additional functionality provided by the EISUpicConnection interface on page 385).

It is recommended that you use the `EISOltpConnection` and `EISUpicConnection` interfaces only if you use the additional functionality offered by these interfaces.

Communication methods provided by the EISConnection interface

The `EISConnection` interface is extended by a number of interfaces. Each of them provides methods for outbound communication between the EJB and the EIS application. For data exchange, these methods are `snd()`, `rcv()`, `call()` and a number of variants of `snd()` and `rcv()` such as `sndString()` or `rcvRecord()`.

The communication methods offered by these interfaces differ in the data format on which the methods are based:

- For byte-array-oriented access to EIS applications:

  `EISConnectionByteArray`

- For byte-container-oriented access to EIS applications:

  `EISConnectionByteContainer`

  An application may use the `EISConnectionByteContainer` interface and implement the `ByteContainer` interface if it wants to exchange structured objects containing text and binary information with a service in a partner application. Code conversion needs to be performed for the text information of this object within the class that implements the `ByteContainer` interface.

  By providing an appropriate `ByteContainer` object, byte stream to string conversion is handled while executing the `sndRecord()`, `rcvRecord()` or `call()` methods. The objects you can obtain from Cobol2Java are objects that use this feature (see Chapter 14, "Cobol2Java").

- For OLTP message-oriented access to EIS applications:

  `EISConnectionOltpMessage`

  Objects of type `OltpMessage` are exchanged with the EIS application over this interface. The `OltpMessage` object serves as a container for the message content, which is assembled from one or more `OltpMessageRecord` objects and/or one or more `OltpMessagePart` objects. While an `OltpMessageRecord` is of arbitrary length, an `OltpMessagePart` may not exceed 32767 bytes. `OltpMessagePart` objects are mapped to message parts by an openUTM partner application.

  `OltpMessageRecord` objects and `OltpMessagePart` objects accept the following data types:

  - `byte[]`
  - `String`
  - `ByteContainer`

- For string-oriented access to EIS applications:

  `EISConnectionString`

For code conversion in the course of the communication process the `EncodingDef` interface is available.

For detailed information concerning these interfaces, please refer to the JavaDoc of BeanConnect.

Additional functionality provided by the EISOltpConnection interface

The `EISOltpConnection` interface extends the `EISConnection` interface by providing the following additional functionality:

- Methods that are specific for asynchronous communication (see Asynchronous communication on page 391): `setDelayTime()`, `getDelayTime()`
- The method `setEndConversation()`
  The method specifies whether the EIS partner application may or may not terminate the current conversation with the next `send...()` call.

To make use of the additional functionality offered by the `EISOltpConnection` interface you have to cast the `EISConnection` object, which you obtain when you call the object `EISConnectionFactory` with `getConnection()`, as type

```
con = (EISOltpConnection)cf.getConnection(...);
```

Additional functionality provided by the EISUpicConnection interface

The `EISUpicConnection` interface is only relevant if the EIS partner is an openUTM application. It extends the `EISConnection` interface by providing the following additional functionality:

- "Emulation" of terminal functions

  openUTM conversations which were programmed for terminals can also be addressed with the aid of BeanConnect. “Emulation” of terminal functions is only possible with the `EISUpicConnection` interface. Terminal functions such as function keys and cursor positions can be handled using this interface.

- Using format names (format identifiers) when sending or receiving data

  When data is exchanged between the resource adapter and the openUTM partner application, it is possible to also send format names (format identifiers). Thus the resource adapter can send to or receive from the openUTM partner structure information about the data along with the user data itself. This function is only supported if the requested service uses the KDCS interface.

- Restart functionality

  The openUTM partner application performs an automatic service restart for user IDs that have been defined with the `USER` control statement in conjunction with the `RESTART=YES` operand.

- The method `isInTransaction()` which is used to query the transaction status at the EIS partner.

To make use of the additional functionality offered by the `EISUpicConnection` interface, you have to cast the `EISConnection` object (which you get calling the `getConnection()` method on the `EISConnectionFactory` object) to the type `EISUpicConnection`:

```
con = (EISUpicConnection)cf.getConnection(...);
```

10.2.1.3 Communication using the connection interfaces

The connection interfaces provide a variety of features which may be used by an EJB in a communication with an EIS application. These may be used as alternatives or in combination:

- Dialog communication (based on the `EISConnection`, `EISOltpConnection` and `EISUpicConnection` interfaces)
- Asynchronous communication (based on the `EISConnection` interface and on additional methods of the interface `EISOltpConnection`)
- Transactional communication
- Connection groups
- Code conversion

The following sections provide information concerning these topics. Examples of the most common communication scenarios are given in the JavaDoc of BeanConnect.

Dialog communication

The recommended way for an EJB to communicate with an EIS application is by using the methods of the `EISConnectionOltpMessage` interface. Objects of type `OltpMessage` are exchanged with the EIS application over this interface. The `OltpMessage` object serves as a container for the message content, which is assembled from one or more `OltpMessageRecord` objects and/or one or more `OltpMessagePart` objects. While an `OltpMessageRecord` is of arbitrary length, the length of an `OltpMessagePart` may not exceed 32767 bytes.

At the time of a `sndOltpMessage()` call, the data to be sent is merely handed over to BeanConnect but not yet transferred to the EIS application. The request to the EIS application is only actually sent when the first `rcvOltpMessage()` method is issued for this connection.

The easiest way to call a dialog service in an EIS application is with the `call()` method. This method allows RPC-like communication with an EIS application.

The following sections outline the communication between an EJB and an openUTM/CICS program by means of `OltpMessage` objects.

The options listed below are available for exchanging data between an EJB and an openUTM/CICS program using `OltpMessage` objects:

- Data exchange based on OltpMessagePart objects with openUTM partners
- Data exchange based on OltpMessageRecord objects

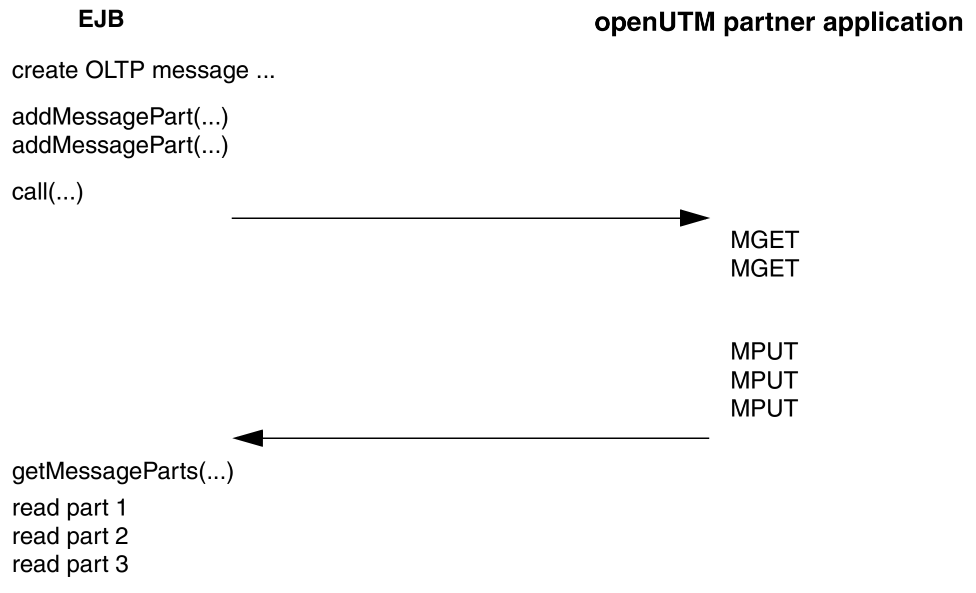
You will find further examples of the most common communication scenarios as well as code examples in the JavaDoc of BeanConnect.

Data exchange based on OltpMessagePart objects with openUTM partners

With `MGET` and `MPUT`, an openUTM program can receive or send one or more message parts each with a maximum length of 32 Kbytes. Here the individual `MessagePart` objects correspond directly to the `MGET` and `MPUT` calls in the openUTM program.

Data is thus exchanged according to the following scheme:

Figure 61: Data exchange based on OltpMessagePart objects (openUTM partners)

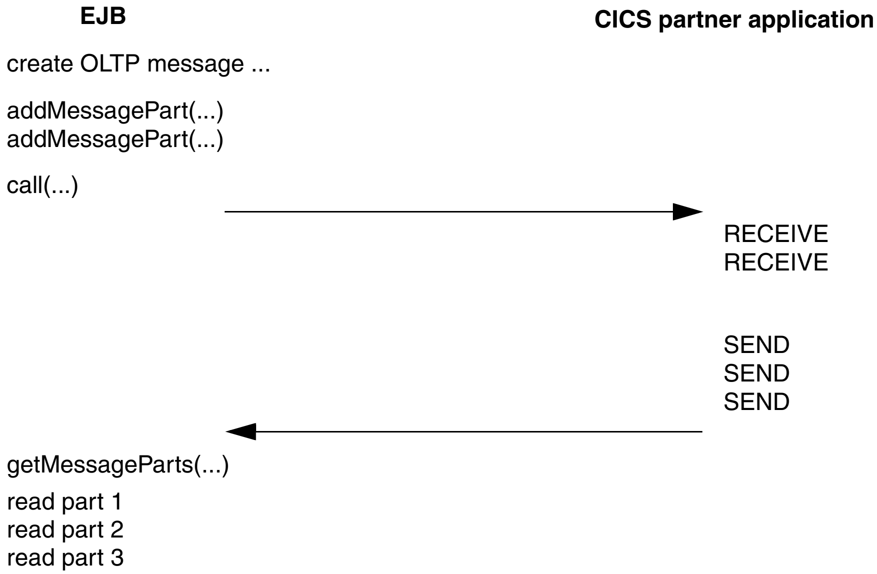


Data exchange based on OltpMessagePart objects with CICS partners

With `RECEIVE` and `SEND`, a CICS program can receive or send one or more message parts each with a maximum length of 32 Kbytes. Here the individual `MessagePart` objects correspond directly to the `RECEIVE` and `SEND` calls in the CICS program.

Data is thus exchanged according to the following scheme:

Figure 62: Data exchange based on OltpMessagePart objects (CICS partners)



Data exchange based on OltpMessageRecord objects

As an alternative to multiple `MessagePart` objects sent within an `OltpMessage` object, you can use `OltpMessageRecord` objects sent within an `OltpMessage` object. In this case, you can send and receive messages which are longer than 32 Kbytes without needing to break it down into packets of 32 Kbytes as a Java programmer.

Figure 63: Data exchange based on OltpMessageRecord objects (OSI-TP protocol with openUTM partners)

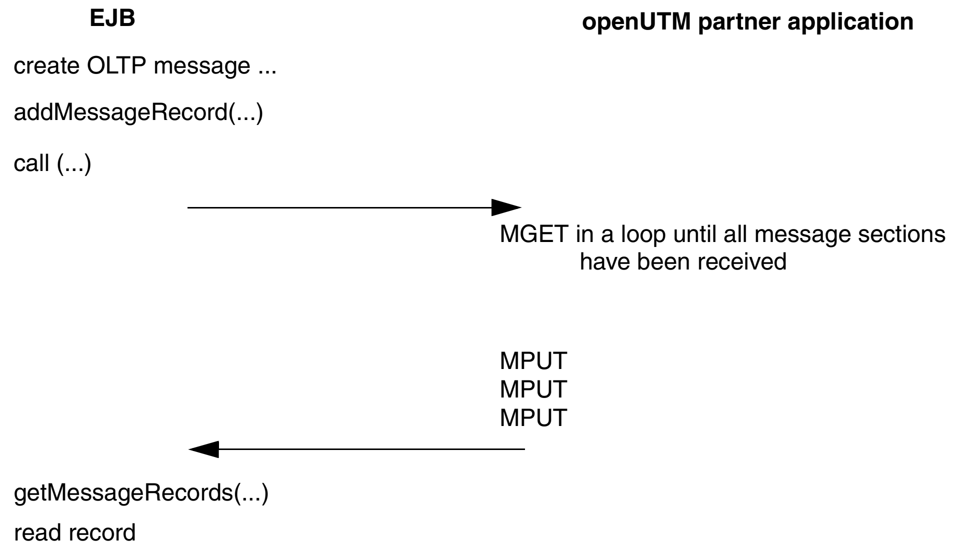
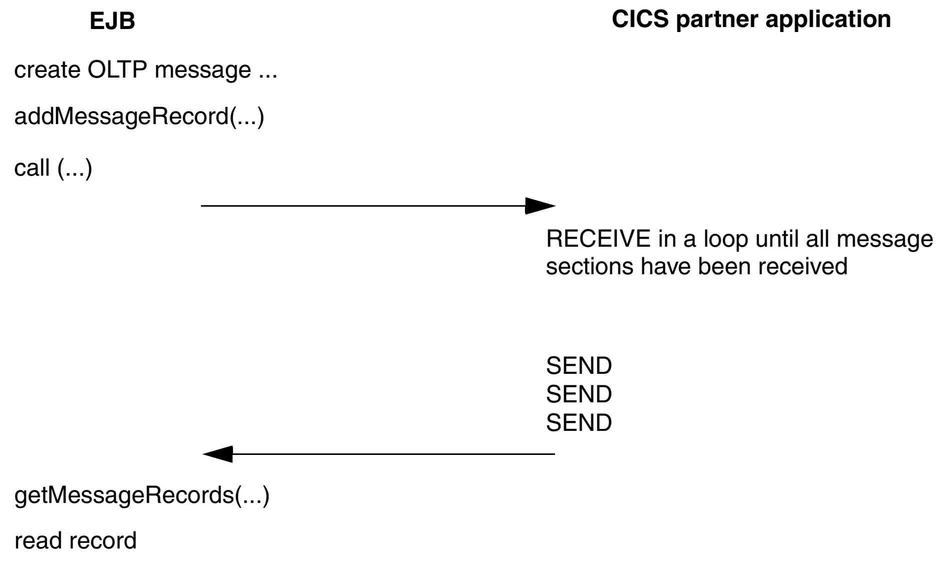
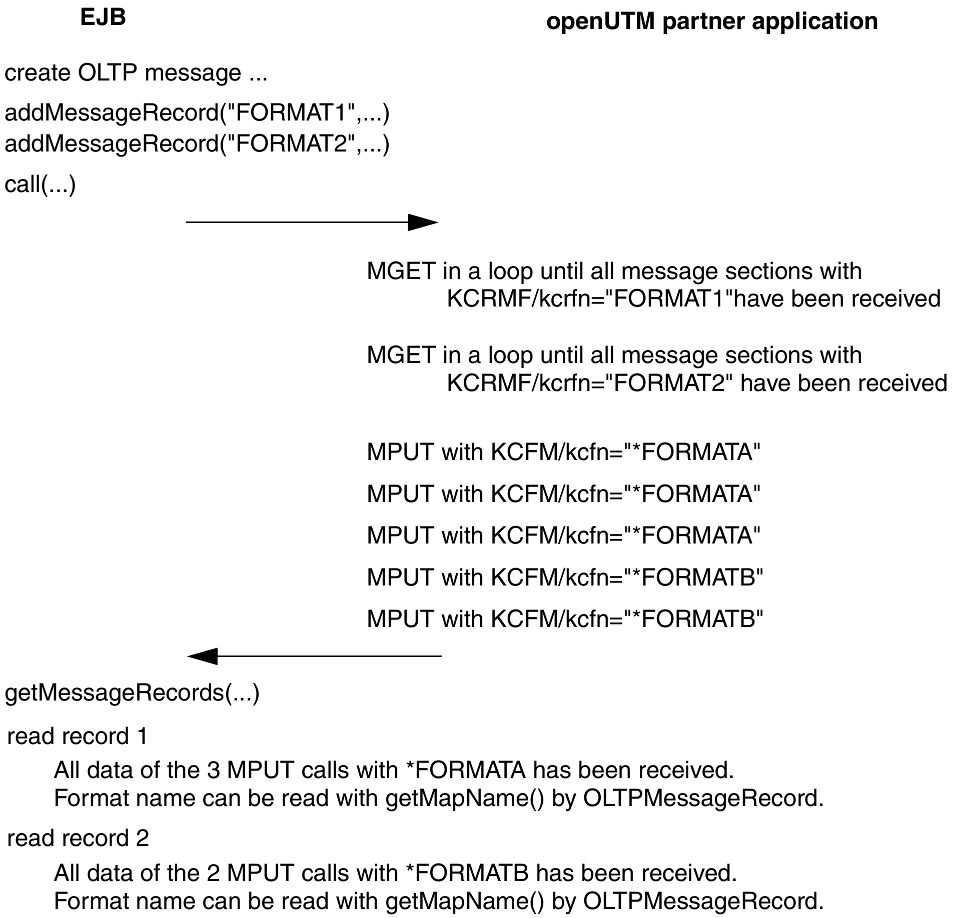


Figure 64: Data exchange based on OltpMessageRecord objects (CICS partners)



When communicating with an openUTM application via the UPIC protocol, you can use `OltpMessageRecord` objects and `OltpMessagePart` objects if you are working with different format names.

Figure 65: Data exchange based on OltpMessageRecord objects (UPIC protocol)



EJB

openUTM partner application

create OLTP message ...

addMessageRecord("FORMAT1",...)
addMessageRecord("FORMAT2",...)

call(...)

MGET in a loop until all message sections with
KCRMF/kcrfn="FORMAT1"have been received

MGET in a loop until all message sections with
KCRMF/kcrfn="FORMAT2" have been received

MPUT with KCFM/kcfn="*FORMATA"

MPUT with KCFM/kcfn="*FORMATA"

MPUT with KCFM/kcfn="*FORMATA"

MPUT with KCFM/kcfn="*FORMATB"

MPUT with KCFM/kcfn="*FORMATB"

getMessageRecords(...)

read record 1

All data of the 3 MPUT calls with *FORMATA has been received.
Format name can be read with getMapName() by OLTPMessageRecord.

read record 2

All data of the 2 MPUT calls with *FORMATB has been received.
Format name can be read with getMapName() by OLTPMessageRecord.

Asynchronous communication

A service is called asynchronous when it does not send a reply message.

The sending of an asynchronous message has to be explicitly triggered by the EJB with the methods `snd()` + `flush()`, `sndLast()` or `sndLastString()` of the `EISConnection` interface, which finalize the message and initiate the send cycle.

An asynchronous message may be sent to the EIS partner either immediately or after a delay. With a delayed sending, the EJB must specify the delay time prior to calling the `snd()` method. You specify the delay time by means of the `setDelayTime()` method. A delayed message is stored by the BeanConnect proxy until the time delay has elapsed. The message is then forwarded to the EIS application.

i An asynchronous message generated in a transaction is sent only if and when the transaction is committed. If the transaction is rolled back, the asynchronous request will not be sent. An asynchronous request with time delay 0 generated outside of a transaction is sent immediately.

If you want to ensure that an asynchronous service is currently being addressed, you can check this using the `getPartnerType()` method or you can use the `setDelayTime(0)` method, which throws an exception if a dialog service is addressed unexpectedly. However, the use of this method results in certain performance impairments.

Examples of asynchronous communication with an EIS partner can be found in the JavaDoc of BeanConnect.

Transactional communication

During deployment of a connection factory, the configuration property `transactional` of the connection factory has to be set (see "transactional" on page 108). A connection factory may either support both transactional and non-transactional communication or it may be limited to non-transactional communication only:

- If the connection factory supports transactional communication, the runtime environment decides at the time a transaction is started whether the communication within a conversation of such a connection will be transactional or not. A conversation which is started while a transaction is open is always included in the transaction.

  A conversation which was already open at the start of the transaction but on which no communication has yet occurred is also included in the transaction. In these cases a transactional protocol will be used for communication; otherwise a non-transactional protocol will be used.

- If the connection factory does not support transactional communication, the following applies: For connections generated with this connection factory, a non-transactional protocol is always used, no matter whether the communication takes place within or outside an application server transaction. This allows the deployer of the resource adapter to explicitly exclude an EIS service from a transaction that may be active in the application server.

You can query the current status of a transaction using the `isInDistributedTransaction()` method of the `EISOltpConnection` interface.

When a transactional protocol is used for communication, the application server transaction and the transaction in the EIS partner are part of a single, distributed transaction which are either committed or rolled back as a single unit. With transactional communication, a conversation must not span more than one transaction. However, a conversation may comprise more than one dialog step. This means that several `snd()/rcv()` pairs may be exchanged with the EIS partner within a single transaction.

If an EJB uses more than one transactional connection within a transactional EJB method, it is strongly recommended that these connections are associated in a single `EISConnectionGroup` (see the following section "Connection groups"). Doing so helps to avoid a resource bottleneck.

Connection groups

The concept of associated connections enables an EJB to simultaneously send several messages via more than one connection. This allows the processing of several dialog services of one or more EIS partner application at the same time, thereby avoiding the time delay caused by the sequential processing of a number of RPC-like calls.

You can associate a connection with another connection by means of an `EISConnectionGroup`. You create the `EISConnectionGroup` using an `EISConnectionGroupFactory`. You get an `EISConnectionGroupFactory` with the method `getEISConnectionGroupFactory()`from a `ConnectionFactory`. For example:

```
ConnectionFactory cf1 = (ConnectionFactory)ic.lookup(...);
ConnectionFactory cf2 = (ConnectionFactory)ic.lookup(...);
EISConnectionGroupFactory cgf =
                        cf1.getEISConnectionGroupFactory();
EISConnectionGroup cg = cgf.getConnectionGroup();
con1 = (EISOltpConnection)cf1.getConnection(cg);
con2 = (EISOltpConnection)cf2.getConnection(cg);
```

You will find a code sample for Example 15, "Associated connections using the EISConnectionGroup interface" on page 408.

The simultaneous dispatch of the messages to all connections of the connection group is initiated by calling the `execute()` method on the `EISConnectionGroup`-object. Connections for transactional communication may be associated with connections for non-transactional communication. For details see the JavaDoc of BeanConnect.

Code conversion

Whenever `String` or `ByteContainer` objects are used, BeanConnect can perform a code conversion. The methods for setting up the proper environment for a code conversion are contained in the interface `net.fsc.beanta.encoding.EncodingDef`. This interface is extended by the `EISConnection` interfaces of BeanConnect. If code conversion has not already been activated at deployment time using the `encodingActive` configuration property (see "encodingActive" on page 106), you can switch on code conversion by calling the method `setEncodingActive()`. The code table to be used for a code conversion is assigned by defining the `encoding` configuration property (see "encoding" on page 105) at deployment time or by using the method `setEncoding()` of the `EISConnection` interface. For details about code conversion of messages and on how to supply your own encoding tables see Chapter 11, "Encoding and national language support" on page 429) and the JavaDoc of package `net.fsc.beanta.encoding`.

10.2.2 Common Client Interface (CCI) for outbound communication

The CCI interfaces for outbound communication are contained in the packages `javax.resource.cci` and `net.fsc.jca.communication.cci`. For outbound communication, the CCI offers almost the same functionality as the BeanConnect-specific interfaces (with the exception of the additional functionality with openUTM partners offered by the `EISUpicConnection` interface). You find information on the program framework using the CCI interface in Section 10.2.4.2, "Program framework for Common Client Interface (CCI)".

Connection factory interfaces

On deployment of the resource adapter, you can specify that you want to use outbound communication via CCI. To do this, you specify one of the following connection factory interfaces (see "Selecting the interfaces to be used" on page 381) in the `<connection-factory-interface>` element of the deployment descriptor file `weblogic-ra.xml`:

- `net.fsc.jca.communication.cci.BCOltpConnectionFactory`
- `net.fsc.jca.communication.cci.BCUpicConnectionFactory`
  (only if the EIS partner is an openUTM application)

In your EJB code, you use a connection factory to set up a connection. CCI provides the following connection factory interfaces:

- `javax.resource.cci.ConnectionFactory`
- `net.fsc.jca.communication.cci.BCOltpConnectionFactory`
- `net.fsc.jca.communication.cci.BCUpicConnectionFactory`
  (only if the EIS partner is an openUTM application)

The `BCOltpConnectionFactory` interface and the `BCUpicConnectionFactory` interface extend the `javax.resource.cci.ConnectionFactory` interface without providing additional functionality.

Using the `BCOltpConnectionFactory` or `BCUpicConnectionFactory` interface makes sense if you want to verify that communication is processed via the OSI-TP or UPIC protocol respectively. Example 11 on page 383 applies analogously. It is recommended that you use the interface `javax.resource.cci.ConnectionFactory`.

Connection interface

The CCI interface for the outbound communication is `javax.resource.cci.Connection`.

10.2.3 Programming information on outbound communication

This section provides programming information on outbound communication between an EJB and an EIS partner application.

10.2.3.1 Addressing an EIS application

A service of an EIS application which is used by your Java EE application has to be configured during deployment by defining the configuration property `connectionURL` (see `connectionURL` on page 104 and on page 120). If the Java EE application utilizes multiple services of the same EIS partner in an EIS you may use the `setServiceName()` method of the connection object to address a specific service explicitly:

```
connection.setServiceName(<service_name>);
```

10.2.3.2 Placing BeanConnect calls in an EJB

In the JNDI service of the application server, you have to execute a `lookup()` for a preconfigured connection factory. A connection factory provides a `getConnection()` method that returns a `Connection` object. During the deployment of the connection factory the properties of this connection (EIS address, EIS service name, etc.) were configured. No matter which type of connection factory has been deployed (`EISUpicConnectionFactory` or `EISOltpConnectionFactory`), a `Connection` object that implements the `EISConnection` interface is always returned. When the application no longer needs a `Connection` object, it must return it to the application server for pooling or destruction by issuing the `close()` method for the `Connection` object.

The application must make sure that it also releases the requested connections with `close()` when errors occur. Otherwise, secondary errors may occur in the application server.

It is recommended that the JNDI `lookup()` for a connection factory is executed during initialization. In an EJB this can, for example, take place within the `ejbCreate()` or `setSessionContext()` method.

The `getConnection()` method and the associated `close()` method must be called directly within the business methods.

10.2.3.3 Authentication (user ID and password)

Authentication takes place by means of user name and password. A distinction is made between

- Container-managed authentication
- Application-managed authentication

It is recommended that you use container-managed authentication.

Container-managed authentication

In the case of container-managed authentication, the access data is handled by the container. The EJB deployer configures the container-managed authentication with the following entry in the EJB deployment descriptor:

```
<res-auth>Container</res-auth>
```

When using container-managed authentication, you call the `getConnection()` method without parameters.

With container-managed authentication, the authentication data itself must be specified in the deployment descriptor for the BeanConnect resource adapter within a `<security-config>` section in the application server-specific deployment file (`weblogic-ra.xml` for the Oracle WebLogic Server).

Application-managed authentication

In the case of application-managed authentication, the access data is handled in the program code of the EJB. The EJB deployer configures the application-managed authentication in the EJB deployment descriptor with the following entry:

```
<res-auth>Application</res-auth>
```

In the EJB source code, you use, for example, the following code sequence instead of the `getConnection()` call without parameters:

```
javax.naming.InitialContext ic = new InitialContext();

String user = (String)ic.lookup("java:comp/env/User");
String password = (String)ic.lookup("java:comp/env/Password");
net.fsc.jca.communication.PasswordCredential pwc =
  new net.fsc.jca.communication.PasswordCredential
    (user, password);
con= (net.fsc.jca.communication.EISConnection)cf.getConnection(pwc);
```

Here, the user ID (user in the example) and the password (password in the example) are defined as environment variables of the EJB. The deployer can adapt environment variables as required. The environment variables can be accessed using the `lookup()` method.

10.2.3.4 Querying information on the conversation with the EIS application

The `isInConversation()` method of the connection object enables you to query the status of the EIS application.

Example 12 Information on the conversation with the EIS application

You want to assure that the conversation with the EIS application will be terminated after method execution has been completed:

```
...
String s = con.call("what will be the echo of this");

if (eis.isInConversation())
{
 con.terminate();
 con.close();
 throw new EJBException
   ("EJB Exception: EIS Partner Service not yet terminated ... ");
}
```

10.2.3.5 Programming hints with respect to CICS applications

CICS transactions have to be designed and coded to comply with the Distributed Transaction Programming (DTP) paradigm. For a description of this programming paradigm, see the IBM CICS documentation, for example the "CICS Distributed Transaction Programming Guide".

The following restrictions and rules have to be considered to allow outbound communication by means of BeanConnect:

- A CICS partner application may never use `SYNCPOINT` or `ISSUE PREPARE` itself. Instead, it may only do so when requested to by the EJB in the Java EE application server.
- Basic conversation is not possible. Basic conversation is programmed for CICS by using commands that begin with GDS, such as `EXEC CICS GDS ALLOCATE`, for example.
- BeanConnect always establishes LU6.2 conversations with SYNCLEVEL 0 or 2 and never with SYNCLEVEL 1. The SYNCLEVEL of an incoming conversation can be queried in CICS-API using `EXTRACT PROCESS`.
- The default value of the communication property `endConversation` is `false` for CICS partners and `true` for openUTM partners.

10.2.3.6 Support of DPL (Distributed Program Link) programs

CICS provides different programming interfaces to invoke another CICS program or to allow a program to be invoked by another CICS program. Two of these facilities are important here.

- DTP (Distributed Transaction Processing) which enables a CICS transaction to communicate with a CICS application running in another system by exchanging messages. DTP programs are coded using the APPC programming interface.
- DPL (Distributed Program Link) which enables a CICS program to invoke a program in another CICS system and wait for the called program to return. Data is exchanged between the programs in a communication area (`COMMAREA`). DPL is similar to a remote procedure call.

BeanConnect makes it possible to invoke outbound transactions via DTP. However, it is not possible to call a DPL program directly. For this, a DTP program is needed in which the program link is wrapped.

An example of such a COBOL program fragment with the name `DPLSERVR.CCP` can be found in the directory `<BC_home>/<proxy_cont_name>/src` (Solaris/Linux) respectively `<BC_home>\<proxy_cont_name>\src` (Windows). This source code contains hints on the changes which must be carried out to build a new program.

The DPL program is called with the CICS `LINK` command, which has three important parameters:

- `PROGRAM` to specify the name of the program to which control is to be passed unconditionally
- `COMMAREA` to specify the communication area that is made available to the linked program
- `LENGTH` which specifies the length in bytes of the communication area.

The input data for the distributed program call is received in a message. The input data must be copied to the communication area. The required output data has to be sent as response to the EJB when the linked program returns.

10.2.4 Program framework for outbound communication

This section provides a program framework for outbound communication between an EJB and an EIS application. The framework contains the principal communication steps.

10.2.4.1 Program framework for BeanConnect-specific interfaces

In BeanConnect you specify the EIS partner to be addressed when deploying a managed connection factory. You use the `lookup()` method to search for the connection factory and obtain a connectivity object by calling the `getConnection()` method.

The connectivity object provided implements the `EISConnection` interface for communication with the EIS application:

1. Set up the initial context:

   ```
   javax.naming.InitialContext ic = new InitialContext();
   ```

2. Reference a connection factory:

   ```
   net.fsc.jca.communication.EISConnectionFactory cf =
     (EISConnectionFactory)ic.lookup
       ("java:comp/env/<resource_reference_name>");
   ```

3. Set up the connection:

   ```
   net.fsc.jca.communication.EISConnection con = (EISConnection)
   cf.getConnection();
   ```

4. If you use the `EISConnection` interface in order to specify the service name (TAC) or the name of the EIS service, proceed as follows:

   ```
   con.setServiceName(<name_of_the_service>);
   ```

   However, you should note that for performance reasons, it is recommended that you work with the preconfigured service names.

5. Create the message that you want to send

   ```
   String requestMessage = "...";
   ```

6. Call the EIS application and receive the reply message:

   ```
   String replyMessage = con.call(requestMessage);
   ```

7. Close the connection:

   ```
   con.close();
   ```

Further information on how to program the `EISConnection` and `EISOltpConnection` interfaces is provided in the JavaDoc of the interface itself.

You will find a sample in Example 13, "Dialog communication using the EISConnection interface" on page 406.

10.2.4.2 Program framework for Common Client Interface (CCI)

When deploying a managed connection factory in BeanConnect you specify the EIS partner to be addressed via this managed connection factory. You use the `lookup()` method to search for the connection factory and obtain a connectivity object by calling the `getConnection()` method.

You can request an interaction object via the CCI connection which you obtain from the CCI connection factory. This interaction object implements an `execute()` method for initiating an interaction. The `execute()` method also knows a `BCCciInteractionSpec` object in addition to the input record and output record. In this `BCCciInteractionSpec` you define an `interactionVerb` (`SYNC_SEND`, `SYNC_SEND_RECEIVE` or `SYNC_RECEIVE`) and, at the same time, the name of the EIS application. This enables you to control an interaction by providing suitable data instead of calling methods. The default value of the interactionVerb is `SYNC_SEND_RECEIVE`.
For details, see the JavaDoc of BeanConnect.

Program framework for dialog communication (CCI)

For dialog communication with your server application over the CCI Interface in BeanConnect, proceed as follows:

1. Set up the initial context:

   ```
   javax.naming.InitialContext ic = new InitialContext();
   ```

2. Reference a connection factory:

   ```
   javax.resource.cci.ConnectionFactory cf =
     (ConnectionFactory)ic.lookup("java:comp/env/eis/myEIS");
   ```

   A dialog service is assigned as the default service to the connection factory referenced by `eis/myEIS`.

3. Set up the connection:

   ```
   javax.resource.cci.Connection con =
      (Connection)cf.getConnection();
   ```

   Alternatively, an EJB may pass security-related information (user ID/password) to BeanConnect in a `BCCciConnectionSpec` object. In this case you specify:

   ```
   net.fsc.jca.communication.cci.BCCciConnectionSpec;
   cred = new BCCciConnectionSpec("myuser", "mypass");
   ```

   ```
   javax.resource.cci.Connection con =
      (Connection)cf.getConnection(cred);
   ```

4. Create an `Interaction` object and an `InteractionSpec` object:

   ```
   Interaction ix = (Interaction)con.createInteraction();
   BCCciInteractionSpec is = new
      BCCciInteractionSpec(InteractionSpec.SYNC_SEND_RECEIVE);
   ```

   Whereas an `Interaction` object is created from the `Connection` object on which it is to be used, the `InteractionSpec` object is created using a constructor of the implementation class.

   An `Interaction` object enables an EJB to communicate with an EIS application. An `InteractionSpec` object holds properties for driving an interaction with this EIS application. It is used by an interaction to execute the specified function in the EIS application.

5. Create a `BCRecord` object that serves as a container for the message to the EIS and another `BCRecord` object that serves as a container for the reply message. This is done by means of a `BCRecordFactory` object:

   ```
   net.fsc.jca.communication.cci.BCRecordFactory rf =
      (BCRecordFactory)cf.getRecordFactory();
   net.fsc.jca.communication.cci.BCRecord reqrec = (BCRecord)
   rf.createBCRecord("request");
   net.fsc.jca.communication.cci.BCRecord replrec = (BCRecord)
   rf.createBCRecord("reply");
   ```

   After creation, a `BCRecord` object contains an empty `OltpMessage` object, which can be retrieved from the `BCRecord` object. Subsequently, the `BCRecord` object can be populated with `OltpMessageRecord` or `OltpMessagePart` objects.

6. Populate the `OltpMessage` object of the message which is destined for the EIS with data (here: with two `OltpMessagePart` objects):

   ```
   net.fsc.jca.communication.OltpMessage request = reqrec.getOltp-
   Message();
   request.addMessagePart("request - message part1");
   request.addMessagePart("request - message part2");
   ```

7. Execute the interaction:

   ```
   ix.execute(is, reqrec, replrec);
   ```

   The `BCRecord` object returned from this call again holds an `OltpMessage` object, which in turn contains the reply sent by the EIS application in an `OltpMessageRecord` or `OltpMessagePart` object.

8. Receive the reply message:

   ```
   net.fsc.jca.communication.OltpMessage reply =
   replrec.getOltpMessage();
   java.util.Iterator<net.fsc.jca.communication.OltpMessagePart> it =
   reply.getMessageParts();

   net.fsc.jca.communication.OltpMessagePart msgPart;
   String msgText="";

   while (it.hasNext()) {
      msgPart = (OltpMessagePart) it.next();
      msgText += msgPart.getText();
   }
   ```

9. Close the connection:

   ```
   con.close();
   ```

Further information on how to program the CCI interfaces is provided in the JavaDoc of the CCI.

You will find a code sample in Example 14, "Dialog communication using the CCI" on page 407.

Program framework for asynchronous communication (CCI)

For asynchronous communication with your server application over the CCI Interface in BeanConnect, proceed as follows:

1. Set up the initial context:

   ```
   javax.naming.Context ic = new InitialContext();
   ```

2. Reference a ConnectionFactory:

   ```
   javax.resource.cci.ConnectionFactory cf =
   (ConnectionFactory)ic.lookup("java:comp/env/eis/myAsyncEIS");
   ```

3. Set up the connection:

   ```
   javax.resource.cci.Connection con =
      (Connection)cf.getConnection();
   ```

   Alternatively, an EJB may pass security-related information (user ID/password) to BeanConnect in a `BCCciConnectionSpec` object. In this case you specify:

   ```
   net.fsc.jca.communication.cci.BCCciConnectionSpec cred =
      new BCCciConnectionSpec("myuser", "mypass");
   ```

   ```
    javax.resource.cci.Connection con =
      (Connection)cf.getConnection(cred);
   ```

4. Create an `Interaction` object and an `InteractionSpec` object:

   An `Interaction` object enables an EJB to communicate with a partner application. An `InteractionSpec` object holds properties for driving an interaction with a partner application. It is used by an interaction to execute the specified function in the EIS application.

   Whereas an `Interaction` object is created from the `connection` object on which it is to be used, the `InteractionSpec` is created using a constructor of the implementation class:

   ```
   Interaction ix = (Interaction)con.createInteraction();
   net.fsc.jca.communication.cci.BCCciInteractionSpec is = new
      BCCciInteractionSpec(InteractionSpec.SYNC_SEND,
                                                     "ASYNTAC");
   ```

   The assignment of the asynchronous service `ASYNTAC` specifies that communication will be asynchronous.

   Here, the `SYNC_SEND` has the same effect as a `flush()` call when using the BeanConnect-specific interfaces.

5. Create a `BCRecord` object that serves as a container for the request message. This is done by means of a `recordFactory`:

   ```
   net.fsc.jca.communication.cci.BCRecordFactory rf=
      (BCRecordFactory)cf.getRecordFactory();
   net.fsc.jca.communication.cci.BCRecord reqrec =
      rf.createBCRecord("request");
   ```

   After creation, a `BCRecord` object contains an empty `OltpMessage` object, which can be retrieved from the `BCRecord` object and then can be assigned with `OltpMessageRecord` and/or `OltpMessagePart` objects.

6. Populate the `OltpMessage` object of the request record with data (here: with two `OltpMessagePart` objects):

   ```
   net.fsc.jca.communication.OltpMessage request = reqrec.getOltp-
   Message();
   request.addMessagePart("request - message part1");
   request.addMessagePart("request - message part2");
   ```

7. Execute the interaction:

   ```
   ix.execute(is, reqrec);
   ```

8. Close the connection:

   ```
   con.close();
   ```

10.2.5 Outbound communication with XATMI partners

You need to take account of the following special characteristics if you program outbound communication with XATMI partners:

- Transaction management

  The transaction environment in the user program and the transactional property of the `ConnectionFactory` determines whether communication with the partner application is performed with or without Commit FU (TRAN or NOTRAN in XATMI).

- typed buffer

  Only typed buffers of type X_OCTET are supported.

- Message length and message segments

  When the OLTP message interface is used, a message segment in a message sent to an XATMI partner may be a maximum of 32,000 bytes in length. Longer message segments are rejected by means of an `OltpMessageException`. The same restriction to 32,000 bytes per message segment applies to all the EIS connection interface methods for which the restriction for other partners is 32,767 bytes.

  Messages to XATMI partners in request/reply mode may only consist of one message segment and this may be a maximum of 32,000 bytes in length.

  Messages to XATMI partners in conversational mode may consist of more than one message segment. Each of these message segments may be up to 32,000 bytes in length.

  Messages passed to BeanConnect via interface methods for which there is no message length restriction, e.g. `sndRecord(String)` are fragmented by BeanConnect into message segments of a maximum of 32,000 bytes in length during communication with XATMI partners in conversational mode. In the case of XATMI partners in request/reply mode, messages with a length of more than 32,000 bytes are rejected with an exception.

- The reception of a FAILURE_RI is indicated to an application by means of an `EISConnectionException`.

- For XATMI partners, the default value of the communication property `endConversation` is always `false`.

10.2.6 Code samples for outbound communication

This section contains the following code samples:

- Dialog communication using the EISConnection interface on page 406.
- Dialog communication using the CCI on page 407.
- Associated connections using the EISConnectionGroup interface on page 408.

Example 13 Dialog communication using the EISConnection interface

```
...
public String callService(String request) throws EJBException
{
  net.fsc.jca.communication.EISConnectionFactory cf = null;
  net.fsc.jca.communication.EISConnection con = null;
  String reply = null;

  try
  {
   javax.naming.InitialContext ic =
     new javax.naming.InitialContext();
   cf = (net.fsc.jca.communication.EISConnectionFactory)
     ic.lookup("java:comp/env/eis/myEIS");
   con= cf.getConnection();

   reply = con.call(request);

   con.close();
   con = null;

   return reply;
  }
  catch (Exception e)
  {
   if (con != null)
   {
    con.close();
   }
   throw new EJBException ("EJB Exception: " + e);
  }
}
```

To allow you to use this method, the deployment descriptor of the EJB must contain the following information:

```
<resource-ref>
 <res-ref-name>eis/myEIS</res-ref-name>
  <res-type>net.fsc.jca.communication.EISConnectionFactory</
    res-type>
  <res-auth>Container</res-auth>
  <res-sharing-scope>Unshareable</res-sharing-scope>
</resource-ref>
```

Example 14 Dialog communication using the CCI

```
...
public String sndRcvJavax(String user, String name, String data)
{

  String retValue = "";
  Connection connection = null;

  try {
    Context ic = new InitialContext();
    ConnectionFactory cf =
      (ConnectionFactory)ic.lookup("java:comp/env/eis/myEIS");

    ConnectionSpec cred = new BCCciConnectionSpec (user, "");
    connection = cf.getConnection(cred);
    Interaction ix = connection.createInteraction();

    InteractionSpec is =
      new BCCciInteractionSpec(
                   InteractionSpec.SYNC_SEND_RECEIVE,"HELLO");
    BCRecordFactory recordFactory =
                 (BCRecordFactory)cf.getRecordFactory();
    BCRecord in = recordFactory.createBCRecord("SendRecord");
    BCRecord out =
              recordFactory.createBCRecord("ReceiveRecord");
    OltpMessage inMsg = in.getOltpMessage();
    inMsg.addMessagePart(data);

    out = (BCRecord)ix.execute(is, in);

    OltpMessage outMsg = out.getOltpMessage();
    Iterator it<OltpMessagePart> = outMsg.getMessageParts();
    OltpMessagePart  msgPart;
    while (it.hasNext()) {
      msgPart = it.next();
      retValue  += msgPart.getText();
    }

  } catch ( Throwable  ex) {
  // Todo: Fehlerbehandlung
  }// tryCatch
  try    {
    if ( connection != null )
        connection.close();  }
  catch (ResourceException e)    {
    retValue += "\n bei connection.close():\n"+getStackInfo(e);
  }
  return retValue;
} // sndRcvJavax
```

To allow you to use this method, the deployment descriptor of the EJB must contain the following information:

```
<resource-ref>
 <res-ref-name>eis/myEIS</res-ref-name>
  <res-type>net.fsc.jca.communication.EISConnectionFactory</res-type>
  <res-auth>Container</res-auth>
  <res-sharing-scope>Shareable</res-sharing-scope>
</resource-ref>
```

Example 15   Associated connections using the EISConnectionGroup interface

```
...
EISOltpConnection con1 = null;
EISOltpConnection con2 = null;
  try {
     javax.naming.InitialContext ic =
     new javax.naming.InitialContext();
     cf = (net.fsc.jca.communication.EISConnectionFactory)
     ic.lookup("java:comp/env/eis/myEIS");
     EISConnectionGroupFactory cgf =
                       cf.getEISConnectionGroupFactory();
     EISConnectionGroup cg = cgf.getConnectionGroup();
     con1 = (EISOltpConnection)cf.getConnection(cg, new
                           PasswordCredential("upicusea", ""));
     con2 = (EISOltpConnection)cf.getConnection(cg, new
                           PasswordCredential("upicuseb", ""));
```

```
        OltpMessage om1 = con1.createMessage();
        om1.addMessagePart("STAT");
        OltpMessage om2 = con2.createMessage();
        om2.addMessagePart("osi-con,l=kdcall");
        con1.sndOltpMessage(om1);
        con2.sndOltpMessage(om2);
        cg.execute();
        String s;
        om2 = con2.rcvOltpMessage();
        Iterator iter<OltpMessageRecord> = om2.getMessageRecords();
        for (s = ""; iter.hasNext(); )
            { s+= iter.next()).getText(); }
        String result_o = "";
        result_o = result_o + "OltpConnection: KDCINF osi-con,
                   l=kdcall\n" + s + "\n";
        om1 = con1.rcvOltpMessage();
        iter = om1.getMessageRecords();
        for (s = ""; iter.hasNext(); ) {
             s+= ((OltpMessageRecord)iter.next()).getText(); }
        result_o = addResult_o(result_o, "OltpConnection:
                       KDCINF STAT\n" + s + "\n");
        s = cg.getGroupName();
        result_o = result_o + "OltpConnection: KDCINF STAT\n" +s
                   + "\n");
        con1.close();
        con2.close();
        return result_o;
    } catch(EISConnectionException ex) {
    ...
```

10.3 Programming inbound communication

For inbound communication, an EIS application addresses an OLTP message-driven bean which is deployed in the application server. Communication between the EIS application and the OLTP message-driven bean requires that the OLTP message-driven bean has been made known to BeanConnect using the Management Console.

This section provides you with information on the following topics:

- OLTP message-driven beans
- Inbound communication with openUTM partners
- Inbound communication with CICS applications
- Inbound communication with other EIS partners (openUTM)
- Inbound communication with XATMI partners
- BeanConnect-specific interfaces for inbound communication
- Common Client Interface (CCI) for inbound communication
- Code samples for inbound communication

10.3.1 OLTP message-driven beans

OLTP message-driven beans are the types of JCA-like message endpoint applications supported by BeanConnect. An EIS application can call OLTP message-driven beans deployed in an application server via BeanConnect.

In order to communicate with an OLTP message-driven bean, an EIS application sends a message to a service known to BeanConnect. BeanConnect then passes the message to an OLTP message-driven bean configured for the message endpoint name that is associated with this service name. The connection between the service name and the message endpoint name is established by the Management Console (see Section 6.8.1, "Configuring inbound message endpoints").

BeanConnect supports two types of OLTP message-driven beans:

- OLTP message-driven beans for dialog communication
- OLTP message-driven beans for asynchronous communication

OLTP message-driven beans for dialog communication

An OLTP message-driven bean for dialog communication receives messages from an EIS application and returns messages. The related BeanConnect-specific interface is `net.fsc.jca.communication.OltpMessageListener`. The CCI interface `javax.resource.cci.MessageListener` also matches the requirements for dialog communication.

These interfaces enable an OLTP message-driven bean to receive a message from and send a reply message to the EIS application. The messages may contain one or more message parts.

OLTP message-driven beans for asynchronous communication

An OLTP message-driven bean for asynchronous communication can receive a message from an EIS application, but is not allowed to send a reply message. The associated BeanConnect-specific interface is `net.fsc.jca.communication.AsyncOltpMessageListener`. The message from the EIS application may contain one or more message segments.

10.3.2 Inbound communication with openUTM partners

For inbound communication with an openUTM partner, BeanConnect supports the communication protocol OSI-TP (transactional or non-transactional) as well as transport-level protocols such as RFC1006 or the openUTM-socket protocol, all of which are non-transactional protocols.

The configuration of the openUTM partner application needs to be adapted properly in order to be able to call an OLTP message-driven bean.

You can call an OLTP message-driven bean for asynchronous communication using the OSI-TP protocol or a transport-level protocol. You can call OLTP message-driven beans for dialog communication using the OSI-TP protocol.

- Connections to BeanConnect for communication using the OSI-TP protocol can be configured with the Management Console (see Chapter 6, "Configuration of BeanConnect").
- Connections to BeanConnect for communication using a transport-level protocol must be configured in the EIS (see Chapter 7, "Adapting the configuration in the EIS partner"). The Management Console does not support the configuration of these connections.
- An LTAC must be configured in the EIS for each OLTP message-driven bean to be called using the OSI-TP protocol. The RTAC name assigned to this LTAC must be identical to the inbound service name of the inbound message endpoint which has been assigned to this EJB at the time of deployment.

- The encoding of the user messages is determined on deployment of the OLTP message-driven bean (`activation-config` properties `encodingActive` and `encoding`, see Section 4.5.1, "Configuration properties in the ejb-jar.xml" on page 132) or on the configuration of the inbound service, see Section 6.8.2, "Configuring inbound services".

The handshake functional unit of OSI-TP must not be used in communication with OLTP message-driven beans.

The messages sent by an openUTM partner application to an OLTP message-driven bean may consist of one or more message parts. Each message part can be read separately by the OLTP message-driven bean (see Section 10.3.6.3, "Program framework using the interfaces AsyncOltpMessageListener and OltpMessageListener"). Likewise, the OLTP message-driven bean may assemble the reply message from several message parts which are to be read by the openUTM partner application with a sequence of `MGET NT` calls. For details see Section 10.3.6.3, "Program framework using the interfaces AsyncOltpMessageListener and OltpMessageListener".

If you use the transport-level protocol to call an OLTP message-driven bean, the first message part sent to this bean must be prefixed with the service name that was assigned to the message endpoint corresponding to this EJB at the time of deployment.

i You can find further information in the openUTM documentation.

10.3.3 Inbound communication with CICS applications

BeanConnect supports the communication protocol LU6.2 for inbound communication with a CICS application. To allow communication to take place by means of BeanConnect, some additional restrictions and rules apply:

- Basic conversation is not supported. Basic conversation is programmed for CICS using commands that begin with GDS.
- PIP data cannot be used for the `CONNECT PROCESS` call. The data is lost.
- It is not possible to use different mode names for different connections to the same partner. For CICS/ESA V4.1 the mode name is set in the session definition and can then be selected implicitly in the program interface using the `SYSID` parameter in the `ALLOCATE` call.
- If an LU6.2 conversation to the Java EE application server cannot be opened due to internal connectivity problems in BeanConnect itself, CICS does not receive a detailed rejection message. The detailed rejection messages can only be found in one of the protocol files of BeanConnect.

- BeanConnect supports SYNCLEVEL 0 (non-transactional conversation) and SYNCLEVEL 2 (transactional conversation).The SYNCLEVEL is set in CICS-API using the SYNCLEVEL parameter for CONNECT PROCESS.
- If inbound communication uses SYNCLEVEL 2, the CICS program must call the proxy for the end of the transaction with the commands `SEND LAST` and `SYNCPOINT` or `ISSUE PREPARE`. Then the proxy terminates the transaction. CICS can request the end of the transaction either when sending the user message or after receiving the answer.
- Only one-step dialogs are possible (one `SEND INVITE` call in the CICS program). However, message and reply may comprise several parts. A `SEND` and a `RECEIVE` call must be executed for each message part. The last part is indicated by the sender with a `SEND INVITE` call.
- With SYNCLEVEL 0, the OLTP message-driven bean terminates communication. This means that CICS may send a message using `SEND INVITE` and receive the corresponding reply message using RECEIVE. Subsequently, the dialog is terminated and `SEND LAST` is no longer permissible. However, it is possible for CICS to submit `SEND LAST` instead of submitting the `SEND INVITE` / `RECEIVE` pair. Here, CICS sends the message to the OLTP message-driven bean without receiving a corresponding reply message.

i A CICS program for inbound communication must be designed and coded to comply with the Distributed Transaction Programming (DTP) paradigm. For a description of this programming paradigm, see the IBM CICS documentation, e.g. the CICS Distributed Transaction Programming Guide.

10.3.4 Inbound communication with other EIS partners (openUTM)

For inbound communication, BeanConnect supports the following EIS partners other than openUTM or CICS:

- UPIC partners
- Transport-system partners such as RFC1006 partners or openUTM-socket partners

The following rules apply for communication using the UPIC protocol or transport-level protocols (RFC1006 or the openUTM-socket protocol), all of which are non-transactional protocols:

- UPIC partners can only be used to call OLTP message-driven beans for dialog communication. UPIC message parts are mapped onto `OltpMessagePart` objects and vice versa.
- If you use the transport-level protocol to call an OLTP message-driven bean, the first message part sent to this bean must be prefixed with the inbound service name that was assigned to the message endpoint corresponding to this EJB at the time of deployment.
- The encoding of the user messages is determined on deployment of the OLTP message-driven bean (`activation-config` properties `encodingActive` and `encoding`, see Section 4.5.1, "Configuration properties in the ejb-jar.xml") or during the configuration of the inbound service (see Section 6.8.2, "Configuring inbound services").

10.3.5 Inbound communication with XATMI partners

During inbound communication, openUTM and UPIC partners that use the XATMI API are also supported. In this case, communication with the openUTM partner is always performed via the OSI-TP protocol.

The following applies to communication with XATMI partners:

- Only typed buffers of type `X_OCTET` are supported.
- An OLTP message-driven bean can use an `OltpMessageContext` object to determine whether the calling EIS partner is an XATMI partner and - if it is - what paradigm it uses (request/reply or conversational). For details, see Section 10.3.6.2, "Determining sender contexts in the OLTP message-driven bean".
- The length of a (coded) message segment must not exceed 32,000 bytes.
- In the request/reply paradigm, only one message segment may be sent.

10.3.6 BeanConnect-specific interfaces for inbound communication

The following BeanConnect-specific interfaces of the package `net.fsc.jca.communication` are supported for inbound communication:

- `AsyncOltpMessageListener` for asynchronous communication

  To receive an inbound message, the interface provides the `onMessage()` method, which contains the inbound message as a parameter.

- `OltpMessageListener` for dialog communication

  To receive an inbound message and send a reply message, the interface provides the `onMessage()` method, which contains the inbound message as a parameter and returns a reply message to the EIS application.

10.3.6.1 Programming information on OLTP message-driven beans

- An OLTP message-driven bean must implement exactly one of the following interfaces:
  - `AsyncOltpMessageListener`
  - `OltpMessageListener`

- An `OltpMessage` object may consist of one or more `OltpMessagePart` objects and/or one or more `OltpMessageRecord` objects. Whereas an `OltpMessageRecord` is of arbitrary length, an `OltpMessagePart` object must not exceed a length of 32767 bytes.

  You can retrieve the message content from `OltpMessagePart` and `OltpMessageRecord` objects as an object of one of the following types:
  - `byte[]`
  - `String`
  - `ByteContainer`

  An OLTP message-driven bean may implement the `ByteContainer` interface if it wants to exchange structured objects containing text and binary information with an EIS application. Here, code conversion needs to be performed for the text information of the structured object. For objects of type `String`, code conversion is performed by BeanConnect. For details, see the JavaDoc of BeanConnect.

- The sequence in which `OltpMessagePart`- and/or `OlpMessageRecord objects` are added to an `OltpMessage` or are returned by an `OltpMessage` corresponds to the sequence in which the message was sent or received.

- The interface `OltpMessageContext` serves two purposes:
  - It provides a method for generating a reply message.
  - It allows status information to be retrieved.

  An OLTP message-driven bean may call the methods of the interface `OltpMessageContext` only from within the `onMessage()` method.

- When calling the method `addMessagePart()` or the method `addMessageRecord()` within the method `onMessage()`, you must use the same `OltpMessage` that was used to call one of the methods `createMessagePart()` or `createMessageRecord()` respectively.

- Asynchronous OLTP message-driven beans receive asynchronous messages, which arrive independently of the initiator's availability. Therefore, neither a reply message nor an exception thrown by an asynchronous OLTP message-driven bean can be returned to the initiator of the asynchronous message.

- If transactional communication via the OSI-TP protocol is used between the EIS application and BeanConnect, the method `onMessage(OltpMessage)` of a dialog based OLTP message-driven bean participates in a distributed transaction, if the transaction attribute `Required` has been assigned to this method.

- Asynchronous OLTP message-driven beans can never be part of a transaction that is distributed between the EIS application and the application server.

- The method `onMessage()` of an asynchronous OLTP message-driven bean which has been deployed with the transaction attribute `Required` is called in the transaction, which has been started by the proxy (never by the EIS). If the transaction is rolled back, the asynchronous message is redelivered to the OLTP message-driven bean if necessary.

  The OLTP message-driven bean can detect such a situation by evaluating the delivery count value of an asynchronous `OltpMessage` object. The `redeliveryThreshold` activation-config property, which is specified at deployment of the OLTP message-driven bean, defines the number of additional attempts to deliver the message if an error occurs (see "redeliveryThreshold" on page 135).

- The support methods of the interface `EncodingDef` are available through the interface `OltpMessageContext` for exchanging messages in codes other than ASCII.

For detailed information, refer to the JavaDoc of BeanConnect.

10.3.6.2 Determining sender contexts in the OLTP message-driven bean

An OLTP message-driven bean can obtain information about the sender via the object `OltpMessageContext`. This includes, for example, the application name and host name of the EIS partner and the inbound service with which the OLTP message-driven bean was called in the proxy. It may, for example, be of interest to identify the inbound service, if several different inbound services have been assigned to a message endpoint in the proxy container.

The object `OltpMessageContext` provides the following methods for querying the sender context:

- `String getBCProxyName()`

  Name of the proxy application, fixed length of 8 characters

- `String getBCProxyHost()`

  Name of the host on which the proxy is running, fixed length of 8 characters.

- `String getBCProxyInboundService()`

  Name of the called inbound service in the proxy, fixed length of 8 characters.

- `enum BCCommunicationProtocolType getBCCommunicationProtocol()`

  Identifier for the communication protocol via which the EIS partner called the inbound service.

  – In the case of dialog communication, the type of communication protocol (or client protocol) can be determined from the enumeration class `BCCommunicationProtocolType`.

  – In the case of asynchronous communication, the protocol type of the logical access point in the proxy is passed, see also `getBCProxyLocalPartnerName()`.

  `BCCommunicationProtocolType` returns the following values:

  '2' corresponds to the protocol type OSI-TP

  '3' corresponds to the protocol type UPIC

  '5' corresponds to the protocol type RFC1006

  '6' corresponds to the protocol type SOCKET

- `String getBCPartnerTransportSelector()`

  String with a fixed length of 8 characters. In the case of asynchronous communication, blanks are passed.

In the case of dialog communication, the following elements are passed depending on the protocol type:

– Protocol type UPIC, RFC1006 or SOCKET: Partner name of the client in the proxy

– Protocol type OSI-TP and openUTM partner in BS2000: BCAM application name of the remote host

– Protocol type OSI-TP and openUTM partner on open platforms: The partner application's T selector

– Protocol type OSI-TP and CICS partner: TRANSPORT-SELECTOR which is assigned to the CICS partner in the openUTM-LU62 Gateway

- `String getBCPartnerNetworkSelector()`

  String with fixed length of 8 characters. In the case of asynchronous communication, blanks are passed.

  In the case of dialog communication, the following elements are passed depending on the protocol type:

  – Protocol type UPIC, RFC1006 or SOCKET: Processor name of the client

  – Protocol type OSI-TP and openUTM partner in BS2000: BCAM processor name of the host on which the partner application is located

  – Protocol type OSI-TP and openUTM partner on open platforms: Host name of the partner computer

  – Protocol type OSI-TP and CICS partner: NETWORK-SELECTOR assigned to the CICS partner in the openUTM-LU62 Gateway

- `String getBCProxyTransportSelector()`

  String with fixed length of 8 characters. In the case of asynchronous communication, blanks are passed.

  In the case of dialog communication, the following elements are passed depending on the protocol type:

  – Protocol type UPIC, RFC1006 or SOCKET: Application name in the proxy application (BCAMAPPL name)

  – Protocol type OSI-TP and openUTM partner: TRANSPORT-SELECTOR of the ACCESS-POINT in the proxy application

  – Protocol type OSI-TP and CICS partner: TRANSPORT-SELECTOR of the associated ACCESS-POINT in the openUTM-LU62 Gateway.

- `String getBCProxyUserId()`

  User ID in the proxy application or, if the protocol type is OSI-TP and the EIS partner has not passed any user ID, the connection name (ASSOCIATION name). Fixed length of 8 characters.

- `String getBCProxyLocalPartnerName()`

  Name of the logical access point in the proxy application. In the case of the protocol type OSI-TP, this is the OSI-LPAP name; for all other protocol types, it is the LTERM name. Fixed length of 8 characters.

- `String getBCRaMessageEndpointName()`

  Name of the called message endpoint.

- `boolean isBCPartnerXATMI()`

  `true` if the EIS partner communicates with the BeanConnect proxy via the XATMI interface, otherwise `false`.

- `boolean isBCPartnerXATMIConversational()`

  `true` if the EIS partner communicates with the BeanConnect proxy via the XATMI interface and has selected the conversational communication paradigm, otherwise `false` (i.e. request/reply paradigm).

Strings which are returned with a fixed length of 8 may be padded with blanks at the end if necessary.

10.3.6.3 Program framework using the interfaces AsyncOltpMessageListener and OltpMessageListener

An OLTP message-driven bean receives the inbound message as the `inMsg` parameter of the `onMessage()` method. The received object is an `OltpMessage` object. From the `OltpMessage` object you can retrieve an `OltpMessageContext` object which in turn contains attributes of the received message and serves dialog OLTP message-driven beans as a factory for creating a response message:

1. Access to the message context:

   ```
   OltpMessageContext oltpMsgCtx = inMsg.getMessageContext();
   ```

2. Access the message content:

   The `OltpMessage` object allows access to the message content which may be processed in the form of `OltpMessageRecord` or `OltpMessagePart` objects.

In the case of `OltpMessagePart` objects you specify:

```
String inMsgTxt = "";
if (inMsg.countMessageParts() > 0) {
  OltpMessagePart inMsgPart;
  Iterator it<OltpMessagePart> = inMsg.getMessageParts();

  for ( ; it.hasNext(); ) {
    inMsgPart = it.next();
    inMsgTxt += inMsgPart.getText();
  }
}
```

In the case of `OltpMessageRecord` objects you specify

```
if (inMsg.countMessageRecords() > 0) {
  OltpMessageRecord inMsgRec;
  Iterator it<OltpMessageRecord> = inMsg.getMessageRecords();

  for ( ; it.hasNext(); ) {
    inMsgRec = it.next();
    inMsgTxt += inMsgRec.getText();
  }
}
```

3. Creating a reply message (only in the case of OLTP message-driven beans for dialog communication):

   An OLTP message-driven bean for dialog communication uses the `OltpMessageContext` interface to create an `OltpMessage` object for the reply message:

   ```
   OltpMessage outMsg = oltpMsgCtx.createMessage();
   ```

4. The `OltpMessage` object needs to be populated with message content (only in the case of OLTP message-driven beans for dialog communication). You can do this in different ways using `OltpMessageRecord` and/or `OltpMessagePart` objects.

   – You should construct the response message using `OltpMessagePart` objects if the message recipient is to be sent a response structured in **message segments**. If the recipient is an openUTM application then it reads each message segment transferred with an `OltpMessagePart` object by means of a separate MGET call.

   – If it is not important for the response message to be structured in message segments then it is more advantageous to use `OltpMessageRecord` objects.

You will find a code sample of an OLTP message-driven bean for dialog communication on page 423 and a code sample of an OLTP message-driven bean for asynchronous communication on page 425.

10.3.7 Common Client Interface (CCI) for inbound communication

The CCI interface for the inbound communication is `javax.resource.cci.MessageListener`. This interface offers the same functionality as the BeanConnect-specific `OltpMessageListener` interface.

In addition, the interfaces `net.fsc.jca.communication.cci.BCRecord` and `net.fsc.jca.communication.OltpMessage` may be used for inbound communication.

10.3.7.1 Programming information on OLTP message-driven beans (CCI)

An OLTP message-driven bean (CCI) must implement the interface `javax.resource.cci.MessageListener`. OLTP message-driven beans (CCI) meet the requirements for dialog communication. The corresponding rules described in the section "Programming information on OLTP message-driven beans" on page 415 apply.

10.3.7.2 Program framework using the interface javax.resource.cci.MessageListener

An OLTP message-driven bean (CCI) receives the inbound message as the `record` parameter of the `onMessage()` method of the `MessageListener` interface. The received object is of type `BCRecord` and contains an `OltpMessage` object. From the `OltpMessage` object you can retrieve an `OltpMessageContext` object that in turn contains attributes of the received message and also serves dialog OLTP message-driven beans as a factory for creating a response message:

1. Extract the `OltpMessage` object from the `BCRecord` object:

   ```
   OltpMessage inMsg = ((BCRecord)record).getOltpMessage();
   ```

2. Set up the message context:

   ```
   String inMsgTxt;
   OltpMessageContext oltpMsgCtx = inMsg.getMessageContext();
   ```

3. Access the message content:

   The `OltpMessage` object allows access to the message content which may be processed in the form of `OltpMessageRecord` or `OltpMessagePart` objects. From these objects, you can retrieve the message content as an object of one of the following types:

   - `byte[]`
   - `String`
   - `ByteContainer`

In the case of `OltpMessagePart` objects you specify:

```
if (inMsg.countMessageParts() > 0) {
  OltpMessagePart inMsgPart;
  Iterator it<OltpMessagePart> = inMsg.getMessageParts();

 for ( ; it.hasNext(); ) {
    inMsgPart = it.next();
    inMsgTxt = inMsgPart.getText();
  }
 }

}
```

In the case of `OltpMessageRecord` objects you specify:

```
if (inMsg.countMessageRecords() > 0) {
  OltpMessageRecord inMsgRec;
  Iterator it<OltpMessageRecord> = inMsg.getMessageRecords();

 for ( ; it.hasNext(); ) {
    inMsgRec = (OltpMessageRecord) it.next();
    inMsgTxt = inMsgRec.getText();
  }

}
```

4. Create an `OLTPMessage` object for the reply message:

   An OLTP message-driven bean for dialog communication uses the `OltpMessageContext` interface to create an `OltpMessage` object for the reply message:

   ```
   OltpMessage outMsg = oltpMsgCtx.createMessage();
   ```

5. The `OltpMessage` object needs to be populated with the reply message content. You can do this in different ways using `OltpMessageRecord` and/or `OltpMessagePart` objects.

   In the case of `OltpMessagePart` objects you specify:

   ```
   OltpMessagePart outMsgPart = outMsg.createMessagePart();
   outMsgPart.setText("reply");
   outMsg.addMessagePart(outMsgPart);
   ```

   In the case of `OltpMessageRecord` objects you specify:

   ```
   OltpMessageRecord outMsgRec = outMsg.createMessageRecord("");
   outMsgRec.setText("reply");
   outMsg.addMessageRecord(outMsgRec);
   ```

6. Before return, the reply message needs to be set in the `BCRecord` object which is subsequently returned from this method:

   ```
   ((BCRecord)record).setOltpMessage(outMsg);
   ```

You will find a code sample in Example 18, "OLTP message-driven bean (CCI)" on page 426.

10.3.8 Code samples for inbound communication

This section contains the following code samples:

- OLTP message-driven bean for dialog communication
- OLTP message-driven bean for asynchronous communication
- OLTP message-driven bean (CCI)

Example 16 OLTP message-driven bean for dialog communication

```
package net.fsc.jca.BeanConnect.oltpmdb;

import javax.ejb.EJBException;
import javax.ejb.MessageDrivenBean;
import javax.ejb.MessageDrivenContext;
import java.util.Iterator;
import net.fsc.jca.communication.*;

public class SampleDialogOltpMdbBean
     implements MessageDrivenBean, OltpMessageListener {

 public void ejbCreate()
            throws EJBException {
    // @TODO: add code
 }

 public void setMessageDrivenContext(MessageDrivenContext ctx)
            throws EJBException {
    // @TODO: add code
 }

 public void ejbRemove()
            throws EJBException {
    // @TODO: add code
 }
```

```
  public OltpMessage onMessage(OltpMessage inMsg) {
     String inMsgTxt;
     OltpMessageContext oltpMsgCtx = inMsg.getMessageContext();

     // read request
     try {
       if (inMsg.countMessageParts() > 0) {
          OltpMessagePart inMsgPart;
          Iterator it<OltpMessagePart> = inMsg.getMessageParts();

          for ( ; it.hasNext(); ) {
            inMsgPart = it.next();
            inMsgTxt  = inMsgPart.getText();
            // @TODO: process message part
          }

          // @TODO: process request
       }
     }
     catch (Exception ex) {
       // @TODO: handle exception
     }

     // setup reply
     OltpMessage outMsg = oltpMsgCtx.createMessage();
     OltpMessagePart outMsgPart = outMsg.createMessagePart();
     try {
       outMsgPart.setText("Reply from SampleDialogOltpMdbBean");
     }
     catch (OltpMessageException ex) {
       // @TODO: add exception handling
     }
     outMsg.addMessagePart(outMsgPart);

     return (outMsg);
  }
}
```

Example 17   OLTP message-driven bean for asynchronous communication

```
package net.fsc.jca.BeanConnect.oltpmdb;

import javax.ejb.EJBException;
import javax.ejb.MessageDrivenBean;
import javax.ejb.MessageDrivenContext;
import java.util.Iterator;
import net.fsc.jca.communication.*;

public class SampleAsynOltpMdbBean
     implements MessageDrivenBean,AsyncOltpMessageListener {
  public void ejbCreate()
             throws EJBException {
    // @TODO: add code
  }

  public void setMessageDrivenContext(MessageDrivenContext ctx)
            throws EJBException {
    // @TODO: add code
  }

  public void ejbRemove()
            throws EJBException {
    // @TODO: add code
  }

  public void onMessage(OltpMessage inMsg) {
    String inMsgTxt;
    OltpMessageContext oltpMsgCtx = inMsg.getMessageContext();

    // read request

    try {
       if (inMsg.countMessageParts() > 0) {
         OltpMessagePart inMsgPart;
         Iterator it<OltpMessagePart> = inMsg.getMessageParts();

         for ( ; it.hasNext(); ) {
           inMsgPart = it.next();
           inMsgTxt  = inMsgPart.getText();
           // @TODO: process message part
         }

         // @TODO: process request
       }
    }
    catch (Exception ex) {
      // @TODO: handle exception
    }

    return;
  }
}
```

Example 18 OLTP message-driven bean (CCI)

```
package net.fsc.jca.BeanConnect.oltpmdb;

import java.util.Iterator;

import javax.ejb.EJBException;
import javax.ejb.MessageDrivenBean;
import javax.ejb.MessageDrivenContext;
import javax.resource.cci.MessageListener;
import javax.resource.cci.Record;

import net.fsc.jca.communication.OltpMessage;
import net.fsc.jca.communication.OltpMessageContext;
import net.fsc.jca.communication.OltpMessageException;
import net.fsc.jca.communication.OltpMessagePart;
import net.fsc.jca.communication.cci.BCRecord;

public class SampleCciOltpMdbBean
             implements MessageDrivenBean, MessageListener {
  public Record onMessage(Record record) {

    String inMsgTxt;
    OltpMessage inMsg = ((BCRecord)record).getOltpMessage();
    OltpMessageContext oltpMsgCtx = inMsg.getMessageContext();

    // read request
    try {
      if (inMsg.countMessageParts() > 0) {
        OltpMessagePart inMsgPart;
        Iterator it<OltpMessagePart> = inMsg.getMessageParts();

        for ( ; it.hasNext(); ) {
          inMsgPart = it.next();
          inMsgTxt  = inMsgPart.getText();
        // @TODO: process message part
      }

      // @TODO: process request

    }
  }
  catch (Exception ex) {
      // @TODO: handle exception
  }

  // setup reply

  OltpMessage outMsg = oltpMsgCtx.createMessage();
  OltpMessagePart outMsgPart = outMsg.createMessagePart();
  try {
    outMsgPart.setText("Reply from SampleCciOltpMdbBean");
  }
  catch (OltpMessageException ex) {
    // @TODO: add exception handling
  }
  outMsg.addMessagePart(outMsgPart);


  ((BCRecord)record).setOltpMessage(outMsg);
  return record;
}

/**
 * Method ejbCreate() as required by EJB spec.
 */
public void ejbCreate()
           throws EJBException {
  // @TODO: add code
}
```

```
    /**
     * Method setMessageDrivenContext() as required by interface
     * javax.ejb.MessageDrivenBean.
     * javax.ejb.MessageDrivenBean#setMessageDrivenContext(
     *                              MessageDrivenContext ctx)
     */
    public void setMessageDrivenContext(MessageDrivenContext ctx)
               throws EJBException {
      // @TODO: add code
    }

    /**
    * Method ejbRemove() as required by interface
    * javax.ejb.MessageDrivenBean.
    *
    * @see javax.ejb.MessageDrivenBean#ejbRemove()
    */
    public void ejbRemove()
               throws EJBException {
    // @TODO: add code
    }
  }
```

11 Encoding and national language support

This chapter provides the following information:

- The section "Encoding" describes the code conversion between a Java program using Unicode and the specific encoding used by the partner system.
- The section "National language support for message output"describes the BeanConnect National Language Support (NLS) feature for language- and country-specific message display from the BeanConnect resource adapter, the BeanConnect proxy and the BeanConnect Management Console.

11.1 Encoding

If BeanConnect receives printable data from partners on BS2000/OSD or on an IBM mainframe, the data stream encoded in 1-byte code (for example EBCDIC) must first be converted into 2-byte Unicode so that a Java program can process it directly. Correspondingly, conversion from 2-byte Unicode to 1-byte code is required when a Java program with BeanConnect sends data to the BS2000/OSD or CICS partner.

To convert 1-byte code to 2-byte Unicode and vice versa, you have the following options:

- Standard conversion between EBCDIC code and Unicode for EIS partners of type openUTM
- Standard conversion between EBCDIC code and Unicode for EIS partners of type CICS
- Using other predefined code tables
- Using custom charsets
- Using other predefined code tables

You can find further detailed information on the topics discussed in this chapter in the JavaDoc for BeanConnect concerning the package `net.fsc.beanta.encoding`.

All code tables used by the IBM systems can be found in

`http://www-03.ibm.com/systems/i/software/globalization/codepages.html`

11.1.1 Standard conversion between EBCDIC code and Unicode for EIS partners of type openUTM

In most cases you do not need to deal with conversion from EBCDIC code to Unicode and vice versa as BeanConnect performs conversion automatically in accordance with the standard code table `OSD_EBCDIC_DF04_DRV`.

Code conversion takes place automatically when the following requirements are met:

- The value `true` is specified in the configuration property `encodingActive`.
- Strings are used for communication.

i If the Java program is to receive the 1-byte EBCDIC data stream unconverted, byte arrays must be used instead of strings.

Code table OSD_EBCDIC_DF04_DRV

The table below shows the assignment of 1-byte EBCDIC code, printable Unicode characters, and 2-byte Unicode defined in the code table `OSD_EBCDIC_DF04_DRV`.

In the following table

- the x and y axes indicate the relevant 1-byte EBCDIC code
- the first line in a field indicates the 2-byte Unicode
  (leading non-significant bytes are not indicated)
- the second line in a field indicates the Unicode printing character

Byte - character (Unicode) correspondence

↓ →	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_A	_B	_C	_D	_E	_F
0_	00	01	02	03	85 …	09	86 †	7F	87 ‡	8D	8E Ž	0B	0C	0D	0E	0F
1_	10	11	12	13	8F	A	8	97 —	18	19	9C œ	9D	1C	1D	1E -	1F
2_	80 €	81	82 ‚	83 ƒ	84 „	92 ’	17	1B	88 ˆ	89 ‰	8A Š	8B ‹	8C Œ	5	6	7
3_	90	91 ‘	16	93 “	94 ”	95 •	96 –	4	98 ˜	99 ™	9A š	9B ›	14	15	9E ž	1A
4_	20	A0	E2 â	7C |	E0 à	E1 á	E3 ã	E5 å	E7 ç	F1 ñ	60 `	2E .	3C <	28 (	2B +	F6 ö
5_	26 &	E9 é	EA ê	EB ë	E8 è	ED í	EE î	EF ï	EC ì	0	21 !	24 $	2A *	29)	3B ;	AF ¯
6_	2D -	2F /	C2 Â	A6 ¦	C0 À	C1 Á	C3 Ã	C5 Å	C7 Ç	D1 Ñ	5E ^	2C ,	25 %	5F _	3E >	3F ?
7_	F8 ø	C9 É	CA Ê	CB Ë	C8 È	CD Í	CE Î	CF Ï	CC Ì	A8 ¨	3A :	23 #	A7 §	27 '	3D =	22 "
8_	D8 Ø	61 a	62 b	63 c	64 d	65 e	66 f	67 g	68 h	69 i	AB «	BB »	F0 ð	FD ý	FE þ	B1 ±
9_	B0 °	6A j	6B k	6C l	6D m	6E n	6F o	70 p	71 q	72 r	AA ª	BA º	E6 æ	B8 ¸	C6 Æ	A4 ¤
A_	B5 µ	7E ~	73 s	74 t	75 u	76 v	77 w	78 x	79 y	7A z	A1 ¡	BF ¿	D0 Ð	DD Ý	DE Þ	AE ®
B_	A2 ¢	A3 £	A5 ¥	B7 ·	A9 ©	40 @	B6 ¶	BC ¼	BD ½	BE ¾	AC ¬	C4 Ä	D6 Ö	DC Ü	B4 ´	D7 ×
C_	7B {	41 A	42 B	43 C	44 D	45 E	46 F	47 G	48 H	49 I	AD	F4 ô	5B [	F2 ò	F3 ó	F5 õ
D_	7D }	4A J	4B K	4C L	4D M	4E N	4F O	50 P	51 Q	52 R	B9 ¹	FB û	5D]	F9 ù	FA ú	FF ÿ
E_	5C \	F7 ÷	53 S	54 T	55 U	56 V	57 W	58 X	59 Y	5A Z	B2 ²	D4 Ô	DB Û	D2 Ò	D3 Ó	D5 Õ
F_	30 0	31 1	32 2	33 3	34 4	35 5	36 6	37 7	38 8	39 9	B3 ³	E4 ä	D9 Ù	FC ü	DA Ú	DF ß

Character (Unicode) - byte correspondence

Substitute character: **6F**

In the following table

- the x and y axes indicate the relevant 2-byte Unicode
- the relevant field indicates the associated 1-byte EBCDIC code

↓ →	**_0**	**_1**	**_2**	**_3**	**_4**	**_5**	**_6**	**_7**	**_8**	**_9**	**_A**	**_B**	**_C**	**_D**	**_E**	**_F**
000_	00	01	02	03	37	2D	2E	2F	16	05	15	0B	0C	0D	0E	0F
001_	10	11	12	13	3C	3D	32	26	18	19	3F	27	1C	1D	1E	1F
002_	40	5A	7F	7B	5B	6C	50	7D	4D	5D	5C	4E	6B	60	4B	61
003_	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	7A	5E	4C	7E	6E	**6F**
004_	B5	C1	C2	C3	C4	C5	C6	C7	C8	C9	D1	D2	D3	D4	D5	D6
005_	D7	D8	D9	E2	E3	E4	E5	E6	E7	E8	E9	CC	E0	DC	6A	6D
006_	4A	81	82	83	84	85	86	87	88	89	91	92	93	94	95	96
007_	97	98	99	A2	A3	A4	A5	A6	A7	A8	A9	C0	43	D0	A1	7
008_	20	21	22	23	24	4	6	8	28	29	2A	2B	2C	9	A	14
009_	30	31	25	33	34	35	36	17	38	39	3A	3B	1A	1B	3E	5F
00A_	41	AA	B0	B1	9F	B2	63	7C	79	B4	9A	8A	BA	CA	AF	5F
00B_	90	8F	EA	FA	BE	A0	B6	B3	9D	DA	9B	8B	B7	B8	B9	AB
00C_	64	65	62	66	BB	67	9E	68	74	71	72	73	78	75	76	77
00D_	AC	69	ED	EE	EB	EF	BC	BF	80	FC	FE	EC	BD	AD	AE	FF
00E_	44	45	42	46	FB	47	9C	48	54	51	52	53	58	55	56	57
00F_	8C	49	CD	CE	CB	CF	4F	E1	70	DD	DE	DB	FD	8D	8E	DF

11.1.2 Standard conversion between EBCDIC code and Unicode for EIS partners of type CICS

In most cases you do not need to deal with conversion from 1-byte code to Unicode and vice versa as BeanConnect performs conversion automatically in accordance with the standard JDK code table Cp1047.

Code conversion takes place automatically when the following requirements are met:

- The value `true` is specified in the configuration property `encodingActive`.
- Strings are used for communication.
- The connection URL is of type `cics://`

ℹ If the Java program is to receive the 1-byte data stream unconverted, byte arrays must be used instead of strings.

11.1.3 Using other predefined code tables

In addition to the standard code table `OSD_EBCDIC_DF04_DRV`, the following code tables are also supplied with the product BeanConnect:

- `OSD_EBCDIC_DF03_IRV` (only openUTM partners)
- `OSD_EBCDIC_DF04_1` (only openUTM partners)
- `OSD_EBCDIC_DF04_15` (only openUTM partners)
- code tables of the JVM (openUTM and CICS partners)

BeanConnect supports the code tables provided with the JVM. You will find a list of the JVM code tables

- for JDK 1.6 at

  `http://docs.oracle.com/javase/6/docs/technotes/guides/intl/encoding.doc.html`

You select the required code table at deployment time as follows:

- In the configuration property `encoding` specify the name of the required code table. You specify a JMV code table with `jdk:<jvm-code-table>`.
- Set the configuration property `encodingActive` to `true` to activate it.

You select the required code table at runtime as follows:

- In the Java program activate the required code table with the `setEncoding()` method of the `EISConnection` interface or the `OltpMessageContext` interface.

i You can use the Management Console to specify a code table for an inbound service. This specification overwrites the value for `encoding` and sets `encodingActive` to `true`, see Section 6.8.2, "Configuring inbound services".

Example 19 Using predefined code tables

Code table `OSD_EBCDIC_DF03_IRV` is to be used:

```
connection.setEncoding(Encoding.getEncoding("OSD_EBCDIC_DF03_IRV"))
```

JVM code table `CP1047` is to be used:

```
connection.setEncoding(Encoding.getEncoding("jdk:Cp1047"));
connection.setEncodingActive(true);
```

The other predefined code tables are shown on the following pages.

In all the following **Byte - Character (Unicode) Correspondence** tables

- the x and y axes indicate the relevant 1-byte EBCDIC code
- the first line in a field indicates the 2-byte Unicode
  (leading non-significant bytes are not indicated)
- the second line in a field indicates the Unicode printing character

In all the following **Character (Unicode) - Byte Correspondence** tables

- the x and y axes indicate the relevant 2-byte Unicode
- the relevant field indicates the associated 1-byte EBCDIC code

Code table OSD_EBCDIC_DF03_IRV

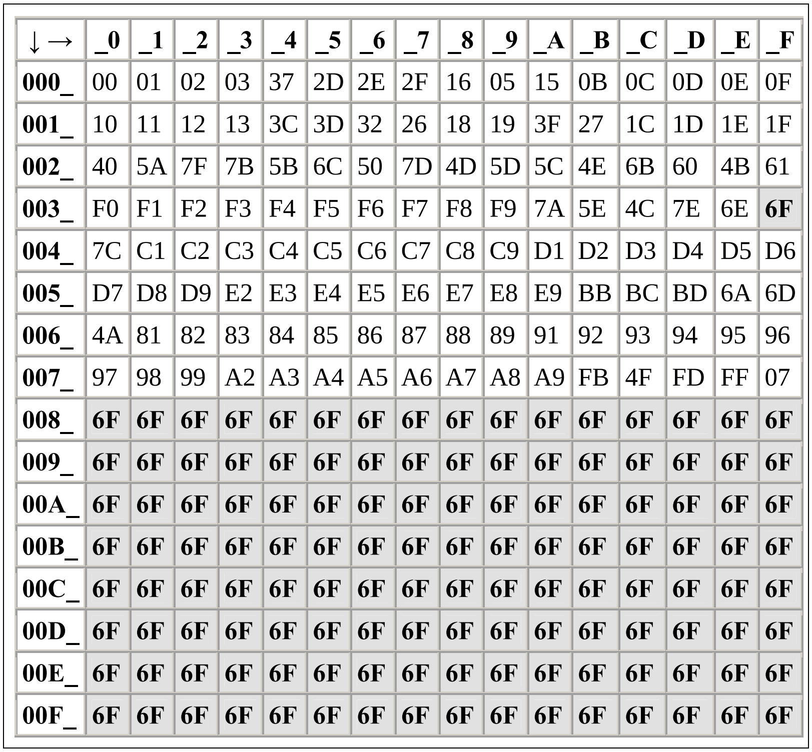
The table below shows the assignment of 1-byte EBCDIC code, printable Unicode characters, and 2-byte Unicode defined in the code table `OSD_EBCDIC_DF03_IRV`.

Byte - character (Unicode) correspondence

↓ →	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_A	_B	_C	_D	_E	_F
0_	00	01	02	03	FFFD	09	FFFD	7F	FFFD	FFFD	FFFD	0B	0C	0D	0E	0F
1_	10	11	12	13	FFFD	0A	08	FFFD	18	19	FFFD	FFFD	1C	1D	1E -	1F
2_	FFFD	FFFD	FFFD	FFFD	FFFD	FFFD	17	1B	FFFD	FFFD	FFFD	FFFD	FFFD	05	06	07
3_	FFFD	FFFD	16	FFFD	FFFD	FFFD	FFFD	04	FFFD	FFFD	FFFD	FFFD	14	15	FFFD	1A
4_	20	FFFD	FFFD	FFFD	FFFD	FFFD	FFFD	FFFD	FFFD	FFFD	60 `	2E .	3C <	28 (	2B +	7C |
5_	26 &	FFFD	FFFD	FFFD	FFFD	FFFD	FFFD	FFFD	FFFD	FFFD	21 !	24 $	2A *	29)	3B ;	FFFD
6_	2D -	2F /	FFFD	FFFD	FFFD	FFFD	FFFD	FFFD	FFFD	FFFD	5E ^	2C ,	25 %	5F _	3E >	3F ?
7_	FFFD	FFFD	FFFD	FFFD	FFFD	FFFD	FFFD	FFFD	FFFD	FFFD	3A :	23 #	40 @	27 '	3D =	22 "
8_	FFFD	61 a	62 b	63 c	64 d	65 e	66 f	67 g	68 h	69 i	FFFD	FFFD	FFFD	FFFD	FFFD	FFFD
9_	FFFD	6A j	6B k	6C l	6D m	6E n	6F o	70 p	71 q	72 r	FFFD	FFFD	FFFD	FFFD	FFFD	FFFD
A_	FFFD	FFFD	73 s	74 t	75 u	76 v	77 w	78 x	79 y	7A z	FFFD	FFFD	FFFD	FFFD	FFFD	FFFD
B_	FFFD	FFFD	FFFD	FFFD	FFFD	FFFD	FFFD	FFFD	FFFD	FFFD	FFFD	5B [	5C \	5D]	FFFD	FFFD
C_	FFFD	41 A	42 B	43 C	44 D	45 E	46 F	47 G	48 H	49 I	FFFD	FFFD	FFFD	FFFD	FFFD	FFFD
D_	FFFD	4A J	4B K	4C L	4D M	4E N	4F O	50 P	51 Q	52 R	FFFD	FFFD	FFFD	FFFD	FFFD	FFFD
E_	FFFD	FFFD	53 S	54 T	55 U	56 V	57 W	58 X	59 Y	5A Z	FFFD	FFFD	FFFD	FFFD	FFFD	FFFD
F_	30 0	31 1	32 2	33 3	34 4	35 5	36 6	37 7	38 8	39 9	FFFD	7B {	FFFD	7D }	FFFD	7E ~

Character (Unicode) - byte correspondence

Substitute character: **6F**



↓→	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_A	_B	_C	_D	_E	_F
000_	00	01	02	03	37	2D	2E	2F	16	05	15	0B	0C	0D	0E	0F
001_	10	11	12	13	3C	3D	32	26	18	19	3F	27	1C	1D	1E	1F
002_	40	5A	7F	7B	5B	6C	50	7D	4D	5D	5C	4E	6B	60	4B	61
003_	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	7A	5E	4C	7E	6E	**6F**
004_	7C	C1	C2	C3	C4	C5	C6	C7	C8	C9	D1	D2	D3	D4	D5	D6
005_	D7	D8	D9	E2	E3	E4	E5	E6	E7	E8	E9	BB	BC	BD	6A	6D
006_	4A	81	82	83	84	85	86	87	88	89	91	92	93	94	95	96
007_	97	98	99	A2	A3	A4	A5	A6	A7	A8	A9	FB	4F	FD	FF	07
008_	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**
009_	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**
00A_	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**
00B_	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**
00C_	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**
00D_	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**
00E_	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**
00F_	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**	**6F**

Code table OSD_EBCDIC_DF04_1

The table below shows the assignment of 1-byte EBCDIC code, printable Unicode characters, and 2-byte Unicode defined in the code table `OSD_EBCDIC_DF04_1`.

Byte - character (Unicode) correspondence

↓ →	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_A	_B	_C	_D	_E	_F
0_	00	01	02	03	85 …	09	86 †	7F	87 ‡	8D	8E Ž	0B	0C	0D	0E	0F
1_	10	11	12	13	8F	A	8	97 —	18	19	9C œ	9D	1C	1D	1E -	1F
2_	80 €	81	82 ‚	83 ƒ	84 „	92 ’	17	1B	88 ˆ	89 ‰	8A Š	8B ‹	8C Œ	05	06	07
3_	90	91 ‘	16	93 “	94 ”	95 •	96 –	4	98 ˜	99 ™	9A š	9B ›	14	15	9E ž	1A
4_	20	A0	E2 â	E4 ä	E0 à	E1 á	E3 ã	E5 å	E7 ç	F1 ñ	60 `	2E .	3C <	28 (	2B +	7C |
5_	26 &	E9 é	EA ê	EB ë	E8 è	ED í	EE î	EF ï	EC ì	DF ß	21 !	24 $	2A *	29)	3B ;	9F Ÿ
6_	2D -	2F /	C2 Â	C4 Ä	C0 À	C1 Á	C3 Ã	C5 Å	C7 Ç	D1 Ñ	5E ^	2C ,	25 %	5F _	3E >	3F ?
7_	F8 ø	C9 É	CA Ê	CB Ë	C8 È	CD Í	CE Î	CF Ï	CC Ì	A8 ¨	3A :	23 #	40 @	27 '	3D =	22 "
8_	D8 Ø	61 a	62 b	63 c	64 d	65 e	66 f	67 g	68 h	69 i	AB «	BB »	F0 ð	FD ý	FE þ	B1 ±
9_	B0 °	6A j	6B k	6C l	6D m	6E n	6F o	70 p	71 q	72 r	AA ª	BA º	E6 æ	B8 ¸	C6 Æ	A4 ¤
A_	B5 µ	AF ¯	73 s	74 t	75 u	76 v	77 w	78 x	79 y	7A z	A1 ¡	BF ¿	D0 Ð	DD Ý	DE Þ	AE ®
B_	A2 ¢	A3 £	A5 ¥	B7 ·	A9 ©	A7 §	B6 ¶	BC ¼	BD ½	BE ¾	AC ¬	5B [	5C \	5D]	B4 ´	D7 ×
C_	F9 ù	41 A	42 B	43 C	44 D	45 E	46 F	47 G	48 H	49 I	AD	F4 ô	F6 ö	F2 ò	F3 ó	F5 õ
D_	A6 ¦	4A J	4B K	4C L	4D M	4E N	4F O	50 P	51 Q	52 R	B9 ¹	FB û	FC ü	DB Û	FA ú	FF ÿ
E_	D9 Ù	F7 ÷	53 S	54 T	55 U	56 V	57 W	58 X	59 Y	5A Z	B2 ²	D4 Ô	D6 Ö	D2 Ò	D3 Ó	D5 Õ
F_	30 0	31 1	32 2	33 3	34 4	35 5	36 6	37 7	38 8	39 9	B3 ³	7B {	DC Ü	7D }	DA Ú	7E ~

Character (Unicode) - byte correspondence

Substitute character: **6F**

↓ →	**_0**	**_1**	**_2**	**_3**	**_4**	**_5**	**_6**	**_7**	**_8**	**_9**	**_A**	**_B**	**_C**	**_D**	**_E**	**_F**
000_	00	01	02	03	37	2D	2E	2F	16	05	15	0B	0C	0D	0E	0F
001_	10	11	12	13	3C	3D	32	26	18	19	3F	27	1C	1D	1E	1F
002_	40	5A	7F	7B	5B	6C	50	7D	4D	5D	5C	4E	6B	60	4B	61
003_	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	7A	5E	4C	7E	6E	**6F**
004_	7C	C1	C2	C3	C4	C5	C6	C7	C8	C9	D1	D2	D3	D4	D5	D6
005_	D7	D8	D9	E2	E3	E4	E5	E6	E7	E8	E9	BB	BC	BD	6A	6D
006_	4A	81	82	83	84	85	86	87	88	89	91	92	93	94	95	96
007_	97	98	99	A2	A3	A4	A5	A6	A7	A8	A9	FB	4F	FD	FF	7
008_	20	21	22	23	24	4	6	8	28	29	2A	2B	2C	9	A	14
009_	30	31	25	33	34	35	36	17	38	39	3A	3B	1A	1B	3E	5F
00A_	41	AA	B0	B1	9F	B2	D0	B5	79	B4	9A	8A	BA	CA	AF	A1
00B_	90	8F	EA	FA	BE	A0	B6	B3	9D	DA	9B	8B	B7	B8	B9	AB
00C_	64	65	62	66	63	67	9E	68	74	71	72	73	78	75	76	77
00D_	AC	69	ED	EE	EB	EF	EC	BF	80	E0	FE	DD	FC	AD	AE	59
00E_	44	45	42	46	43	47	9C	48	54	51	52	53	58	55	56	57
00F_	8C	49	CD	CE	CB	CF	CC	E1	70	C0	DE	DB	DC	8D	8E	DF

Code table OSD_EBCDIC_DF04_15

The table below shows the assignment of 1-byte EBCDIC code, printable Unicode characters, and 2-byte Unicode defined in the code table `OSD_EBCDIC_DF04_15`.

Byte - character (Unicode) correspondence

↓ →	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_A	_B	_C	_D	_E	_F
0_	00	01	02	03	85 …	09	86 †	7F	87 ‡	8D	8E Ž	0B	0C	0D	0E	0F
1_	10	11	12	13	8F	A	8	97 —	18	19	9C œ	9D	1C	1D	1E -	1F
2_	80 €	81	82 ‚	83 ƒ	84 „	92 ’	17	1B	88 ˆ	89 ‰	8A Š	8B ‹	8C Œ	5	6	7
3_	90	91 ‘	16	93 “	94 ”	95 •	96 –	4	98 ˜	99 ™	9A š	9B ›	14	15	9E ž	1A
4_	20	A0	E2 â	E4 ä	E0 à	E1 á	E3 ã	E5 å	E7 ç	F1 ñ	60 `	2E .	3C <	28 (	2B +	7C |
5_	26 &	E9 é	EA ê	EB ë	E8 è	ED í	EE î	EF ï	EC ì	DF ß	21 !	24 $	2A *	29)	3B ;	9F Ÿ
6_	2D -	2F /	C2 Â	C4 Ä	C0 À	C1 Á	C3 Ã	C5 Å	C7 Ç	D1 Ñ	5E ^	2C ,	25 %	5F _	3E >	3F ?
7_	F8 ø	C9 É	CA Ê	CB Ë	C8 È	CD Í	CE Î	CF Ï	CC Ì	161 š	3A :	23 #	40 @	27 '	3D =	22 "
8_	D8 Ø	61 a	62 b	63 c	64 d	65 e	66 f	67 g	68 h	69 i	AB «	BB »	F0 ð	FD ý	FE þ	B1 ±
9_	B0 °	6A j	6B k	6C l	6D m	6E n	6F o	70 p	71 q	72 r	AA ª	BA º	E6 æ	17E ž	C6 Æ	20AC €
A_	B5 µ	AF ¯	73 s	74 t	75 u	76 v	77 w	78 x	79 y	7A z	A1 ¡	BF ¿	D0 Ð	DD Ý	DE Þ	AE ®
B_	A2 ¢	A3 £	A5 ¥	B7 ·	A9 ©	A7 §	B6 ¶	152 Œ	153 œ	178 Ÿ	AC ¬	5B [	5C \	5D]	17D Ž	D7 ×
C_	F9 ù	41 A	42 B	43 C	44 D	45 E	46 F	47 G	48 H	49 I	AD	F4 ô	F6 ö	F2 ò	F3 ó	F5 õ
D_	160 Š	4A J	4B K	4C L	4D M	4E N	4F O	50 P	51 Q	52 R	B9 ¹	FB û	FC ü	DB Û	FA ú	FF ÿ
E_	D9 Ù	F7 ÷	53 S	54 T	55 U	56 V	57 W	58 X	59 Y	5A Z	B2 ²	D4 Ô	D6 Ö	D2 Ò	D3 Ó	D5 Õ
F_	30 0	31 1	32 2	33 3	34 4	35 5	36 6	37 7	38 8	39 9	B3 ³	7B {	DC Ü	7D }	DA Ú	7E ~

Character (Unicode) - byte correspondence

Substitute character: **6F**

↓ →	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_A	_B	_C	_D	_E	_F
000_	00	01	02	03	37	2D	2E	2F	16	05	15	0B	0C	0D	0E	0F
001_	10	11	12	13	3C	3D	32	26	18	19	3F	27	1C	1D	1E	1F
002_	40	5A	7F	7B	5B	6C	50	7D	4D	5D	5C	4E	6B	60	4B	61
003_	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	7A	5E	4C	7E	6E	**6F**
004_	7C	C1	C2	C3	C4	C5	C6	C7	C8	C9	D1	D2	D3	D4	D5	D6
005_	D7	D8	D9	E2	E3	E4	E5	E6	E7	E8	E9	BB	BC	BD	6A	6D
006_	4A	81	82	83	84	85	86	87	88	89	91	92	93	94	95	96
007_	97	98	99	A2	A3	A4	A5	A6	A7	A8	A9	FB	4F	FD	FF	7
008_	20	21	22	23	24	4	6	8	28	29	2A	2B	2C	9	A	14
009_	30	31	25	33	34	35	36	17	38	39	3A	3B	1A	1B	3E	5F
00A_	41	AA	B0	B1	9F	B2	D0	B5	79	B4	9A	8A	BA	CA	AF	A1
00B_	90	8F	EA	FA	BE	A0	B6	B3	9D	DA	9B	8B	B7	B8	B9	AB
00C_	64	65	62	66	63	67	9E	68	74	71	72	73	78	75	76	77
00D_	AC	69	ED	EE	EB	EF	EC	BF	80	E0	FE	DD	FC	AD	AE	59
00E_	44	45	42	46	43	47	9C	48	54	51	52	53	58	55	56	57
00F_	8C	49	CD	CE	CB	CF	CC	E1	70	C0	DE	DB	DC	8D	8E	DF

Results for entries > 0x00FF

Zeichen (in)		Byte (out)
152	Œ	B7
153	œ	B8
160	Š	D0
161	š	79
178	Ÿ	B9
17D	Ž	BE
17E	ž	9D
20AC	€	9F

11.1.4 Using custom charsets

There are two ways of generating user-defined character sets:

- You can generate a custom charset provider that is embedded in the JVM.
- You can generate a legacy code table.

11.1.4.1 Custom Charset Provider

This is the preferred method for using new custom charsets.

You can create your own code tables for use in the JVM and activate them by applying the procedure described in Section 11.1.3, "Using other predefined code tables". Instead of a predefined JDK code table name, use your own code table name with the syntax `jdk:<my_code_table>`.

A detailed description on how to implement and deploy your own code tables in the Java runtime environment can be found in the Java documentation at

`http://docs.oracle.com/javase/6/docs/api/java/nio/charset/package-summary.html`

and in the BeanConnect JavaDoc for the package `net.fsc.beanta.encoding`.

A Java sample code is present in the BeanConnect delivery scope in the package `net.fsc.beanconnect.encoding.sample`.

In order to use your own code tables, you have to place the classes in the directory `<JDK>/jre/lib/ext`. Further information is available in "Java Extension Mechanism".

11.1.4.2 Creating and using legacy code tables

You use the method described here if you do not want to extend the JVM for user-defined character sets as described inSection 11.1.4.1, "Custom Charset Provider" but instead the extension is only to apply to BeanConnect.

If the predefined code tables do not satisfy your requirements, you can create code tables of your own. A Java sample source for a user-defined code table is supplied with the product. You will find this example in the JavaDoc for the package `net.fsc.beanta.encoding` in the `Encoding.CustomEncoder` class.

To avoid problems with the class loaders when using the classes of your own code tables, you should add the classes to the file `BeanConnect.rar`.

You select the required code table at deployment time as follows:

- In the configuration property `encoding`, specify the name of the required code table using `custom:<my-code-table>`.
- Set the configuration property `encodingActive` to `true` to activate it.

You select the required code table at runtime as follows:

- In the Java program activate the required code table with the `setEncoding()` method of the `EISConnection` or the `OltpMessageContext` interface as follows:

```
OwnTable myTable = new OwnTable();
connection.setEncoding(new Encoding.CustomEncoder(myTable));
connection.setEncodingActive(true);
```

i For compatibility reasons, not only the procedure described for using customer-defined code tables is supported, but also the following procedure for utilizing user-defined code tables in which conversion is implemented with two Java programs:

```
OwnTable_ByteToChar myTableByteToChar = new
OwnTable_ByteToChar();
OwnTable_CharToByte myTableCharToByte = new
OwnTable_CharToByte();
setEncoding(new Encoding.Custom(myTableByteToChar,
                                myTableCharToByte)
);
```

These code tables only support 1-byte encoding. This means that the encoding source and destination for one `char` can only be one byte. If you need 2-byte code tables, we recommend the use of a custom charset which provides all options of code conversion.

11.2 National language support for message output

BeanConnect permits internationalization and localization of the messages of the BeanConnect resource adapter, the BeanConnect proxy and the BeanConnect Management Console. Internationalization and localization means that the messages that a component passes to the users or writes to log files are output correctly for the relevant environment (country, language).

For this, Java offers classes which provide support for internationalization (such as the class `Locale`). To determine the language in which messages are to be output, there are two criteria in BeanConnect:

- language (for example en)
- region (for example US)

The language code is defined as per ISO-639. The possible values for the region are described in ISO-3166.

By default, BeanConnect supplies messages for the language *en* and the region *US*. If the user does not make any explicit selection, the default values apply as specified by the current JVM (`Default-Locale`).

BeanConnect provides two methods of changing these default settings:

- On starting a Java program, set the locale by defining the following two system properties:
  - `net.fsc.tpbasics.i18n.defaultCountry` and
  - `net.fsc.tpbasics.i18n.defaultLanguage`
- Store the required settings in the file `beanconnect_i18n.properties`, which must be accessible in the class path of the relevant JVM.

BeanConnect first tries to determine the settings using the system properties. If no system properties have been defined, the system searches for the property file and reads the settings from there. The standard method should be to set the locale in `beanconnect_i18n.properties`.

Default configuration

A JAR file (BeanConnectI18N.jar) is deployed with the installation of the BeanConnect resource adapter, the BeanConnect proxy or the BeanConnect Management Console. This file must be entered in the class path of the JVM. By default, this file has the following contents:

- beanconnect_i18n.properties for setting the locale
- the message files for the individual components (default language: en, US)
- a number of Java classes (Msg…class) which must remain unchanged in the JAR file.

To allow access to the message files, their names must follow a certain pattern.

Several message files with the name <component>.properties are included. These files act as a fallback and are used if you have given incorrect values for the language and region. The currently used message files are:

Resource adapter	basics.properties stub.properties proxy.properties ui.properties oltpmsg.properties jconnect.properties encoding.properties
Proxy	proxy.properties oltpmsg.properties
Management Console	mc.properties tpbasics.properties

In addition, a file <component>_en_US.properties can be contained in the JAR file. This is generally an identical copy of the file <component>.properties, since en_US is the default value for BeanConnect message files.

Examples of file names:

- proxy.properties
- proxy_de.properties
- proxy_en_US.properties
- proxy_fr_FR.properties

Introducing a new language

You can provide support for other languages by adding message files to the JAR file `BeanConnectI18N.jar`. These message files are text files and can be edited with a text editor (such as notepad, vi).

The following example shows the steps necessary for support of German messages.

1. Extract the default message file (for example `proxy.properties`) from the JAR file using the command `jar` or the program WinZip.
2. Translate the messages into German.
3. Store the new message file with the name `proxy_de_DE.properties` and add it to the JAR file under the same path (`net.fsc.tpbasic.i18n.r`) as the default file.
4. Extract the file `beanconnect_i18n.properties`.
5. Enter `de` as the language and `DE` as the country:

   ```
   net.fsc.tpbasics.i18n.defaultLanguage=de
   ```

   ```
   net.fsc.tpbasics.i18n.defaultCountry=DE
   ```
6. Write the file `beanconnect_i18n.properties` back to the JAR file.

Apply step 1 to 3 to all the other message files (`stub.properties`, `ui.properties`, etc.).

When switching to a new language, note that `BeanConnectI18N.jar` must be edited for the BeanConnect resource adapter in the file `BeanConnect.rar` if the connector is deployed.

For the BeanConnect proxy and the BeanConnect Management Console the `BeanConnectI18N.jar` file must be edited in the BeanConnect home directory `<BC_home>/lib`.

12 High availability and scalability

The BeanConnect proxy container is installed with default configuration values. For heavy-load operations it may be necessary to change this configuration. In most cases a specific message is output to the proxy container logging when configuration limits are reached.

This chapter provides information on:

- Shared memory of the proxy container
- Number of proxy container processes
- Page pool area and cache of the proxy container
- Number of parallel connections to the EIS partner
- Asynchronous processing in the proxy container
- OSI-SCRATCH-AREA in the proxy container
- Number of semaphores in the proxy container

12.1 Shared memory of the proxy container

All messages sent and received by the proxy container are stored in a shared memory before executing in one of the proxy container processes. During communication scenarios with large quantities of data, it may happen that the shared memory in which messages are buffered is too small.

You can solve this problem via the environment variables `UTM_IPC_LETTER` and `UTM_IPC_EXPT_LETTER`, which are specified in the start procedure of the proxy container:

- Solaris and Linux systems:
  `<proxy_cont_home>/shsc/startcontainer.sh`
- Windows systems:
  `<proxy_cont_home>\shsc\startcontainer.cmd`

12.1.1 Adapting the shared memory

The necessary modification depends on the insert in the `U189` message:

- **U189** `&OBJ1` **(** `&PTRM, &PRNM` **): IPC shortage of LETT EXTP FULL** or
  U189 `&OBJ1` **(** `&PTRM, &PRNM` **): IPC shortage of LETT MAX ILETT** or
  U189 `&OBJ1` **(** `&PTRM, &PRNM` **): IPC shortage of LETT MAX OLETT**

  These inserts mean that the size of a message exceeds the maximum size of the data area for a connection. The default maximum size after installation is 32 (specified in 4KB units).

  To correct it, you must change the value of the `UTM_IPC_EXTP_LETTER` environment variable to a bigger value.

- **U189** `&OBJ1` **(** `&PTRM, &PRNM` **): IPC shortage of LETT IPC FULL**

  These inserts mean that the size of all messages exceeds the maximum size of the data area for all connections. The default maximum size after installation is 1600 (specified in 4KB units).

  To correct it, you must change the value of the `UTM_IPC_LETTER` environment variable to a bigger value.

In most cases, however, the following "rule of thumb" can be used:

- `UTM_IPC_EXTP_LETTER` = maximum size of an outbound/inbound message in 4KB units
- `UTM_IPC_LETTER` = `<nConn>` * `UTM_IPC_EXTP_LETTER`, but not less than 1600 (6.4 MB).

  `<nConn>` is the maximum number of connections between the proxy container and its partner applications (EIS partner, resource adapter, Management Console).

12.2 Number of proxy container processes

The maximum number of processes required in the proxy container is as high as the sum of processes required for outbound and inbound communication.

Outbound communication

For outbound communication, the number of processes required of the bean is the number of parallel connections of the bean if the connections are not organized in groups. If the connections are organized in groups, the number of processes required is the number of connection groups used at a time.

In addition, the total number of processes depends on the number of beans using outbound communication at a time and whether this communication is transactional or non-transactional. In the case of transactional communication, a proxy process is exclusively assigned to a bean until the transaction is completed; in the case of non-transactional communication this is done only for the time required for a conversation.

Inbound communication

For inbound communication, the number of required processes is at least as high as the maximum number of parallel inbound messages. The parallel inbound messages can at most be as many as the number of parallel connections to all EIS partners.

12.2.1 Displaying the workload of processes

You can display the workload of the container application in the Management Console:

1. Open the subtree of the proxy in the navigation area.
2. Under **Advanced Features** choose the entry **Properties/Statistics**.

The value for the workload is given in the **Properties / Statistics** panel under **Workload**. If the value under **Workload (maximum)** is higher than 80%, you should increase the number of processes (see the following section).

12.2.2 Setting the number of processes

By default, three processes are started. You can change the number of processes in the Management Console:

1. From the context menu of the proxy, select the entry **Edit Properties**.
2. In the **Edit Properties of Local/Remote Proxy** property sheet choose the **Performance Settings** tab.
3. Under **Number of Proxy Container Processes** change the value for **Total**.

The maximum number of processes is limited by internal settings to 250. You should also take into account that a higher number of processes also increases the system overhead for managing those processes and affects the size of the recovery file.

Moreover, additional virtual memory is required for each container process. For example, when running on Solaris systems in 32-bit mode, each additional `utmwork` process requires approximately 30-40 MB of additional memory. The precise requirement is dependent on the configuration.

⚠ **Caution!**
If too little physical memory (RAM) is made available then the system transfers memory areas to hard disk (paging). This can drastically slow down operation.

If there are more than 50 processes to be started, you additionally have to change the statements `MAX TASKS`, `MAX TASKS-IN-PGWT` and/or `MAX ASYNTASKS` in the file `input.system.txt` manually.

You find the file `input.system.txt`

- on Solaris and Linux systems under the directory:
  `<proxy_cont_home>/def`
- on Windows systems under the directory:
  `<proxy_cont_home>\def`

Adapt the statements as described in the following text:

- `MAX TASKS=<number>`

  To increase the number of processes, increase the value for `<number>` in this statement. `<number>` specifies the maximum number of processes.

- `MAX TASKS-IN-PGWT=<number>`

  `<number>` specifies the maximum number of messages for outbound communication, which can be processed in parallel.

- `MAX ASYNTASKS=<number>`

  `<number>`specifies the maximum number of asynchronous messages for inbound communication, which can be processed in parallel.

To activate these changes you have to carry out an update configuration and restart the proxy (see "Saving and activating the configuration of the BeanConnect proxy" on page 233):

1. From the context menu of the proxy, choose the entry **Update Configuration**.
2. From the context menu of the proxy, choose the entry **Start Proxy**.

12.3 Page pool area and cache of the proxy container

The required memory size for the proxy container depends on the size and the number of the outbound and inbound messages that must be processed in parallel. Asynchronous outbound and inbound messages occupy memory in the proxy container until they are sent or processed completely.

The cache of the proxy container is a shared memory. Therefore the main memory required on the computer on which the proxy container is running depends on the cache size.

- If the **Proxy Container Mode** is set to **Performance Enhanced (Non-durable Asynchronous Processing)** (default setting) and the main memory is sufficient, you should set the cache to the same size as the page pool to avoid read/write access due to a cache shortage.
- If the **Proxy Container Mode** is set to **Durable Asynchronous Processing**, the cache can only save on read access, as the data is always saved in the page pool.

The setting for the **Proxy Container Mode** is to be found as described below:

1. From the context menu of the proxy, select the entry **Edit Properties**.
2. In the **Edit Properties of Local/Remote Proxy** property sheet choose the **Performance Settings** tab.

i You must activate the **Expert Mode** to display the **Performance Settings** tab.
For detailed information refer to the online help system of the Management Console.

Changing the size of the storage areas

After installation, the size of the page pool area as well as the cache size is set to 20 MB. You can change the size of these storage areas in the Management Console:

1. From the context menu of the proxy, select the entry **Edit Properties**.
2. In the **Edit Properties of Local/Remote Proxy** property sheet choose the **Performance Settings** tab.
3. Under **Proxy Container Storage Area Sizes** change the value for **Pagepool (MB)** and **Cache (MB)**.

12.4 Number of parallel connections to the EIS partner

Parallel connections to an EIS partner via the OSI-TP or LU6.2 protocol

The number of parallel connections required from the proxy container to an EIS partner depends on the maximum number of parallel messages that can be exchanged simultaneously between the applications in the application server and the EIS partners

You change the number of connections to the EIS partner in the Management Console:

1. Open the subtree of the proxy in the navigation area.
2. Open the subtree below **EIS Partners**.
3. From the context menu of the according EIS partner, choose the entry **Edit Properties**.
4. On the tab **General** specify the value for **Connections**.

The Management Console outputs the **Number Of Connections Modified** dialog box where it suggests that you also modify the values of the other connection parameters. To accept this suggestion, click **Accept**.

If you reject the suggestion with **Cancel** then please note that you must set sensible values for the other connection parameters yourself:
If more outbound than inbound communication is carried out via the connections to an EIS partner, less than half of the connections should be defined as contention winner. The number has to be adjusted to the number of contention winners defined in the EIS configuration. The sum of both definitions has to be equal to the number of connections.

Number of parallel connections for inbound communication via the UPIC, RFC1006 or openUTM Socket protocol

You can change the number of UPIC, socket and RFC1006 connections for the inbound communication in the Management Console:

1. From the context menu of the proxy, select the entry **Edit Properties**.
2. In the **Edit Properties of Local/Remote Proxy** property sheet choose the **Performance Settings** tab.
3. Under **Number of Parallel Connections** change the value for **Inbound UPIC**, **Inbound Socket** or **Inbound RFC1006**.

These values also refer to the number of parallel connections to all EIS systems which communicate using these protocols. The default value for all three partner types is 10 parallel connections.

i The Management Console itself also requires a UPIC connection to communicate with the proxy.

12.5 Asynchronous processing in the proxy container

For the settings for asynchronous processing in the proxy container the duration of the requests has to be considered as well as whether inbound or outbound communication is used.

12.5.1 Duration of asynchronous requests

Asynchronous outbound requests which have not yet been sent to the EIS will be deleted by default when the proxy container is stopped. This applies to

- scheduled requests which have not yet reached their execution time.
- requests which could not be sent yet because there is no connection to the EIS partner.

Asynchronous inbound requests which have not yet been started will also be deleted when the proxy container is stopped. This applies to

- requests which have not yet been sent to the application server.
- requests which have to be sent to the application server again (redelivery).

If you want asynchronous requests to last longer, set the **Proxy Container Mode** to **Durable Asynchronous Processing** via the Management Console. In this case you should increase the page pool size too, if necessary (see "Changing the size of the storage areas" on page 453).

You change the setting for the **Proxy Container Mode** in the Management Console:

1. From the context menu of the proxy, select the entry **Edit Properties**.
2. In the **Edit Properties of Local/Remote Proxy** property sheet choose the **Performance Settings** tab.
3. Under **Proxy Container Mode** choose the option for **Durable Asynchronous Processing**.

i You must activate the **Expert Mode** to display the **Performance Settings** tab.
For detailed information refer to the online help system of the Management Console.

12.5.2 Inbound communication

The number of processes that may be active in parallel for processing an asynchronous inbound message depends on the application scenario, but may only be as high as the number of available processes minus one. By default, the proxy is started with the setting that a maximum of two processes at a time may be active for processing asynchronous inbound messages.

You can change the number of processes in the Management Console:

1. From the context menu of the proxy, select the entry **Edit Properties**.
2. In the **Edit Properties of Local/Remote Proxy** property sheet choose the **Performance Settings** tab.
3. Under **Number of Proxy Containers** change the value for **For Asynchronous Jobs**.

ℹ You must activate the **Expert Mode** to display the **Performance Settings** tab.
For detailed information refer to the online help system of the Management Console.

12.6 OSI-SCRATCH-AREA in the proxy container

The size of the dynamic storage area for OSI-TP connections is defined in the `input.system.txt` file with the following statement:

```
MAX OSI-SCRATCH-AREA=<value>
```

`<value>` indicates the size of the storage area in KB.
The default value is 1024 (1MB), the maximum value is 32767 KB.

You find the file `input.system.txt`

- on Solaris and Linux systems under the directory:
  `<proxy_cont_home>/def`
- on Windows systems under the directory:
  `<proxy_cont_home>\def`

Adjusting the storage area

If the dynamic storage area is not sufficient during operation (e.g. due to a large number of parallel connections), the execution of the proxy container is aborted with the message `K060`. The message contains an insert specifying the reason for the abortion whose description indicates a storage shortage, e.g. `SACT28`. Reasons for an abortion are described in Chapter 13, "Logging, diagnostics and troubleshooting".

In this case you should change the value of the statement `MAX OSI-SCRATCH-AREA=<value>`. As an initial measure it makes sense to double the value. Then you must update the proxy configuration (e.g. in the Management Console: **Update Configuration** command in the context menu of the proxy container).

12.7 Number of semaphores in the proxy container

The proxy container uses a range of semaphore keys for global operations. If you run a large number of container processes the maximum value should be increased. In the case of a shortage, you will find the message `U189` with the insert `SEMA USED` in the container logging file (see on page 543).

Increasing the number of semaphores

The number of semaphores is defined in the `input.system.txt` file with the following statement:

```
MAX SEMARRAY=(<key>,<number>)
```

You find the file `input.system.txt`

- on Solaris and Linux systems under the directory:
  `<proxy_cont_home>/def`
- on Windows systems under the directory:
  `<proxy_cont_home>\def`

To increase the number of semaphores, increase the value for `<number>` in the `MAX SEMARRAY` statement. For calculating `<number>` the following formula applies:

```
<number>=nProc/20+2
```

`<nProc>` specifies the number of container processes.

13 Logging, diagnostics and troubleshooting

This chapter provides information on the following topics:

- Logging with Log4j
- Logging with JDK logging
- Configuring logging with Log4j
- Diagnosis of the BeanConnect resource adapter
- Diagnosis of the BeanConnect proxy container
- Diagnosis of the BeanConnect Management Console
- Diagnosis of the openUTM-LU62 Gateway
- Diagnosis of SNAP-IX for Solaris systems
- Diagnosis of the IBM Communications Server for Linux
- Diagnosis of the IBM Communications Server for Windows systems
- Collecting diagnostic information
- Error messages of the BeanConnect proxy container
- Error messages of the openUTM-LU62 Gateway
- Error codes

i Since the BeanConnect proxy is based on openUTM, you will also require the openUTM manual "Messages, Debugging and Diagnostics" for diagnostic operations.

13.1 Logging with Log4j

BeanConnect uses the software product Log4j for logging functionality. Log4j is part of the Apache Jakarta project. Log4j offers interfaces for logging information (runtime information, trace records, etc.) and for configuring log output.

Log4j uses configuration files written in XML. The names of these files are fixed. All BeanConnect components using Log4j are delivered with preconfigured configuration files.The names of the output files are predefined in the configuration files. Different BeanConnect components use different names for their output files.

This section provides information on the following topics:

- Basic principles of Log4j
- Loggers
- Appenders
- How the rolling file appender works

> ℹ You can find more detailed information on Log4j at
> `http://logging.apache.org/log4j/2.x/manual/index.html`

13.1.1 Basic principles of Log4j

Log4j uses two main components: the “logger”, which is the source of the messages, and the “appender”, which defines the destination of the messages. The messages transferred to a logger are output by all the appenders assigned to this logger.

13.1.1.1 Loggers

A logger is a source of messages. A program that is to write log information obtains logger objects from Log4j and outputs its messages via these objects.

Name space
The logger name space has a hierarchical structure. The naming convention is the same as that for Java packages. In other words, the individual levels of the hierarchy are separated from each other by dots in the name. Within this hierarchy, loggers inherit their properties from their parent loggers unless they have explicitly defined their own properties. The root of the hierarchy is formed by the “root logger”, which does not have its own name but is always present.

Example 20 Logger name space

The logger called `BeanConnect` is the (direct) parent logger of the logger called `BeanConnect.info` and is also the parent logger of the logger `BeanConnect.Datasources.OLTP`.

Level

The level is a property that can be assigned to both a logger and a message. When the logger is called, the logging program decides which level a message has. Depending on which level has been assigned to the relevant logger, Log4j decides whether or not the transferred message is to be logged. Messages are logged if the message level is greater than or equal to the logger level. The logger is disabled if the level is `OFF`.

Log4j supports the following levels (in descending order):

Level	Meaning
`FATAL`	Serious error, highest level
`ERROR`	Error
`WARN`	Warning
`INFO`	Information
`DEBUG`	Debug output
`TRACE`	Trace output, lowest level
`OFF`	Can only be assigned to loggers. All messages output via this logger are suppressed. The logger is switched off.

Example 21 Logging level

If a logging event with the level `DEBUG` is passed to a logger assigned to the level `INFO`, this message is suppressed, but `INFO`, `WARN`, `ERROR` and `FATAL` messages are still output.

If you set the logger level `ERROR`, only messages with the level `ERROR` and `FATAL` are output.

13.1.1.2 Appenders

The message destinations are defined by appenders. Log4j provides a number of different predefined appenders, including the following:

- File appender

  The messages are written to a file.

- Console appender

  The messages are written to `System.out` or `System.err`.

- Socket appender

  The messages are written to a socket and can thus also be sent between computers to a Log4j socket reader that can further process the messages (see Section 13.3.3, "Configuring the BeanConnect Management Console as a Log4j socket reader" for the resource adapter and for the proxy).

- Rolling file appender

  The messages are written to a file. When the specified extent threshold is reached, the file is closed and the messages are written to a new trace file.

The logging events transferred to a logger are output through the appender(s) assigned to the logger.

Example 22 Output of logging events through the appender

If there is a logger called `Trace` to which the appenders `Console` (console appender) and `File` (file appender) are assigned, any message that is output by this logger is output through both the file appender and the console appender. It thus appears both in the file and on the console.

13.1.1.3 How the rolling file appender works

The rolling file appender enables logging to a file with several backup files. You can configure the maximum amount of disk space that the trace files may use. The rolling file appender works in a single tasking environment (as in the resource adapter) as well as in a multi tasking environment (as in the proxy container).

Logging files

The rolling file appender creates the files `<File>.gen` and (in a multi tasking configuration) `<File>.lck` for internal use only and requires these in operation. `<File>` is the name of the current logging file, e.g. `BeanConnect.logging.txt`. The file in the given example is specified in the configuration file for the `BeanConnectShortLoggingFile` appender.

The rolling file appender always writes to the logging file `<File>`. At switch-over:

- The file `<File>` is copied to a backup file.
- The file `<File>` is written again.

At switch-over, the oldest existing backup files are deleted when:

- After the file is copied, there are more backup files than the value specified for `<MaxNbrBackupFiles>` and
- `<MaxNbrBackupFiles>` is greater than 0.

Specifications in angle brackets (<>) such as`<File>` or `<MaxNbrBackupFiles>` are appender properties. For further information, see Section 13.5.2, "Predefined logging configuration of a resource adapter" and Section 13.6.1, "Predefined logging configuration of a proxy".

Example 23 Logging files

In this example, `<File>=<Filename>.txt and <MaxNbrBackupFiles>=3`.

Situation before switch-over:

<Filename>.txt	In use
<Filename>.13.txt	Oldest backup file
<Filename>.14.txt	
<Filename>.15.txt	Newest backup file

At switch-over, `<Filename>.txt` is copied to `<Filename>.16.txt`, `<Filename>.13.txt` is deleted, and `<Filename>.txt` is then written again.

Situation after switch-over:

<Filename>.txt	In use
<Filename>.14.txt	Oldest backup file
<Filename>.15.txt	
<Filename>.16.txt	Newest backup file

13.2 Logging with JDK logging

The Apache Log4j package is used as standard for logging in BeanConnect products. Since the BeanConnect resource adapter runs in an application server, it is possible that Log4j should/may not be used in the server environment.

To solve this problem, the BeanConnect Logging Framework is used at all points within the resource adapter. This logging framework is based on the Apache Jakarta Commons Logging 1.0.4 package with BeanConnect specific extensions.

The BeanConnect Logging Framework supports the following logging packages as standard:

- Apache Log4j
- Java Logging API

Logging with Apache Log4j is the default setting. It is used if you do not make any changes to the BeanConnect RAR archive supplied with the resource adapter.

To use Java Logging, proceed as follows

- Delete the file `BeanConnectLog4j.jar` from the resource adapter's BeanConnect RAR archive.

JDK logging functions in a similar way to Log4j, see Section 13.1.1, "Basic principles of Log4j".

You must enter the BeanConnect-specific logger manually in a JDK logging configuration file. For a description of JDK logging, refer to the Java documentation under `http://docs.oracle.com/javase/6/docs/technotes/guides/logging/overview.html`

13.3 Configuring logging with Log4j

This section contains information on the following topics:

- Configuring logging for BeanConnect resource adapter and proxy
- Editing the Log4j configuration file using the BeanConnect Management Console
- Configuring the BeanConnect Management Console as a Log4j socket reader
- Displaying the logging events in the BeanConnect Management Console
- Display the Log4j logging file using the BeanConnect Management Console

13.3.1 Configuring logging for BeanConnect resource adapter and proxy

The following Log4j configuration file is preconfigured for the **BeanConnect resource adapter**:
`BeanConnect.log4j.properties.xml`. After installation of the resource adapter (see Chapter 3, "Installing BeanConnect"), this XML file is stored in the `config` subdirectory.

ℹ If you want to use the Management Console to process the file `BeanConnect.log4j.properties.xml` for the resource adapter then an MC-CmdHandler must be available on the computer on which the application server is running.

The Log4j configuration file `log4j.properties.xml` is preconfigured for the **BeanConnect proxy**. After installation of the proxy (see Chapter 3, "Installing BeanConnect"), this XML file is stored in the proxy's `config` subdirectory.

ℹ The changes to the logging configuration do not take effect until you start or save and restart the proxy (**Save/Restart** command - **Save & Restart** in the proxy's context menu).

You can edit the configuration entries in both configuration files using a text editor. Alternatively, you can edit the file using the Management Console as described below. The Management Console allows you to add loggers or to configure and remove loggers and appenders.

13.3.1.1 Configuring loggers

Use the Management Console to adapt the configuration of the loggers of a resource adapter:

1. Open the Log4j configuration file as follows:

   BeanConnect resource adapter:

   – In the resource adapters' navigation tree, click the node **Log4j Configuration**.

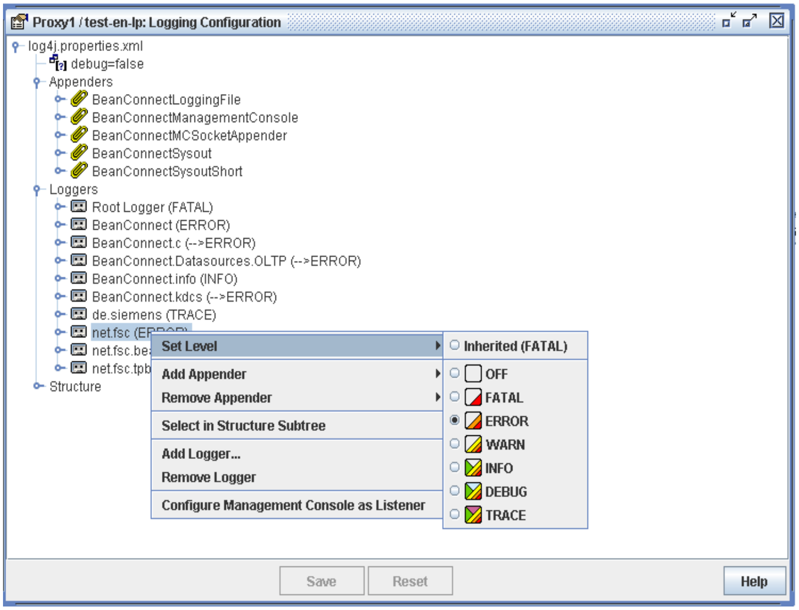
   BeanConnect proxy:

   – If necessary, start the MC-CmdHandler for the remote proxy.

   – In the proxy's navigation tree, click the nodes **Advanced Features** - **Diagnosis** - **Configuration** - **Proxy Container Log4j**.

   The Management Console displays the configuration in the **Logging Configuration** panel.

Figure 66: Configuring the loggers (example of BeanConnect proxy)



Proxy1 / test-en-lp: Logging Configuration

```
log4j.properties.xml
    debug=false
    Appenders
        BeanConnectLoggingFile
        BeanConnectManagementConsole
        BeanConnectMCSocketAppender
        BeanConnectSysout
        BeanConnectSysoutShort
    Loggers
        Root Logger (FATAL)
        BeanConnect (ERROR)
        BeanConnect.c (-->ERROR)
        BeanConnect.Datasources.OLTP (-->ERROR)
        BeanConnect.info (INFO)
        BeanConnect.kdcs (-->ERROR)
        de.siemens (TRACE)
        net.fsc (ERROR)
        net.fsc.be
        net.fsc.tpb
    Structure
```

The loggers are displayed in a tree structure. All the existing loggers are listed under the **Loggers** node. The identifier of a logger consists of its name and the level assigned to it in parentheses. If the logger inherited the level from its parent, the level is indicated by an arrow (-->).

2. To change the configuration, right-click a logger or one of its subnodes and choose the appropriate command from the context menu.

You will find detailed information on the panels and the commands of the context menus in the Management Console's online help system.

13.3.1.2 Configuring appenders

Use the Management Console to adapt the configuration of an appender:

1. Open the Log4j configuration file as follows:

   BeanConnect resource adapter:

   – In the resource adapter's navigation tree, click the node **Log4j Configuration**.

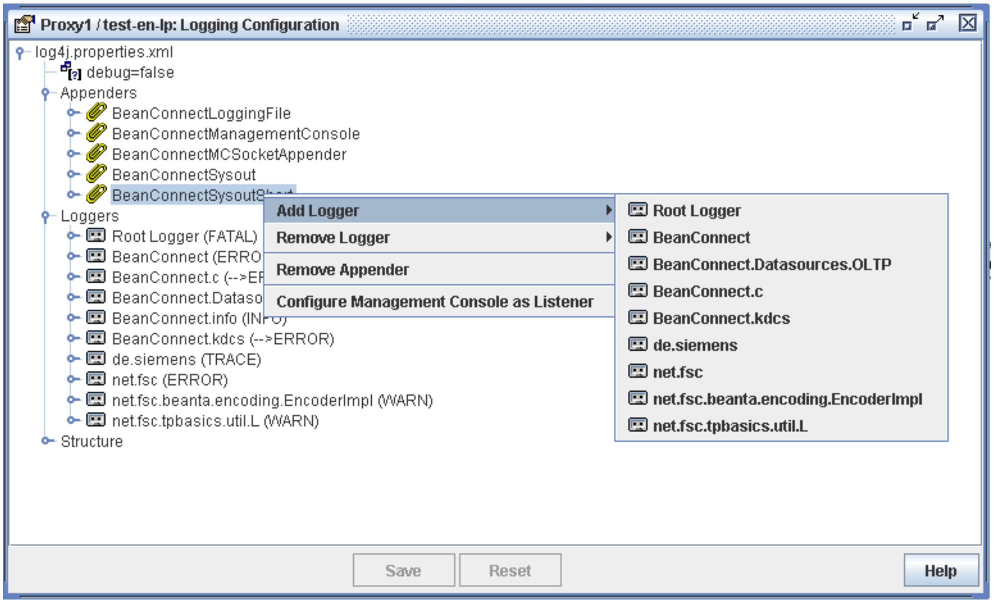
   BeanConnect proxy:

   – If necessary, start the MC-CmdHandler for the remote proxy.

   – In the proxy's navigation tree, click the nodes **Advanced Features** - **Diagnosis** - **Configuration** - **Proxy Container Log4j**.

   The Management Console displays the configuration in the **Logging Configuration** panel.

Figure 67: Configuring the appenders (example of BeanConnect proxy)



The appenders are displayed in a tree structure. All the existing appenders are listed under the **Appenders** node.

2. To change the configuration, right-click an appender or one of its subnodes and choose the appropriate command from the context menu.

   If you want to add a logger to the appender, for example, choose the **Add Logger** command from the appender's context menu. The list displays all the loggers that can still be added to the appender.

You will find detailed information on the panels and the commands of the context menus in the Management Console's online help system.

13.3.2 Editing the Log4j configuration file using the BeanConnect Management Console

The Management Console allows you to edit any Log4j configuration file in XML format:

1. Choose the **Extras - Edit Log4j Configuration File** command in the Management Console.

   The **Choose Log4j Configuration File (XML)** dialog box appears.

2. In this dialog box, select a configuration file in XML format.

   The Management Console displays the configuration contained in this file in the **Edit Log4j Configuration File** panel. You edit any configuration file in the same way as the Log4j configuration file of a resource adapter or a proxy (see Section 13.3.1.1, "Configuring loggers" and Section 13.3.1.2, "Configuring appenders").

You will find detailed information on the panels and commands of the context menus in the Management Console's online help system.

13.3.3 Configuring the BeanConnect Management Console as a Log4j socket reader

The Management Console can function as a Log4j socket reader for a resource adapter or a proxy. Consequently, logging output can be sent directly to a Management Console window (see Section 13.3.4, "Displaying the logging events in the BeanConnect Management Console".

To configure the Management Console for this, proceed as follows:

1. Choose the **Configure Management Console as Listener** command from the context menu of any node in the **Logging Configuration** panel.
2. If the Management Console is to wait for logging events at the specified listener port right from the start of any subsequent session, select the **Automatically Begin to Listen** option on the **General** tab of the **General Diagnostic Info Configuration** panel.
3. BeanConnect resource adapter:
   Set the appender `BeanConnectMCSocketAppender` as required in the resource adapter file `BeanConnect.log4j.properties.xml`:
   - The host on which the Management Console is running must be specified in the `RemoteHost` parameter.
   - In the `Port` parameter, you must specify the listener port at which the Management Console waits for logging events from the proxy container.

i The Management Console provides a list of all the logging listeners. This can be accessed via the **Log4j Logging Listeners** node at the topmost level of the navigation tree.

For the proxy, configuration involves the following steps:

1. If a socket listener port has not yet been entered in the properties of the relevant proxy for this case, the Management Console first asks for this port.

i The port is a local listener port and must therefore be different from all the other local listener ports already used on the Management Console's host. However, the Management Console can only check whether the ports entered are unique by referring to its own data and is subject to the same constraint when enforcing uniqueness. Listener ports used by other applications on the host cannot be checked by the Management Console. The user has to check that they are unique.

2. After the port has been entered, the Management Console checks whether the following appenders exist in the logging configuration of the proxy:

   - an async appender with the name `BeanConnectManagementConsole`
   - a socket appender with the name `BeanConnectMCSocketAppender`

   If these appenders are missing, they are created by the Management Console. If they exist, certain parameters of the appenders are checked and set appropriately.

3. The Management Console assigns the `BeanConnectManagementConsole` appender to the loggers `BeanConnect` and `de.siemens` (if they exist). Consequently, logging events are output to the Management Console via these two loggers or via loggers that inherit the appenders from these two loggers.

4. The Management Console activates logging for the current session and waits for logging events at the specified listener port.

Changing a listener port

To change the socket listener port subsequently, proceed as follows:

- Click **Advanced Features** - **Diagnosis** - **Configuration** - **General Diagnostic Info** in the navigation tree of the proxy and enter the **MC Logging Listener Port**.

i This change does not take effect until the proxy is started again or saved and reloaded (**Save/Restart** - **Save & Restart** command in the context menu of the proxy).

13.3.4 Displaying the logging events in the BeanConnect Management Console

If you have configured the Management Console appropriately (see Section 13.3.3, "Configuring the BeanConnect Management Console as a Log4j socket reader"), it receives the logging events output to the socket appender provided on the relevant resource adapter or proxy.

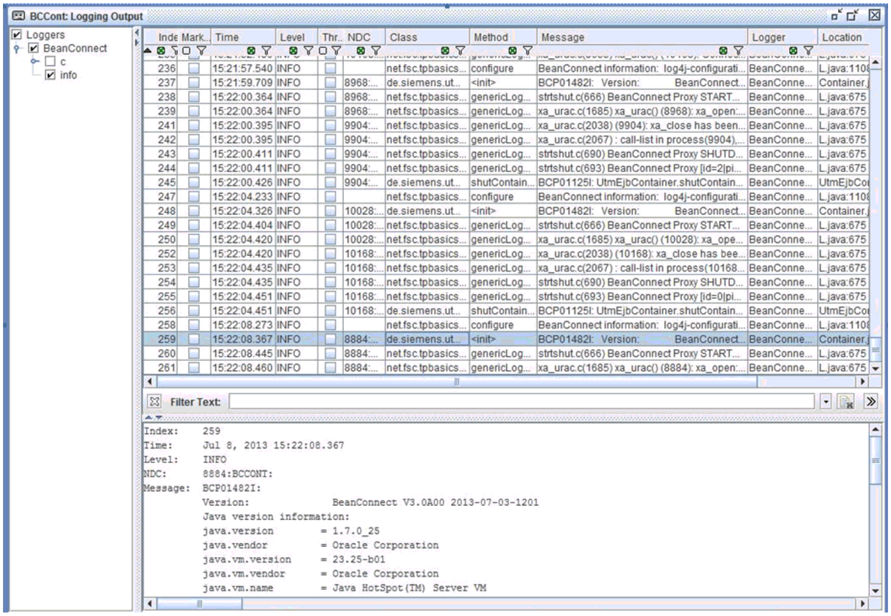
To display the logging events on the Management Console:

- Start the relevant proxy (see Section 8.1.1, "Starting a proxy").
- Check whether the **Automatically Begin to Listen** option in the **General Diagnostic Info Configuration** panel is set. Only if this option is selected does the Management Console wait for logging events at the specified listener port right from the start.

  Alternatively, you can also use the **Start Listening** command in the context menu of the **Proxy Container Log4j** node beneath the **Configuration** node.
- Click **Advanced Features** - **Diagnosis** - **Output** - **Proxy Container Log4j** in the navigation tree of the proxy.

  The Management Console displays the messages received in the **Logging Output** panel.

Figure 68: Logging output



Inde	Mark..	Time	Level	Thr..	NDC	Class	Method	Message	Logger	Location
236	□	15:21:57.540	INFO	□		net.fsc.tpbasics...	configure	BeanConnect information: log4j-configurati...	BeanConne...	L.java:110
237	□	15:21:59.709	INFO	□	8968:...	de.siemens.ut...	<init>	BCP01482I: Version: BeanConnect...	BeanConne...	Container.
238	□	15:22:00.364	INFO	□	8968:...	net.fsc.tpbasics...	genericLog...	strtshut.c(666) BeanConnect Proxy START...	BeanConne...	L.java:675
239	□	15:22:00.364	INFO	□	8968:...	net.fsc.tpbasics...	genericLog...	xa_urac.c(1685) xa_urac() (8968): xa_open:...	BeanConne...	L.java:675
241	□	15:22:00.395	INFO	□	9904:...	net.fsc.tpbasics...	genericLog...	xa_urac.c(2038) (9904): xa_close has been...	BeanConne...	L.java:675
242	□	15:22:00.395	INFO	□	9904:...	net.fsc.tpbasics...	genericLog...	xa_urac.c(2067) : call-list in process(9904),...	BeanConne...	L.java:675
243	□	15:22:00.411	INFO	□	9904:...	net.fsc.tpbasics...	genericLog...	strtshut.c(690) BeanConnect Proxy SHUTD...	BeanConne...	L.java:675
244	□	15:22:00.411	INFO	□	9904:...	net.fsc.tpbasics...	genericLog...	strtshut.c(693) BeanConnect Proxy [id=2|pi...	BeanConne...	L.java:675
245	□	15:22:00.426	INFO	□	9904:...	de.siemens.ut...	shutContain...	BCP01125I: UtmEjbContainer.shutContain...	BeanConne...	UtmEjbCo
247	□	15:22:04.233	INFO	□		net.fsc.tpbasics...	configure	BeanConnect information: log4j-configurati...	BeanConne...	L.java:110
248	□	15:22:04.326	INFO	□	10028:...	de.siemens.ut...	<init>	BCP01482I: Version: BeanConnect...	BeanConne...	Container.
249	□	15:22:04.404	INFO	□	10028:...	net.fsc.tpbasics...	genericLog...	strtshut.c(666) BeanConnect Proxy START...	BeanConne...	L.java:675
250	□	15:22:04.420	INFO	□	10028:...	net.fsc.tpbasics...	genericLog...	xa_urac.c(1685) xa_urac() (10028): xa_ope...	BeanConne...	L.java:675
252	□	15:22:04.420	INFO	□	10168:...	net.fsc.tpbasics...	genericLog...	xa_urac.c(2038) (10168): xa_close has bee...	BeanConne...	L.java:675
253	□	15:22:04.435	INFO	□	10168:...	net.fsc.tpbasics...	genericLog...	xa_urac.c(2067) : call-list in process(10168...	BeanConne...	L.java:675
254	□	15:22:04.435	INFO	□	10168:...	net.fsc.tpbasics...	genericLog...	strtshut.c(690) BeanConnect Proxy SHUTD...	BeanConne...	L.java:675
255	□	15:22:04.451	INFO	□	10168:...	net.fsc.tpbasics...	genericLog...	strtshut.c(693) BeanConnect Proxy [id=0|pi...	BeanConne...	L.java:675
256	□	15:22:04.451	INFO	□	10168:...	de.siemens.ut...	shutContain...	BCP01125I: UtmEjbContainer.shutContain...	BeanConne...	UtmEjbCo
258	□	15:22:08.273	INFO	□		net.fsc.tpbasics...	configure	BeanConnect information: log4j-configurati...	BeanConne...	L.java:110
259	□	15:22:08.367	INFO	□	8884:...	de.siemens.ut...	<init>	BCP01482I: Version: BeanConnect...	BeanConne...	Container.
260	□	15:22:08.445	INFO	□	8884:...	net.fsc.tpbasics...	genericLog...	strtshut.c(666) BeanConnect Proxy START...	BeanConne...	L.java:675
261	□	15:22:08.460	INFO	□	8884:...	net.fsc.tpbasics...	genericLog...	xa_urac.c(1685) xa_urac() (8884): xa_open:...	BeanConne...	L.java:675

```
Index:    259
Time:     Jul 8, 2013 15:22:08.367
Level:    INFO
NDC:      8884:BCCONT:
Message:  BCP01482I:
          Version:             BeanConnect V3.0A00 2013-07-03-1201
          Java version information:
          java.version        = 1.7.0_25
          java.vendor         = Oracle Corporation
          java.vm.version     = 23.25-b01
          java.vm.vendor      = Oracle Corporation
          java.vm.name        = Java HotSpot(TM) Server VM
```

The **Logging Output** panel consists of the following areas:

- **Logger Tree**: Displays all the loggers for which there are messages in the Management Console. Loggers whose paths are shown in red in the structure have at least one message with the logging level WARN.
- **Logging Event List**: Displays those logging events available in the Management Console that match the current settings in the **Logging Output** panel. Messages that appear in red have at least the logging level WARN.

  When you select an event from the event list, it is displayed in detail in the lower part of the event list.

You will find detailed information on the panels and commands of the context menus in the Management Console's online help system.

Changing the layout of the panel

By default, the logger tree is displayed on the left of the panel and the message list on the right. You can change this setting so that the logger tree and message list are each displayed on a separate tab:

1. Click the **Properties** button in the **Logging Output** panel.
   The **Logging Properties** dialog box appears.
2. Select the display mode under **Display Mode**.
   - **split**
     The logger tree and logging event list are displayed on the left and right of the panel respectively.
   - **tabbed**
     The logger tree and logging event list are displayed on two separate tabs.

Activating/deactivating the display of the loggers

The event list only displays the events of loggers that are activated in the event list.

To activate or deactivate the display for a logger:

Click the relevant logger node or its check box. This node, all the nodes in the path of the node up to the node logger and all the associated child nodes are activated/deactivated.

Changing the display of messages

Events are displayed in the event list in accordance with the following criteria:

- Only messages of the loggers that are activated in the logger tree (see the section described above).
- Only messages that are not currently suppressed.

  The Clear button in the Logging Output panel consigns to the background all the logging events of the relevant proxy that currently exist in the Management Console. The messages are then no longer included in the output in the panel although they still exist in the Management Console. You can display them again by clicking the **Consider All** button.
- Only those messages whose logging level corresponds at least to the minimum logging level set for the list.

  You set the logging level for the message list either by means of the buttons in the **Logging Output** panel or by means of the **Minimum Displayed Logging Level** option in the **Logging Properties** dialog box.
- If a **Filter Mask** is entered in the **Logging Properties** dialog box, only the messages that contain the filter text in at least one column (a distinction is drawn between uppercase and lowercase).

13.3.5 Display the Log4j logging file using the BeanConnect Management Console

The Management Console allows you to display Log4j logging files in XML format offline.

A logging file in XML format is created, for example, when you have added an XML file appender to a predefined file appender (**Add XML File Appender** command in the context menu of the relevant file appender).

To display the logging file created:

1. Choose the **Extras - Open Log4j Logging File (XMC)** command in the Management Console.

   The **Choose a Log4j Logging File (XMC)** dialog box appears.

2. Select a logging file in XML format in this dialog box. By default, a logging file in XML format has file name extension `.xmc`.

   The Management Console displays the logging file offline in the **Log4j Logging File** panel. The panel is structured in the same way as the **Logging Output** panel, in which messages are output online in the Management Console (see Section 13.3.4, "Displaying the logging events in the BeanConnect Management Console").

13.4 LogWriter for connection factories

The application server provides LogWriters to which the BeanConnect resource adapter writes system level information on managed connection factories and managed connections when certain events occur. This information is intended for the application server administrator and is not intended for the diagnosis of problems in application programs.

Events and event classes

The events are subdivided into the following classes which comprise specific events:

- Errors
- Transactions:
  - Begin of a transaction for a connection
  - Commit/Rollback of a transaction for a connection
- Lifecycle:
  - Generation of a managed connection
  - Request for a connection handle for a managed connection
  - Switch of a connection handle for a managed connection
  - Release of a connection handle and inclusion of the managed connection in the application server's connection pool
  - Removal of a managed connection from the application server's connection pool
  - Release of a managed connection
  - Application exceptions for a connection handle that are thrown to an application
  - System exceptions that are thrown for a managed connection

Configuring a LogWriter in the application server

In the case of Oracle WebLogic Server, you configure the LogWriter for a managed connection factory in the file `weblogic-ra.xml` as follows:

- Logging level

  The logging level determines the granularity with which BeanConnect outputs messages on a managed connection factory at the LogWriter. In the case of the Oracle WebLogic Server, you configure the logging level in the file `weblogic-ra.xml` by specifying the property `logLevel`. This value can be set separately for each managed connection factory. There are four log levels: NONE, ERROR, INFO and ALL.

  For details, see Section 4.3, "Setting configuration properties for outbound communication via OSI-TP / LU6.2" and Section 4.4.1.3, "Setting the configuration properties for UPIC". You will find examples in Example 6, "Configuration properties in the file weblogic-ra.xml" and Example 7, "Configuration properties in the file weblogic-ra.xml".

- Logging attributes

  You set further logging attributes in the subelement `<logging>`. You can find further details on the subelement `<logging>` in the schema description for the file `weblogic-ra.xml` in the Oracle Weblogic Server documentation.

  Enter the path name of the file for the logging output in the attribute `<log-filename>`.

  You can enable or disable logging using the attribute `<logging-enabled>`

  You can use other attributes to control, for example, the size to which a log file can grow or the maximum number of log files that can be created for a managed connection factory if `file rotation` is configured.

  Example

```
<logging>
    <log-filename>C:/temp/log/BeanConnect/echo.log</log-filename>
    <logging-enabled>true</logging-enabled>
    <rotation-type>bySize</rotation-type>
    <number-of-files-limited>true</number-of-files-limited>
    <file-count>3</file-count>
</logging>
```

  If you are dealing with different connection factories then you should specify different files. Otherwise, conflicts may occur when writing the files and the log records may be truncated.

Example

```
<connection-instance>
   <jndi-name>eis/beanconnect_oltp_echo</jndi-name>
      <connection-properties>
         <logging>
         <log-filename>C:/temp/log/BeanConnect/echo.log</log-filename>
            <logging-enabled>true</logging-enabled>
            <rotation-type>bySize</rotation-type>
            <number-of-files-limited>true</number-of-files-limited>
            <file-count>3</file-count>
         </logging>
         <properties>
            <property>
               <name>ConnectionURL</name>
               <value>oltp://echo</value>
            </property>
            <property>
                <name>displayName</name>
                <value>sample application/echo</value>
            </property>
            <property>
                <name>logLevel</name>
                <value>ALL</value>
            </property>
         </properties>
      </connection-properties>
    </connection-instance>
```

Format of the logging records

All the records that BeanConnect writes to the LogWriter have the following structure:

```
BeanConnect:<date-time> <identifier> message
```

`<date-time>` Specifies the date and time on which the record was written.
Format (example): 2013-03-20 08:30:26.810+0100.

`<identifier>` Specifies the identifier.

In the case of a managed connection factory, this is the value configured in the `DisplayName` property of the file `weblogic-ra.xml`, see Section 4.3, "Setting configuration properties for outbound communication via OSI-TP / LU6.2" and Section 4.4.1.3, "Setting the configuration properties for UPIC".

In the case of a managed connection, the identifier has the form `BCUnnnnn`, where each n stands for a digit.

In the case of a connection handle, the identifier has the form `BCUnnnnn.i`, where each n stands for a digit and i stands for a number.

`message` Message output by the resource adapter

Example 24 Entries in the LogWriter file

1. In the case of lifecycle events, the date, time and identifier of the managed connection are logged:

```
BeanConnect:2013-03-20 08:30:51.225+0100 <sample application/echo>: Managed connection
with id <BCU00002> destroyed
```

2. In the case of events which refer to an exception, the exception is also logged:

```
BeanConnect:2013-03-20 08:33:35.198+0100 <sample application/echo>: rcvString():
Exception thrown for connection <BCU00003.2>:
net.fsc.jca.communication.EISConnectionException:
net.fsc.jca.communication.EISConnectionException: exceptionShortageOfResources:
shortage of resources (40Z,KD10): no connection to partner; partner:
(SMPOSICL,gssbwrit), Dialog, error code: undefined error code [EC_UNDEFINED:0],
connectionId: , error code: undefined error code [EC_UNDEFINED:0], connectionId:
BCU00003.2, proxy: MYPROXY:30004/BCU30004, userId: BCU00003; diagnostic string:
```

3. In the case of communications with transactions, six logging records are usually generated:

```
BeanConnect:2013-03-20 08:30:27.138+0100 <sample application/echo>: Managed connection
with id <BCU00002> taken from pool
BeanConnect:2013-03-20 08:30:27.138+0100 <sample application/echo>: Connection handle
with id <BCU00002.1> created
BeanConnect:2013-03-20 08:30:27.653+0100 <sample application/echo>: Transaction
started for managed connection "BCU00002" with xid: formatID=48801, gtrid=002157A9
D15A3057 A4BD, bqual=6569732F 6265616E 636F6E6E 6563745F 6F6C7470 5F656368 6F
BeanConnect:2013-03-20 08:30:43.815+0100 <sample application/echo>: Transaction
committed for managed connection "BCU00002" with xid: formatID=48801, gtrid=002157A9
D15A3057 A4BD, bqual=6569732F 6265616E 636F6E6E 6563745F 6F6C7470 5F656368 6F
BeanConnect:2013-03-20 08:30:43.908+0100 <sample application/echo>: Connection handle
with id <BCU00002.1> released
BeanConnect:2013-03-20 08:30:44.267+0100 <sample application/echo>: Managed connection
with id <BCU00002> returned for pooling
```

4. In the case of communication without transactions, four logging records are usually generated:

```
BeanConnect:2013-03-20 08:50:41.117+0100 <sample application/echo>: Managed connection
with id <BCU00005> taken from pool
BeanConnect:2013-03-20 08:50:41.117+0100 <sample application/echo>: Connection handle
with id <BCU00005.4> created
BeanConnect:2013-03-20 08:50:53.753+0100 <sample application/echo>: Connection handle
with id <BCU00005.4> released
BeanConnect:2013-03-20 08:50:54.112+0100 <sample application/echo>: Managed connection
with id <BCU00005> returned for pooling
```

13.5 Diagnosis of the BeanConnect resource adapter

This section provides information on the following topics:

- Overview of logging in the BeanConnect resource adapter
- Predefined logging configuration of a resource adapter
- Logging of user interface calls

13.5.1 Overview of logging in the BeanConnect resource adapter

This section provides information on logging in the BeanConnect resource adapter. The logging facilities use Log4j by default. The scope of logging is controlled by configuration files. The following configuration files are delivered with the BeanConnect resource adapter installation:

- the file `config/BeanConnect.log4j.properties.xml`

  basic logging information (default)

- The file `config/BeanConnect.log4j.properties_default.xml`

  corresponds to the file `config/BeanConnect.log4j.properties.xml`

- the file `config/BeanConnect.log4j.properties_debug.xml`

  detailed logging information

- the file `config/BeanConnect.log4j.properties_error.xml`

  only the information on errors is logged

You can use one of these configuration files depending on your requirements.

You will find detailed information on controlling the depth of logging manually in Section 13.1, "Logging with Log4j" and Section 13.5, "Diagnosis of the BeanConnect resource adapter".

Default logging in the resource adapter

- After the resource adapter has been deployed, basic logging is activated automatically.
- The logging files are saved in the logging directory of the application server (for Oracle WebLogic Server in `<WebLogicServerDomainDirectory>/servers/<ServerName>/logs`).
- The following logging files are created:
  - `BeanConnect.logging.txt` (information for the user)
  - `BeanConnect.extlogging.txt` (logging data which you must send to system service if an error occurs.)
  - `BeanConnect.extlogging.txt.xmc` (additional logging data which you must send to the system service if an error occurs.)

Adapting logging in the resource adapter

Copy the configuration files present in the resource adapter installation to the application server's configuration directory.
For Oracle WebLogic Server, the directory is
`<WebLogicServerDomainDirectory>/config`.

- To extend logging (comprehensive runtime logging):
  Rename the copied file
  `config/BeanConnect.log4j.properties_debug.xm` to
  `BeanConnect.log4j.properties.xml`.
- To restrict logging (error logging only)
  Rename the copied file
  `config/BeanConnect.log4j.properties_error.xml` to
  `BeanConnect.log4j.properties.xml`
- Application-specific logging control for the resource adapter
  Adapt the file `config/BeanConnect.log4j.properties.xml` to your requirements as described below (see Section 13.1, "Logging with Log4j" and Section 13.5, "Diagnosis of the BeanConnect resource adapter") and rename it to
  `BeanConnect.log4j.properties.xml`.

13.5.2 Predefined logging configuration of a resource adapter

Logging in the resource adapter is preconfigured with the following default values in the file `BeanConnect.log4j.properties.xml` (see Section 13.3.1, "Configuring logging for BeanConnect resource adapter and proxy").

Appender

`BeanConnectSysoutShort`	**Description**: Console appender with the destination `System.out` and “short” output format, i.e. without a specification of the message source. At this appender, BeanConnect outputs messages on the resource adapter configuration as well as warnings and error messages. **Recommendation**: No further loggers should be assigned to this appender.

`BeanConnectShortLoggingFile`

Description:
Rolling file appender for logging to a file with several backup files (see Section 13.1.1.3, "How the rolling file appender works") and "short" output.

The same information is output to this appender that is sent to `BeanConnectSysoutShort`.

At this appender, BeanConnect outputs messages on the resource adapter configuration as well as warnings and error messages

This appender has three parameters:

- `File`

  (Relative) file name of the current logging file

  Default: `log/BeanConnect.logging.txt`

- `MaxNbrBackupFiles`

  Maximum number of backup files. If you specify 0 here, no backup files are created.

  Default: 10

- `MaxFileSizePerProcessKB`

  The maximum size in KB that the output file is allowed to reach before being rolled over to backup files.

  Default: 1024

Recommendation:
No further loggers should be assigned to this appender. However, the user may increase the level for the `BeanConnect.ui` logger if necessary (i.e. from WARN to INFO, DEBUG or TRACE).

`BeanConnectLoggingFile`	**Description**: Rolling file appender for logging to a file with several backup files (see Section 13.1.1.3, "How the rolling file appender works") and, in contrast to `BeanConnectShortLoggingFile`, more output with technical details. This appender is used for system diagnosis. If necessary, BeanConnect writes detailed diagnostic log information to this appender. This appender has three parameters: ● `File` (Relative) file name of the current logging file Default: `log/BeanConnect.extlogging.txt` ● `MaxNbrBackupFiles` Maximum number of backup files. If you specify 0 here, no backup files are created. Default: 10 ● `MaxFileSizePerProcessKB` The maximum size in KB that the output file is allowed to reach before being rolled over to backup files. Default: 1024 **Recommendation**: No further loggers may be added to this appender. If necessary, users should be able to increase the level for the `BeanConnect.ui` logger (i.e. from WARN to INFO, DEBUG or TRACE, see Section 13.5.3, "Logging of user interface calls") on their own initiative, i.e. without consulting the system diagnostics.

`BeanConnectLoggingFileXML`

Description:
Rolling file appender for logging to a file with several backup files (see Section 13.1.1.3, "How the rolling file appender works"). The output of this appender has an XML-like layout. The file is provided for input to the Management Console (see also Section 13.3.5, "Display the Log4j logging file using the BeanConnect Management Console").

The same information is output to this appender that is sent to the appender `BeanConnectLoggingFile`. This appender is used for system diagnosis. If necessary, BeanConnect writes detailed diagnostic log information to this appender.

This appender has three parameters:

- `File`

  (Relative) file name of the current logging file

  Default:
  `log/BeanConnect.extlogging.txt.xmc`

- `MaxNbrBackupFiles`

  Maximum number of backup files. If you specify 0 here, no backup files are created.

  Default: 10

- `MaxFileSizePerProcessKB`

  The maximum size in KB that the output file is allowed to reach before being rolled over to backup files.

  Default: 1024

  Recommendation:
  No further loggers may be added to this appender. If necessary, users should be able to increase the level for the `BeanConnect.ui` logger (i.e. from WARN to INFO, DEBUG or TRACE, see Section 13.5.3, "Logging of user interface calls") on their own initiative, i.e. without consulting the system diagnostics.

`BeanConnectMCSocketAppender`	**Description**: Auxiliary appender used to log to the Management Console. This appender has four parameters: ● `RemoteHost` Host where the Management Console is running. Default: `localhost` ● `Port` Listener port of the Management Console. Default: 31015 ● `LocationInfo` Always true. ● `ReconnectionDelay` A positive integer representing the number of milliseconds between each failed connection attempt to the server. Default: 10000 **Recommendation** No further loggers should be added to this appender.
`BeanConnectManagementConsole`	**Description**: Appender for logging to the Management Console. **Recommendation**: Loggers that are to be displayed at the Management Console should be added to this appender.

Loggers

You are recommended not to change the logger settings - with the exception of the logger `BeanConnect.ui`.

`BeanConnect`	Parent logger of all the other BeanConnect loggers Level: `WARN`
`BeanConnect.in`	Logger for BeanConnect inbound communication.
`BeanConnect.info`	Logger for runtime information Level: `INFO`
`BeanConnect.out`	Logger for BeanConnect outbound communication.
`BeanConnect.ui`	Logger for the BeanConnect user interface calls Level: `WARN` In the default configuration, the output of the logger `BeanConnect.ui` is sent to the appenders `BeanConnectShortLoggingFile`, `BeanConnectLoggingFile`, `BeanConnectLoggingFileXML`.
`de.siemens` `net.fsc`	Parent loggers of all class-specific BeanConnect loggers that serve to output debug traces Level: `WARN`
`net.fsc.beanta.encoding`	Logger for BeanConnect's encoding support Level: `WARN`
`net.fsc.tpbasics.util.L`	Logger for supporting Log4j

13.5.3 Logging of user interface calls

Application (EJB) calls to BeanConnect are logged at separate loggers (`BeanConnect.ui` and its child loggers). This output can assist users in diagnosing problems in their applications.

Output to the logger `BeanConnect.ui` may use the levels `INFO`, `DEBUG` or `TRACE`. For a summary, see the following table:

List of the BeanConnect.ui loggers, their levels and meaning

Logger	Level	Meaning
`BeanConnect.ui.out`	`INFO`	Logging of outbound interface calls without data
`BeanConnect.ui.oltpmsg`	`INFO`	Logging of OLTP message interface calls without data
`BeanConnect.ui.data.api`	`DEBUG`	Logging of all data transferred at the user interface
`BeanConnect.ui.data.net`	`TRACE`	Logging of all data transferred at the user interface in the form in which it is transferred in the network, i.e. possibly recoded.
`BeanConnect.ui.in`	`INFO`	Logging of the OLTP message objects transferred to and from the message endpoint application

Example 25 Control of logging output

- Logger `BeanConnect.ui` set to level `INFO`:

  All the user interface calls are logged without data.

- Logger `BeanConnect.ui` set to level `DEBUG`:

  All the user interface calls are logged together with the data transferred on these calls.

- Logger `BeanConnect.ui` set to level `TRACE`:

  All the user interface calls are logged together with the data transferred on these calls and any encoded data.

- Logger `BeanConnect.ui.out` set to level `INFO` and `BeanConnect.ui.data.net` set to level `TRACE`:

  All the outbound interface calls are logged together with the encoded data. Calls at the OLTP message interface and data transferred at the outbound interface are not logged.

13.6 Diagnosis of the BeanConnect proxy container

There is a variety of information available for diagnosis of a BeanConnect proxy container. This information is distributed among a number of files in the container home directory on the basis of functionality.

This section provides information on the following topics:

- Predefined logging configuration of a proxy
- Log files of the BeanConnect proxy container
- Traces of the BeanConnect proxy container

13.6.1 Predefined logging configuration of a proxy

The logging of each proxy is preconfigured with the following default values after installation.

Appender

`BeanConnectSysoutShort`	**Description**: Console appender with the destination `System.out` and “short” output format, i.e. without any specification of the message source. BeanConnect outputs general messages as well as warnings and error messages at this appender **Recommendation**: No further loggers should be added to this appender.
`BeanConnectSysout`	Console appender with the destination `System.out` and detailed output format for making easier the system diagnosis. BeanConnect outputs fatal messages for which no other appender is configured to this appender. **Recommendation**: No further loggers should be added to this appender.

BeanConnectLoggingFile

Description:
Rolling file appender for logging to a file with several backup files (see the section Section 13.1.1.3, "How the rolling file appender works")

This appender is used for system diagnosis. BeanConnect writes detailed diagnostic logging information to this appender if necessary.

This appender has three parameters:

- File

  (Relative) file name of the current logging file

  Default: logs/logging.txt

- MaxNbrBackupFiles

  Maximum number of backup files. If you specify 0 here, no backup files are created.

  Default: 10

- MaxFileSizePerProcessKB

  Process-specific switchover threshold in KB. When the writing of a logging event causes the file size to exceed this process-specific threshold, the file is switched. The size of a logging file at switchover is thus somewhere between the values for <MaxFileSizePerProcessKB> and <number_of_processes>* <MaxFileSizePerProcessKB>.

  Default: 500

Recommendation:
No further loggers should be added to this appender.

BeanConnectLoggingFileXML

Description:
Rolling file appender for logging to a file with multiple backup files (see Section 13.1.1.3, "How the rolling file appender works"). The output from this appender has an XML-like layout. The file can be evaluated at the Management Console (see also Section 13.3.5, "Display the Log4j logging file using the BeanConnect Management Console").

Although this appender is preconfigured, it is not used in the standard BeanConnect configuration. It is used for system diagnostics. If necessary, BeanConnect writes detailed diagnostic logging information to this appender.

Recommendation:
If required or if so requested by the system service, this appender should be assigned the same loggers as the appender BeanConnectLoggingFile.

Loggers

It is recommended not to change the logger settings.

Root-Logger	Level: FATAL.
BeanConnect	Parent logger of all the other BeanConnect loggers Level: ERROR
BeanConnect.c	Logger for debug traces from the C components of BeanConnect Level: ERROR
BeanConnect.info	Logger for runtime information Level: INFO
BeanConnect.Datasources.OLTP	Logger for OLTP data sources Level: ERROR
BeanConnect.kdcs	Logger for debug traces of the KDCS calls from the proxy container application. Level: ERROR
de.siemens and net.fsc	Parent loggers of all class-specific BeanConnect loggers that serve to output debug traces. Level: ERROR
net.fsc.tpbasics.util.L	Logger for supporting Log4j Level: WARN
net.fsc.beanta.encoding .EncoderImpl	BeanConnect's logger for encoding support Level: WARN

13.6.2 Log files of the BeanConnect proxy container

The proxy container is based on openUTM. There are several logging and diagnosis files with information from openUTM. This section provides information on the following topics:

- stdout/stderr log
- System log file SYSLOG
- Dumps and diagnostic dumps
- Application log under Windows

13.6.2.1 stdout/stderr log

Messages from the proxy container to `stdout/stderr` are logged to files in the proxy container home directory. The files are available by default. The files `utmp.err.<suffix>` and `utmp.out.<suffix>` are created for the first time when the proxy container is started and the suffix YY-MM-DD.HHMMSS corresponds to the start time.

Switching the log file

At runtime, the proxy container application is set in such a way that `stdout/stderr` output is switched to new files every day at midnight. The suffix `YY-MM-DD.HHMMSS` added to these files corresponds to the switchover time.

In addition, you can also switch over the `stdout/stderr` files manually using the Management Console. To do this, choose the **Switch Protocol Files** command in the proxy or proxy cluster's context menu.

Displaying the log files

You can display the content of the files as follows:

- In the Management Console, choose the nodes **Advanced Features** - **Diagnosis** - **Output** - **General Diagnostic Info** under the proxy container's node. In the case of a proxy cluster, you must also select the proxy.

  Select the entry **Container STDERR Diagnostics** or **Container STDOUT Diagnostics** Then click the **Show File button.**
  The Management Console then lists all the `utmp.out.*` or `utmp.err.*` files for selection. Select the required files and click the **OK** button. The Management Console transfers the selected files to the local diagnostic path of the proxy and displays them in a **Text File** panel.

  For details, please refer to the Management Console's online help system.

- You can also view and evaluate these logs using a standard editor directly in the proxy container's home directory.
- Under Windows you can also use the commands **Advanced** - **Show STDERR Diagnostics** or **Advanced** - **Show STDOUT Diagnostics** in the program group of the proxy container to display the `utmp.err` or `utmp.out` file respectively.

Backing up the log files

The files are backed up to the proxy container directory `out-err` the next time the proxy container is started. Before the backup, all the files in the directory `out-err` are deleted. If you want to use these files for diagnostic purposes, you must therefore back them up before restarting the proxy.

If the proxy container is executed as a service on Windows systems (see section "Starting the proxy container as a Windows service" on page 259), its first output to `stdout` is written to the file `utmp.out` and its first output to `stderr` is written to the file `utmp.err`.
If the log is switched, the files have the names:
`utmp.err.<suffix>` and `utmp.out.<suffix>`.

The logs from the last application run are automatically backed up in the directory `out-err`.

13.6.2.2 System log file SYSLOG

The system log file records important events (in the form of binary messages) from the proxy container run. It contains important information which can be used to diagnose errors. It is stored in the file directory `SYSLOG` in the proxy container home directory. The file is available by default.

It can be displayed as follows:

- In the Management Console, choose the nodes **Advanced Features** - **Diagnosis** - **Output** - **General Diagnostic Info** under the proxy container's node and select the entry **Container Syslog Files**. Then click the **Show File** button.

  If there is more than one file of the selected type, the **Select Diagnostic File To Show** dialog box opens showing the appropriate files on the host. Select the diagnostic file you want to show.
  The Management Console transfers the converted text file `syslog.<number>.txt` to the local diagnostic path of the proxy and displays the file in a **Text File** panel.

  For details, please refer to the Management Console's online help system.

- On Solaris and Linux systems, change to the proxy container home directory and call the script as follows:

  ```
  shsc/syslog.sh
  ```

  The available log files are listed and you are asked whether you want to display each file in sequence. If you select a file, the formatted information is written to the `slogout` file in the proxy container home directory and displayed in the system's default editor.

- On Windows systems (cmdline), choose the command **Advanced** - **Show Syslog** in the program group of the proxy container.

  A command line prompt is opened in which the available log files are listed. Select the number of the file you want to display. The formatted information is then written to the `slog.out` file in the proxy container home directory and displayed in the system's default editor.

- On Windows systems, change to the proxy container home directory and call the script as follows:

  ```
  shsc\syslog.cmd
  ```

  The formatted information is then written to the `slog.out` file in the proxy container home directory and displayed in the system's default editor.

13.6.2.3 Dumps and diagnostic dumps

If the proxy container crashes or fatal errors occur while the proxy container is running, an openUTM dump is generated and stored in the subdirectory `DUMP` of the proxy container home directory.

ℹ Please note that these dumps should be analyzed primarily by BeanConnect specialists.

13.6.2.4 Application log under Windows

If the proxy container is started as a service and problems occur during startup, the application log may provide additional information (see the section "Starting the proxy container as a Windows service" on page 259):

1. Choose **Start - Settings - Control Panel - Administrative Tools - Event Viewer**.
2. In the event viewer, click **Application Log**, and then select the source **openUTM**.

13.6.3 Traces of the BeanConnect proxy container

This section provides information on the following topics:

- OSS trace
- BCAM trace
- CMX trace

13.6.3.1 OSS trace

The function logs activities in the proxy container relating to OSI-TP connections to the openUTM partner application or to the openUTM-LU62 Gateway.

By default the OSS trace is deactivated. To activate the OSS trace of the proxy container in the Management Console:

1. Select the following nodes in the navigation tree of the proxy:
   Advanced Features - **Diagnosis** - **Configuration** - **General Diagnostic Info**.

   The **General** tab of the **General Diagnostic Info Configuration** panel is displayed.

2. Select the option **Activate OSS Trace**.

If the proxy container is running, the change comes into effect dynamically when saving the proxy. If the proxy container is not running, the change comes to effect when the proxy is next started.

The traces are written in a binary format to `OSST.*` files in the proxy container home directory. You can evaluate and display the trace information:

- In the Management Console, choose the nodes
  Advanced Features - **Diagnosis** - **Output** - **General Diagnostic Info** in the navigation tree of the proxy and select the entry **Container OSS Traces**. Then click the **Show File** button.

  The Management Console transfers the converted text file `osstrac.txt` to the local diagnostic path of the proxy (default: `diag/<proxy_cont_name>` in the Management Console home directory) and displays the file in a **Text File** panel.

- On Solaris and Linux systems, change to the proxy container home directory and call the script as follows:

  ```
  shsc/ositrace.sh
  ```

  The traces produced are then written to the `osstrac.txt` file in the proxy container home directory and displayed in the system's default editor.

- On Windows systems, change to the proxy container home directory and call the script as follows:

  ```
  shsc\ositrace.cmd
  ```

  The traces produced are then written to the `osstrac.txt` file in the proxy container home directory and displayed in the system's default editor.

For further diagnosis, you have to send the evaluated trace to the diagnostic system service.

13.6.3.2 BCAM trace

The function logs all connection-oriented activities within the proxy container.

By default, the BCAM trace is deactivated. To activate the BCAM trace of the proxy container in the Management Console:

1. Select the following nodes in the navigation tree of the proxy:
   Advanced Features - **Diagnosis** - **Configuration** - **General Diagnostic Info**.

   The **General** tab of the **General Diagnostic Info Configuration** panel is displayed.

2. Select the option **Activate BCAM Trace**.

If the proxy container is running, the change comes into effect dynamically when saving the proxy. If the proxy is not running, the change comes to effect when the proxy is next started.

The traces are written in a binary format to `KDCBTRC.*` files in the proxy container home directory. You can evaluate and display the trace information:

- In the Management Console choose the nodes **Advanced Features** - **Diagnosis** - **Output** - **General Diagnostic Info** and select the entry **Container BCAM Traces**. Then click the **Show File** button.

  The Management Console transfers the converted text file `btrc.txt` to the local diagnostic path of the proxy
  (default: `diag/<proxy_cont_name>` in the Management Console home directory). The Management Console displays the file in a **Text File** panel.

- On Solaris and Linux systems, change to the proxy container home directory and call the script as follows:

  ```
  shsc/nettrace.sh
  ```

  The traces produced are then written to the `btrc.txt` file in the proxy container home directory and displayed in the system's default editor.

- On Windows systems, change to the proxy container home directory and call the script as follows:

  ```
  shsc\nettrace.cmd
  ```

  The traces produced are then written to the `btrc.txt` file in the proxy container home directory and displayed in the system's default editor.

For further diagnosis, you have to send the evaluated trace to the diagnostic system service.

13.6.3.3 CMX trace

This function logs transport layer activities in the proxy container relating to connections to the openUTM partner application or to the openUTM-LU62 Gateway.

By default the CMX trace is deactivated. To activate the CMX trace of the proxy container in the Management Console:

1. Select the following nodes in the navigation tree of the proxy:
   Advanced Features - **Diagnosis** - **Configuration** - **General Diagnostic Info**.

   The **General** tab of the **General Diagnostic Info Configuration** panel is displayed.

2. Select the option **Activate CMX Trace**.

The change comes into effect when the proxy is next started.

Solaris and Linux systems

The CMX traces of the proxy container are written in a binary format in `CMX*` files in the `cmxt` subdirectory of the proxy container home directory.

Windows systems

The CMX traces of the proxy container are written in binary format to `<number>.CMX` files. These files are processed in the configured trace path of CMX. You can specify the trace path by using the PCMX-32 tool **Trace Control**. Choose the command **Options** - **Trace Path** and specify the path. To start **Trace Control**, select the command **Trace Control** from the PCMX-32 program group. The PCMX-32 program group is available after installing PCMX-32.

Displaying the CMX traces in the BeanConnect Management Console

You can evaluate and display the CMX trace information in the Management Console.

In the Management Console, choose the nodes **Advanced Features** - **Diagnosis** - **Output** - **General Diagnostic Info** under the proxy's node and select the entry **Container CMX Traces**. Then click the **Show File** button.

The Management Console transfers the converted text file with the appended suffix `.txt` to the local diagnostic path of the proxy (default: `diag/<proxy_cont_name>` in the Management Console home directory). The Management Console displays the file in a **Text File** panel.

For further diagnosis, you have to send the evaluated trace to the diagnostic system service.

13.7 Diagnosis of the BeanConnect Management Console

The Management Console uses Log4j to output its own logging messages and debug traces.

The traces are written in the file `logging.txt` in the subdirectory `logs` of the Management Console. The files are always available by default. The content of the files can be displayed using any text editor.

The output traces are controlled by the file `log4j.properties.xml` in the `config` subdirectory of the Management Console.

There are two special loggers for the MC-CLI command line interface:

- `name="com.fujitsu.ts"` for the Java classes of the MC-CLI and
- `name="mccli"` for the Jython modules of the MC-CLI

You will find detailed information on the trace mechanism of the Management Console in the Management Console's online help system.

13.8 Diagnosing the BeanConnect tools

MC-CmdHandler

By default, logging for the MC-CmdHandler is performed using Log4j. The scope of logging is controlled via configuration files. The following configuration files are present in the BeanConnect resource adapter installation:

- The file `mccmdhandler.log4j.properties.xml`

  Basic logging data (default)

- The file `mccmdhandler.log4j.properties_debug.xml`

  Detailed logging data

13.9 Diagnosis of the openUTM-LU62 Gateway

This section provides information on the following topics:

- Traces and logs of the openUTM-LU62 Gateway
- Diagnosis information for the openUTM-LU62 Gateway

13.9.1 Traces and logs of the openUTM-LU62 Gateway

The openUTM-LU62 Gateway writes messages into log files and optionally provides trace information for diagnostics functionality. There are two different kinds of traces:

- Instance traces
- XAP-TP traces

You can define different trace levels to configure the extent of the instance traces.

13.9.1.1 Activate/deactivate traces

By default, the instance traces and the XAP-TP traces are deactivated. You can activate the traces for the openUTM-LU62 Gateway using the Management Console.

Select the openUTM-LU62 Gateway under the item **openUTM-LU62 Gateways** in the navigation tree or **openUTM-LU62 Gateways** in the proxy's navigation tree and choose **Edit Properties** in the context menu. The table **Edit Properties of openUTM-LU62 Gateway** lists the defined values which you are now able to edit.

- Trace Level
  Specifies the level of instance traces logged by the openUTM-LU62 Gateway.
- Activate XAP-TP Trace
  Select this option to specify that the openUTM-LU62 Gateway is to log XAP-TP traces.The function logs the activities of the components XAP-TP provider and OSS with respect to connections to the proxy container.

If the openUTM-LU62 Gateway is running, the trace settings come into effect dynamically when saving the openUTM-LU62 Gateway.

If the traces are activated, they are written in the following files:

- On Solaris and Linux systems:
  - instance trace file
    `/opt/lib/utmlu62/PROT/inlog.<lu_name>.<suff>`
  - XAP-TP trace file
    ```
    /opt/lib/utmlu62/PROT/xaplog.
                         <lu_name>.<suff1>.<suff2>
    ```
- On Windows systems:
  - instance trace file
    `<gateway_home>\PROT\inlog.<lu_name>.<suff>`
  - XAP-TP trace file
    `<gateway_home>\PROT\xaplog.<lu_name>.<suff1>.<suff2>`

Here `<lu_name>` stands for a local LU alias name, `<suff>/<suff1>/<suff2>` are numerical suffixes and `<gateway_home>` indicates the directory where the openUTM-LU62 Gateway is installed.

13.9.1.2 Evaluating traces and logs

The traces of the openUTM-LU62 Gateway are written in binary format.

You can convert and display all the openUTM-LU62 Gateway traces and log files in the Management Console:

1. Choose the following nodes in the navigation tree of the proxy:
   Advanced Features - **Diagnosis** - **Output** - **General Diagnostic Info**
2. Select one of the following entries:
   - **LU62 Gateway Instance Traces**
   - **LU62 Gateway Instance Protocol Flow**
   - **LU62 Gateway XAP-TP Traces**
   - **LU62 Gateway Protocol Files**
3. Click the **Show File** button.

The Management Console transfers the converted text file to the local diagnostic path and displays the file in a **Text File** panel.

Two files result from the conversion of a binary instance trace:

- generated instance trace
- instance log flow

In addition to the trace files the openUTM-LU62 Gateway writes messages into log files:

- On Solaris and Linux systems:

  `/opt/lib/utmlu62/PROT/prot.<lu_name>`

- On Windows systems:

  `<gateway_home>\PROT\prot.<lu_name>.txt`

Here `<lu_name>` stands for a local LU alias name and `<gateway_home>` indicates the directory where the openUTM-LU62 Gateway is installed.

The names of the log files and the generated traces in the local diagnostic path are as follows:

- instance traces: `inlog.*.txt`
- instance protocol flow: `inlog.*.flow.txt`
- XAP-TP traces: `xaplog.*.txt`
- log files: `prot.*.txt`

Instance protocol flow

In the instance protocol flow files you will find an abbreviated description of the protocol flow (LU6.2 and OSI-TP protocol).

The openUTM-LU62 Gateway uses the APPC interface for communication via the LU6.2 protocol and the XAP-TP interface for communication via the OSI-TP protocol.

On the side of the LU6.2 protocol, the following information is displayed in the protocol flow for every message:

- name of the APPC call
- TP-ID which is assigned by the SNAP-IX or the IBM Communications Server
- direction of processing
- additional parameters

The direction of processing is indicated by an arrow. An arrow pointing to the left indicates a message sent by the openUTM-LU62 Gateway and an arrow pointing to the right indicates a message received by the openUTM-LU62 Gateway.

The administration data is exchanged between the openUTM-LU62 Gateway and the LU6.2 partner while the openUTM-LU62 Gateway is starting up or if a connection error occurs. The transaction code X'06F2' is used for this purpose. These protocol flows are indicated by a single arrow (-->). A protocol flow of an application program is indicated by a double arrow (==>).

Each message contains a correlation number in order to make it easier to associate an LU6.2 conversation and a parallel connection via XAP-TP. Protocol flows that are not associated with a conversation are assigned the correlation number zero.

Additionally, every message is output with the time and the corresponding line number in the original output file. The protocol flow does not contain user data. This user data can only be found in the original output file.

13.9.2 Diagnosis information for the openUTM-LU62 Gateway

The following information is required when diagnosing errors:

- The status of the openUTM-LU62 Gateway. Check this using the Management Console. Select the **Check Availability** command from the context menu of the openUTM-LU62 Gateway.

  The openUTM-LU62 Gateway and its availability are displayed in a table. If the openUTM-LU62 Gateway is available, you can display detailed information by double-clicking on this entry or by using the button **Result Details**.

  The following information is relevant:

  - **LLU-NAME**: Alias name of the local LU via which the EIS partner is identified. It consists of
    - the value specified for the EIS partner in the Management Console in the **Prefix** field on the **General** tab of the **Edit Properties of EIS Partner** property sheet
    - another generated name part to ensure that names are unique.
  - **atot**: The number of parallel connections established between the proxy container and the openUTM-LU62 Gateway.
  - **stot**: The number of established connections (sessions) between the openUTM-LU62 Gateway and the CICS application.

  The number of control sessions is also output. If the number of control sessions is 0, then a configuration error has occurred. If **atot** indicates the value 0, then a configuration error has occurred or the proxy container has not been started. If **stot** indicates the value 0, then a configuration error has occurred or the communication service has not been started or the EIS partner is not running. If **stot** indicates the value 2, then a configuration error has occurred and the specified mode is not known in VTAM on the z/OS.

- The description of the error situation.

- All available diagnosis files:
  - On Solaris and Linux systems:
    - instance trace files
      `/opt/lib/utmlu62/PROT/inlog.<lu_name>.<suff>`
    - XAP-TP trace files
      `/opt/lib/utmlu62/PROT/xaplog.<lu_name>.<suff1>.<suff2>`
    - log files
      `/opt/lib/utmlu62/PROT/prot.<lu_name>`
  - On Windows systems:
    - instance trace files
      `<gateway_home>\PROT\inlog.<lu_name>.<suff>`
    - XAP-TP trace files
      `<gateway_home>\PROT\xaplog.<lu_name>.<suff1>.<suff2>`
    - log files
      `<gateway_home>\PROT\prot.<lu_name>.txt`

  Here `<lu_name>` stands for a local LU alias name, `<suff>, <suff1> and <suff2>` are numerical suffixes and `<gateway_home>` indicates the directory where the openUTM-LU62 Gateway is installed.

⚠ If you restart an openUTM-LU62 Gateway, the following diagnostic files in the subdirectory `<gateway_home>/PROT` are deleted:

- `in.dump.<lu_name>`
- `xaplog.<lu_name>.*`
- `xap.dump.<lu_name>.*`
- `prot.<lu_name>.old`
- `prot.<lu_name>.*.old`
- `core.<lu_name>`

This means that you must save the diagnostic files before restarting the openUTM-LU62 Gateway.

The files `prot.<lu_name>` and `inlog.<lu_name>.*` are saved with the suffix `.old`. On Windows systems, the file `prot.<lu_name>` has the additional suffix `.txt`.

13.10 Diagnosis of SNAP-IX for Solaris systems

Logging files containing different types of messages and several trace options are provided for diagnosing SNAP-IX problems.

Messages in log files

SNAP-IX differentiates between three types of messages in log files:

- Problem

  Messages with the type `Problem` indicate a serious and unexpected event and are always logged.

- Exception

  Messages with the type `Exception` indicate events which degrade system performance or which will cause problems or degrade performance in the future.

- Audit

  Messages with the type `Audit` indicate normal events during the SNAP-IX run.

SNAP-IX traces

SNAP-IX provides several trace options for diagnosing SNAP-IX-specific problems (Line Tracing, API Tracing, Client-Server-Tracing, TN Server Tracing and Internal Tracing).

13.10.1 Diagnosis with the Management Console

You can configure logging and traces for SNAP-IX and evaluate and display message logs and trace files using the Management Console.

Configuring logging and traces

Select a communication service under the item **Communications Services** in the navigation tree or in **Communication Service** in the proxy's navigation tree and then choose **Edit Properties** in the communication service's (in this case SNAP-IX) context menu. The table **Edit Properties of Communication Service Instance** lists the defined values which you are now able to edit.

Check the appropriate options to activate or deactivate logging and traces. Additionally, you can switch on more detailed versions of audit logging (with the **Verbose Audits** option) and of problem and exception logging (with the **Verbose Errors** option).

If SNAP-IX is running, the change comes into effect when you save the Communication Service.

Evaluating logging and traces

SNAP-IX writes the following files to the directory `/var/opt/sna/`:

- Audit messages to the logging file `sna.aud`.
- Error messages (problem and exception logging) to the logging file `sna.err`.
- Line traces in binary form to the trace files `sna1.trc` and `sna2.trc`.

To display these files, choose the following nodes in the navigation tree of the proxy: **Advanced Features** - **Diagnosis** - **Output** - **General Diagnostic Info**. In the table, select one of the entries **SNAP-IX Audit Log**, **SNAP-IX Error Log** or **SNAP-IX Line Trace**. Then click the **Show File** button.

The Management Console converts a selected trace file to a text file and transfers the converted text file or the message log file to the local diagnostic path and displays the file in a **Text File** panel.

For detailed information, please refer to the SNAP-IX documentation.

13.11 Diagnosis of the IBM Communications Server for Linux

Logging files containing different types of messages and several trace options are provided for diagnosing IBM Communications Server problems.

Messages in Log Files

The IBM Communications Server differentiates between three types of messages in log files:

- Problem

  Messages with the type `Problem` indicate serious and unexpected events and are always logged.

- Exception

  Messages with the type `Exception` indicate events which degrade system performance or which will cause problems or degrade performance in the future.

- Audit

  Messages with the type `Audit` indicate normal events during the IBM Communications Server run.

IBM Communications Server traces

The IBM Communications Server provides several trace options for diagnosing IBM-Communications-Server-specific problems (Line Tracing, API Tracing, Client-Server-Tracing, TN Server Tracing and Internal Tracing).

13.11.1 Diagnosis with the Management Console

You can configure logging and traces for the IBM Communications Server and evaluate and display message logs and trace files using the Management Console.

Configuring logging and traces

Select a communication service under the item **Communications Services** in the navigation tree or in **Communication Service** in the proxy's navigation tree and then choose **Edit Properties** in the communication services (in this case IBM Communications Server for Linux) context menu. The table **Edit Properties of Communication Service Instance** lists the defined values which you are now able to edit.

Check the appropriate options to activate or deactivate logging and traces. Additionally you can switch on more detailed versions of audit logging (with the **Verbose Audits** option) and of problem and exception logging (with the **Verbose Errors** option).

If the IBM Communications Server is running, the change comes into effect when you save the Communication Service.

Evaluating logging and traces

The IBM Communications Server writes the following files to the directory `/var/opt/ibm/sna/`:

- Audit messages to the logging file `sna.aud`.
- Error messages (problem and exception logging) to the logging file `sna.err`.
- Line traces in binary form to the trace files `sna1.trc` and `sna2.trc`.

To display these files, choose the following nodes in the navigation tree of the proxy: **Advanced Features** - **Diagnosis** - **Output** - **General Diagnostic Info**. In the table, select one of the entries **Communications Server (Linux) Audit Log**, **Communications Server (Linux) Error Log** or **Communications Server (Linux) Line Trace**. Then click the **Show File** button.

The Management Console converts a selected trace file to a text file and transfers the converted text file or the message log file to the local diagnostic path and displays the file in a **Text File** panel.

For detailed information, please refer to the IBM Communications Server documentation.

13.12 Diagnosis of the IBM Communications Server for Windows systems

On Windows systems, the IBM Communications Server provides a **Log Viewer** and a **Trace Facility** for diagnosing problems. These tools have graphical user interfaces. You can start the tools from the relevant program groups.

For detailed information, please refer to the IBM Communications Server documentation.

13.12.1 Diagnosis with the Management Console

You can evaluate and display the message log and the trace log files of the IBM Communications Server using the Management Console.

Configuring logging and traces
Select a communication service under the item **Communications Services** in the navigation tree or in **Communication Service** in the proxy's navigation tree and then choose **Edit Properties** in the communication service's (in this case IBM Communications Server for Linux) context menu. The table **Edit Properties of Communication Service Instance** lists the defined values which you are now able to edit.

Check the appropriate options to activate or deactivate logging and traces. Additionally you can switch on more detailed versions of audit logging (with the **Verbose Audits** option) and of problem and exception logging (with the **Verbose Errors** option).

If the IBM Communications Server is running, the change comes into effect when you save the Communication Service.

Evaluating logging and traces
To display these files, choose the following nodes in the navigation tree of the proxy: **Advanced Features** - **Diagnosis** - **Output** - **General Diagnostic Info**. In the table, select one of the entries **Communications Server (Windows) Message Log** or **Communications Server (Windows) Trace Log.** Then click the **Show File** button.

The Management Console converts a selected trace file to a text file and transfers the converted text file or the message log file to the local diagnostic path and displays the file in a **Text File** panel.

13.13 Collecting diagnostic information

BeanConnect provides support for gathering all available diagnostic information in the proxy with a single mouse click.

In the Management Console, select the command **Advanced Features - Diagnosis - Output - General Diagnostic Info** from the navigation tree of the proxy. Then click the **Get All Files** button or select the **Get All Files** command from the context menu of any table entry.

A **Select File** dialog box is opened where you can specify the target directory for the traces and log files. In this dialog box, a subdirectory of the configured local diagnostic path is proposed. The name of the subdirectory is built from the current date and time (`<local-diag-path>/<date-time>`).

After selection of the target directory, the Management Console starts to collect all available files with diagnostic information for the BeanConnect proxy and for the proxy components. If necessary, the files are converted from binary format to text format and then the files are copied into the target directory. An action dialog box is displayed which provides information about the progress and the outcome of the action.

13.14 Error messages of the BeanConnect proxy container

This section provides information on the following topics:

- Configuration error messages
- Runtime error messages

13.14.1 Configuration error messages

openUTM, on which the BeanConnect proxy container is based, is configured using the configuration tool KDCDEF. The workflow and input to this tool are controlled by the Management Console. KDCDEF normally works without any manual intervention by the user.

If you intervene in the configuration process and, for example, manipulate the KDCDEF input files, successful configuration can no longer be guaranteed.

Each KDCDEF run is logged with messages to the `kdcdef.out` file in the `def` subdirectory of the proxy container home directory. You will find error messages in this file if they are available.

The configuration was successful if the process terminated with the message

- K450 KDCFILE generated; KAA size: `&KAASIZE` K

If errors are detected in the input files or another internal error occurs, the process is terminated with one of the following messages:

- K448 KDCFILE generated with warnings; KAA size: `&KAASIZE` K
- K449 There was at least one ERROR. Generation aborted.

Please inform system service if you cannot identify any connection with manual manipulation of the input files with respect to the preceding error messages.

13.14.2 Runtime error messages

The messages generally indicate problems between the resource adapter and the proxy container or between the proxy container and the EIS partner or openUTM-LU62 Gateway. User errors or internal problems in BeanConnect or in the openUTM-LU62 Gateway could be the reasons for these problems.

13.14.2.1 Types of messages

In BeanConnect, there are three groups of messages:

Group 1	Messages which log normal behavior.
Group 2	Messages which log problems and errors. You can respond to these messages.Any notes or actions are described in this document.
Group 3	Internal messages from BeanConnect.

This chapter contains all messages of groups 2 and 3 which can be displayed at runtime of the proxy container in alphabetical order. Messages of group 1 are not described.

The runtime messages are logged in the files `utmp.out.<suffix>` and `utmp.err.<suffix>` where `<suffix>` indicates the date and time stamp. The files are saved in the proxy container home directory.

Each message is preceded by an individual ID. A "&" character precedes the name of an insert. The description of a message provides the meaning of those inserts needed by the BeanConnect user. All other inserts are needed by system service for diagnosis. Some inserts contain information on error codes during file processing (DMS error codes) or on error codes of the system. These inserts are described in the section Section 13.16.2, "System error codes".

Meaning of the openUTM-specific terms in the BeanConnect proxy container which are used in the messages:

DMS	File access
UTM	openUTM component
UTM-D	Component for distributed communication
XAP-TP	Component for the OSI-TP protocol stack

Please check the following issues before contacting system service:

- Does the configuration of the proxy container in the Management Console correspond to the configuration of the resource adapter, e.g. the values for **proxyURL** or **inboundListenerPort** (see the configuration of resource adapter in Section 4.2.1, "Defining general properties in ra.xml" and the configuration in the Management Console in Section 6.5, "Configuring the BeanConnect resource adapter")?
- Does the configuration of the proxy container in the Management Console correspond to the configuration of the EIS partner or openUTM-LU62 Gateway, e.g. the values for **Host** and **Port**?
- Have the proxy container, all proxy components and the EIS partner been started and are they available? You can check availability with the Management Console (see Section 8.6, "Checking the availability of BeanConnect proxies").

The following lists contain the messages which can be issued by BeanConnect. Additional information has been added to the descriptions to explain the actions (responses) to the messages:

- K messages
- P messages
- U messages

K messages and U messages are output by default.

P messages only occur during the communication via the OSI-TP protocol, i.e. during communication between the proxy container and the EIS partner or between the proxy container and openUTM-LU62 Gateway.

If a K message is output, you should take into account the corresponding P or U message where appropriate.

13.14.2.2 K messages

BCSYSEX K009 Transaction code &TAC **is invalid** (&RCDC) **- input please**
Invalid service name called by an EIS using a UPIC, Socket or RFC1006 connection.
Action: The table below lists the possible error codes together with error causes and possible error recovery actions. If an error message occurs that is not described in the list below, inform system service.

Error code &RCDC

&RCDC	Meaning
KM01	The service has not been generated. Action: Configure an inbound service and assign this service name to the inbound message endpoint or change the client program.

BCSYSEX K017 Service &TCVG **terminated by openUTM (**&RCCC/&RCDC &RCF2A**) - input please**
An inbound transaction was rolled back.

Action: Normal behavior if &RCCC is 70Z and &RCDC is K306. Otherwise inform system service.

K036 Connection setup: &PTRM/&PRNM/&BCAP/&LTRM &RSLT, &REA1
BeanConnect outputs the message when the connection from the resource adapter to the proxy container is set up.

&RSLT	Meaning
Y	Connection set up.
N	Connection was not set up; the cause is given in &REA1.

&REA1	Meaning
X' 00'	Connection already set up.
X' 0A'	BeanConnect proxy container shut down. Action: Start the BeanConnect proxy container.
X' 0C'	Connection cleardown being executed. Action: Repeat the request.
X' 12'	No further free entry available in terminal pool. Action: Depends on the insert &LTERM.

&REA1	Meaning
X' 1B'	The IP address of the EIS partner could not be determined Action: Check EIS partner host and /or inform network administration
X' 2E'	The connection has not yet been completely cleared down. Action: Repeat the request.

&LTERM	Meaning
BCRA	Action: Increase the **Number of Parallel Outbound Connections** in the **Performance Settings** tab of the **Edit Properties of Proxy** property sheet in the Management Console.
BCUP	Action: Increase the **Number of Parallel Inbound UPIC Connections** in the **Performance Settings** tab of the **Edit Properties of Proxy** property sheet in the Management Console.
BCSO	Action: Increase the **Number of Parallel Inbound Socket Connections** in the **Performance Settings** tab of the **Edit Properties of Proxy** property sheet in the Management Console.

K040 Warning level `&WLEV` for &PGPOOL no longer exceeded
This message is only output if message K041 was previously output and means that the measures recommended there are currently no longer necessary.

K041 Warning level `&WLEV` for &PGPOOL exceeded
Action: Increase the **Proxy Container Storage Area Size (Page Pool)** in the **Performance Settings** tab of the **Edit Properties of Proxy** property sheet using the Management Console and carry out the todo topics which are shown in the todo topic list of the Management Console.

K043 DMS error `&DMSE` for file `&FNAM`
The DMS error code is output in insert `&DMSE`.

Action: The possible DMS error codes are described in Section 13.16, "Error codes". Alternatively inform system service.

K049 Error = &RCCC2 during application startup

The BeanConnect proxy container issues message K049 whenever the start of a container task is aborted due to an error. The error code &RCCC2 shows the cause of the error. Action: The table below lists the possible error codes together with error causes and possible error recovery actions. If an error message occurs that is not described in the list below, inform system service.

Error code &RCCC2

Code	Error cause	Action
20	Shared memory cannot be allocated by the first task of the BeanConnect proxy container due to insufficient address space.	Check system generation.
21	Shared memory cannot be allocated by the first task of the BeanConnect proxy container due to insufficient address space or due to the proxy container being terminated.	As for 20 or normal behavior.
22	File access error.	See DMS error code, Section 13.16, "Error codes".
24	File access error.	See DMS error code, Section 13.16, "Error codes".
32	The task lock bourse could not be created. This error occurs when there are too few semaphore entries available for the BeanConnect processes. This can occur when the termination of a process and its restart overlap.	Start again with a larger number of semaphores (see the section Section 12.7, "Number of semaphores in the proxy container").
33	BeanConnect refuses to start more tasks for the proxy container because the proxy container has already terminated (normal or abnormal termination).	Normal behavior.
35	While restarting a proxy container task, BeanConnect detects that the proxy container is being aborted.	Normal behavior.
50	As for 20.	Check system generation.
55	DMS error with KDCA file.	See DMS error code, Section 13.16, "Error codes".
56	DMS error with page pool file.	See DMS error code, Section 13.16, "Error codes".
57	DMS error with restart file.	See DMS error code, Section 13.16, "Error codes".

Code	Error cause	Action
58	Error with the SYSLOG file of the BeanConnect proxy. Possible causes: The `<proxy_cont_home>/SYSLOG` directory is not correct.	Delete (if necessary) and regenerate the prepared SYSLOG file in the <proxy_cont_home> directory with following commands: ● Solaris and Linux: . ./initenv.sh $UTMPATH/ex/kdcslog . 20 ● Windows: initenv.cmd %UTMPATH%\ex\kdcslog . 20
59	Error when opening SYSLOG file.	See DMS error code, Section 13.16, "Error codes".
79	A BeanConnect proxy container task makes a request, but it runs out of memory.	Check system generation.
84	Insufficient memory.	Check system generation.
85	Insufficient memory.	Check system generation.
91	Error when starting BeanConnect proxy container. You can find a detailed description of the error under message K124.	See message K124 on page 529.

K055 Asynchronous service `&ATAC1` **terminated by openUTM; KCRCCC=**`&RCCC`**; KCRCDC=**`&RCDC`**;user=**`&USER`**; LTERM=**`&LTRM`

An asynchronous inbound transaction was rolled back and the asynchronous inbound message will be redelivered if necessary.

Action: Normal behavior when KCRCCC is 000 and KCRCDC is 0000 or KCRCCC is 70Z and KCRCDC is K306. Otherwise inform system service.

K060 Application run aborted; reason = &TRMA

BeanConnect creates a memory dump whenever a proxy container application is aborted or a dump requested. Such a dump is produced for each work process of the proxy container.

The **Group** column in the table below describes the reason group to which the dump error code (&TRMA) belongs. The following groups exist:

A	The cause is a user error, e.g. an error in ● generating and administering proxy container applications with the Management Console ● generating the system (e.g. division of the address space)
D	The dump was created for diagnostic purposes.
F	The dump is a continuation dump, another task has caused the BeanConnect proxy container to terminate abnormally.
M	The cause is a memory bottleneck.
U, S, X	The cause is an internal error in BeanConnect.

If an error message occurs that is not described in the list below, inform system service.

Code	Group	Reason
ALGxxx	ASU	Shared memory bottleneck. Action: Tune the system kernel.
ASIS99	D	BeanConnect proxy container was terminated abnormally by the administrator.
BRSREM	F	BeanConnect proxy container was terminated abnormally by the administrator.
CACHT1 through CACHT6	F	After a process has initiated the abnormal termination of the BeanConnect proxy container, another task of the BeanConnect proxy container has terminated abnormally itself (= continuation dump). Action: Depends on the reason for the abnormal termination of the proxy container.
DIAGDP	D	A diagnostic dump has been generated. The BeanConnect proxy container is running normally.
ENDE14	F	As for code CACHT1.

Code	Group	Reason
ENDPET	A	The BeanConnect proxy container cannot be terminated normally because distributed transactions still exist with the status PTC (prepare to commit). The transaction can be terminated after the BeanConnect proxy container has been restarted. Additionally, a connection to the EIS partners that are involved in the distributed transactions must exist. Action: Restart the BeanConnect proxy container.
FMMM10	A	An input message cannot be stored because the page pool is full. Action: Increase the **Proxy Container Storage Area Size (Page Pool)** in the **Performance Settings** tab of the **Edit Properties of Proxy** property sheet in the Management Console**.**
IOyxxx	ASU	An unrecoverable error has occurred during file processing, yxxx = DMS error code, see the section Section 13.16, "Error codes".
IPC035	A	Error on locking the IPC shared memory segment. This may be caused by the `remove` script being called or by a `Forced Clear` while the BeanConnect proxy container is still running.
IPC037	FU	After a process has initiated abnormal termination of the BeanConnect proxy container, another task of the BeanConnect proxy container has terminated abnormally itself (= continuation dump).
IPCEND	FU	After a process has initiated abnormal termination of the BeanConnect proxy container, another process of the BeanConnect proxy container has terminated abnormally itself (= continuation dump).
IPCREM	F	The BeanConnect proxy container was terminated abnormally with the `remove` script or `Forced Clear`.
ISLPT1 through ISLPT4	F	As for code CACHT1.
LATCT1	F	As for code CACHT1.
JVMABT		The JVM has initiated the abnormal termination of the BeanConnect proxy container, see also file `hs_err*.log` in the container directory
LCACT1	F	As for code CACHT1.
LKAA04	US	As for code CACHT1.
LKAAT1	F	As for code CACHT1.
LKLCT1 through LKLCT4	F	As for code CACHT1.
LPCMT1	F	As for code CACHT1.
OSAFT2	F	As for code CACHT1.

Code	Group	Reason
OSTM07	A	A log record for a distributed transaction cannot be backed up, since the page pool is full. Action: Increase the **Proxy Container Storage Area Size (Page Pool)** in the **Performance Settings** tab of the **Edit Properties of Proxy** property sheet in the Management Console.
PCMM05	AU	The page pool is full. Action: Increase the **Proxy Container Storage Area Size (Page Pool)** in the **Performance Settings** tab of the **Edit Properties of Proxy** property sheet in the Management Console.
PEND02	A	No further data can be written because the page pool is full. Action: Increase the **Proxy Container Storage Area Size (Page Pool)** in the **Performance Settings** tab of the **Edit Properties of Proxy** property sheet in the Management Console.
PENDER	AUD	A dump was created after abnormal termination of a transaction. Before this, the message K017 or K055 was output. The proxy container is still running normally.
PENDT1 PENDT2	F	As for code CACHT1.
PUTR01	AU	Error on writing to file. Possible reason: Disk storage bottleneck.
PWRT03	AU	Memory bottleneck. Action: Check memory requirements and operating system generation.
PWRT06	AU	As for code CACHT1.
RESTRT	D	The diagnostic dump is taken with a restart of the BeanConnect proxy container after an abnormal termination.
SACT28	M	Memory bottleneck. Action: Modify the value for `OSI-SCRATCH-AREA` in the proxy container and update the configuration files of the proxy container. Otherwise inform system service. For details see Chapter 12, "High availability and scalability".
SHM002	A	The shared memory key is not unique on the host. This may be an inherited error from K078 OSS 03. Action: Check the proxy container generation.The used shared memory keys are written to the <proxy_cont_home>/def/input.system.txt file. Parameter: MAX *SHMKEY. The shared memory keys depend on the generated port number. If necessary, you must generate the proxy container with another port number.

Code	**Group**	**Reason**
SLOG09	SU	Problem when writing the message buffer to the current SYSLOG file. Action: The DMS error code in the preceding K043 message may explain the cause of the error. Otherwise inform system service.
SLOG10	SU	An attempt by a continuation work process to switch over to the current SYSLOG file generation has failed. Action: It may be necessary to take into account any preceding K043 messages. Otherwise inform system service.
SMSG03	ASU	Problem when writing the message buffer to the current SYSLOG file. Action: The DMS error code in the preceding K043 message may explain the cause of the error. Otherwise inform system service.
STnnnn	ADSU	Error when processing the start of a proxy container task, where nnnn is the number indicating the error cause in the message "K049 Error nnnn during application startup". Processes of the BeanConnect proxy container that are still active continue running.
WAIT41	ASU	Problem when connecting to the network. See also the corresponding message U3xx of the BeanConnect proxy container. Action: Check the BeanConnect proxy container generation
WAITT1 WAITT2	F	As for code CACHT1.
XATT02	F	As for code CACHT1.
XFGE01	F	As for code CACHT1.

K065 Net message: `&PTRM/&PRNM/&BCAP/&LTRM &FIL1B &FIL2B`
Action: Depends on the values FIL1B and FIL2B.

The inserts &FIL1B and &FIL2B have the following meaning:

FIL1B	Meaning
X' F0' - X' FF'	Normal behavior.
other	See actions for FIL2B.

FIL2B	Meaning	Actions
00000008	Invalid parameter.	Inform system service.
00000014	Connection letter too long.	Inform system service.
00000030	Internal error.	Inform system service.
0400001C	Resource bottleneck.	Inform system service.
04000020	BeanConnect proxy container not signed on.	Inform system service.
30000020	Error signing on.	Inform system service.
other	Connection-specific events.	Normal behavior.

K075 Program exchange aborted by the task `&Process; &CTYP &PROG &PVER`
Action: Inform system service.

K078 ffffffff yyyyyyyyy

Meaning of the parameters:

ffffffff	Contains a short code for the error that has occurred (see table below).
yyyyyyyyy	Specific, context-related error message.

If an error code ffffffff occurs that is not described in the list below, inform system service.

ffffffff	Reason	Actions
ALME 01	Memory bottleneck on application start.	Check memory requirement, tune operating system.
ATEXIT 00 - 04	Process termination.	Information message.
DIAG 01 - 07	Information about the process environment.	Information message.
ENV 00 - 04	Information about the process environment.	Information message.
IPC 02	Memory bottleneck when creating the IPC shared memory.	Tune system kernel parameter.
IPC 03 - 07	Information about the dimension of the IPC shared memory.	Information message.
MEM 01	Memory bottleneck when starting the application. Full text: K078 MEM 01 in utmwork nn Bytes not available.	Check storage requirements; tune operating system.
MSG 01 - 02	Information about the process environment.	Information message.
NET 01 - 02	Information about the process environment.	Information message.
OSS 01	Information about the OSS shared memory.	Information message.
OSS 03	Error while loading the OSS shared memories into the address space.	Check storage requirements; tune operating system.
PIPE 01	Information about the process environment.	Information message.

ffffffff	Reason	Actions
SEM 01	Error while creating a semaphore. A semaphore is not unique on the host.	Check the proxy container generation. The semaphore keys depend on the generated port number. The used semaphore keys are written to the file `<proxy_cont_home>/def/input.system.txt` (`MAX SEMARRAY`). Change the first value in the brackets or you must generate the proxy container with another port number.
SIGNAL 01 - 02	Signal handling in utmwork process.	Information message.
SYSPROT 01 - 02	Information about the process environment.	Information message

K104 UTM-D TIMEOUT (&RCVDANNO): `&LSES, &LPAP, &AGUS`**; old state: (** `&OCVST, &OTAST` **); action:** `&ACTION`**; new state: (** `&NCVST, &NTAST` **).**

Reason: Timeout during communication between the proxy container and the EIS partner.

Action: Depends on the value of RCVDANNO.

ACTION	Meaning
ASYNCH	Timeout during asynchronous communication: Either the association occupancy timer or the reply timer has expired.
STPROG	Timeout during outbound dialog communication without any impact on the transaction.
COMMIT RESET	Timeout during outbound/inbound communication: The transaction is either committed (COMMIT) or rolled back (RESET).

RCVDANNO (Bytes 1-2)	Meaning
X'F331	A timer has expired during inbound communication after the sending of a response to the EIS. Action: Use the Management Console to increase the value of **Transaction Communication Timer** in the **Timer Settings** tab of the **Edit Properties of Proxy** property sheet.

RCVDANNO (Bytes 1-2)	Meaning
X'F332'	A timer has expired during outbound communication after all the EIS partners involved in the distributed transaction have been requested to initiate termination of the transaction. Action: Use the Management Console to check and, if necessary, increase the value for **Reply Timer** in the **Edit Properties of an Outbound Service** property sheet for all the outbound services involved in the transaction.
X'F332'	The ready timer has expired during inbound communication via OSI-TP. Action: Use the Management Console to increase the value of **Prepare to Commit Timer** in the **Timer Settings** tab of the **Edit Properties of Proxy** property sheet.
X'F400'	The OSI-TP association occupancy timer for a dialog job has expired during outbound communication.
X'F520'	The OSI-TP association occupancy timer for an asynchronous job (internal timer of 60 seconds) has expired during outbound communication.
X'F522'	The timer which monitors reception of acknowledgment of an asynchronous message sent via an OSI-TP association during outbound communication has expired.
X'F534'	The response timer has expired during outbound communication for a dialog job. Action: Use the Management Console to increase the value for **Reply Timer** in the outbound service's **Edit Properties of an Outbound Service** property sheet.

K119 OSI-TP error information: &OSLPAP, &USER, &TAC, &DIA1, &DIA2, &DIA3
The inserts &DIA1, &DIA2, &DIA3 contain the reasons for outputting message K119.

Reason with openUTM: The dialog with the EIS partner was terminated.

Reason with CICS: The dialog with the proxy component openUTM-LU62 Gateway was terminated.

Action: See the table below.

&DIA1	&DIA2	&DIA3	Meaning
01	06	02	No free connection available. Action: Increase the number of connections in the **General** tab of the **Edit Properties of an EIS Partner** property sheet in the Management Console.
02	02	01	Outbound communication: The outbound service is not known in the EIS. Action: Harmonize the partner service name specified in the outbound service with the service name generated in the EIS.
02	03	18	Outbound communication: The EIS partner has rejected auhentication with the specified security credentials. Action: Check the configuration or programming.
02	03	19	Outbound communication: The EIS partner has rejected the OSI-TP dialog because the partner LPAP which represents the proxy container in the EIS partner is locked or is in QUIET state.
02	03	21	Outbound communication: Memory bottleneck at the EIS partner.
02	03	22	Outbound communication: The outbound service is not known in the EIS partner or is locked or there is no authorization. Possible action: Harmonize the partner service name specified in the outbound service with the service name generated in the EIS.
02	03	23	Outbound communication: The outbound service is known in the EIS partner but cannot be called.
02	03	24	Outbound communication: The queue level for the asynchronous outbound service has been reached at the EIS partner.
02	03	25	Outbound communication: The dialog-based outbound service is configured as an asynchronous service in the EIS partner.
03	03	18	Inbound communication: The user employed by the EIS partner for the dialog is not configured in the proxy container or the employed password is false. Action: Check the configuration.

&DIA1	&DIA2	&DIA3	Meaning
03	03	21	Inbound communication: The proxy container storage area (page pool) is full. Action: In the Management Console, increase the value of **Pagepool** in the **Performance Settings** tab in **Proxy Container Storage Area Size** in the **Edit Properties of Proxy** property sheet i
04	02	00 01	The EIS partner has terminated the dialog. Action: See the messages in the EIS partner.
09 09	03 03	21 31	The proxy container storage area (page pool) is full. Action: In the Management Console, increase the value of **Pagepool** in **Proxy Container Storage Area Size** in the **Performance Settings** tab of the **Edit Properties of Proxy** property sheet
10	11 12 13 15 16	xx xx xx xx xx	No free connection available Action: In the Management Console, increase the number of connections (**Connections**) in the **General** tab of the **Edit Properties of an EIS Partner** property sheet.
10	17	xx	An inbound dialog process is active for every connection to the EIS partner and another inbound process is received on a connection that was closed and newly generated during this time. Action: Increase the reply timer in the **Edit Properties of an Outbound Service** property sheet in the Management Console. The specified value must be greater than the reply timer in the EIS partner
other values			Inform system service.

K124 Error: `&RCXAPTP` **at startup of XAP-TP occurred in phase:** `&PHAXAPTP`
Action:

For RCXAPTP = 106: Possible cause: Work process could not be started because proxy is already shut down

Otherwise: Inform system service.

K128 UTM-D job rejected: `&CON/&PRNM/&BCAP/&LPAP &LSES &REA1 &RCDC &TAC`
Meaning of the inserts with openUTM:

Insert	Meaning
&CON	OSI-CON name
&PRNM	Eight blanks
&BCAP	ACCESS-POINT name
&LTRM	OSI-LPAP name

The insert `&REA1` contains the reason that message K128 was output.

&REA1	Meaning	Action
X' 01'	Invalid service name called by an EIS using an OSI-TP connection.	If &RCDC=KM01: Configure an inbound message endpoint and assign this service name to the inbound message endpoint or change the OSI-TP client. Otherwise inform the system service.
X' 02'	Service name is generated with an error. - Recipient tpsu title in TP-BEGIN-DIALOGUE-RI	Inform system service.
X' 03'	An asynchronous service is to be started, the service which is assigned to the message endpoint is configured as a dialog-orientated service. - Receipt of a TP-END-DIALOGUE-RI protocol element	Check whether the OLTP message-driven bean implements the asynchronous interface and maybe enter the inbound message endpoint with the asynchronous service again. Or change the OSI-TP client.
X' 04'	A dialog-orientated service is to be started, the service which is assigned to the message endpoint is configured as an asynchronous service.	Check whether the OLTP message-driven bean implements the dialog-orientated interface and maybe enter the inbound message endpoint with the dialog-orientated service again. Or change the OSI-TP client.
X' 05'	An asynchronous service is to be started, but the message queue of the service has reached the generated threshold. - The connection is cleared down.	Inform system service.

K135 UPIC message: `&PTRM/&PRNM/&BCAP/&LTRM/&UPCREAS/&UPCSTAT/&UPCPROT/&UPVENC1/&UPPENC2`
Action if UPCREAS = 0D:
Increase the **proxy container storage area size** in the **Performance Settings** tab of the **Edit Properties of Proxy** property sheet in the Management Console.

Action if UPCREAS ≠0D: Inform system service.

K139 Switching SYSLOG failed! Still using file `&FNAM`
Action: It may be possible to ascertain the reason for the error that occurred on switchover from the DMS error code in the preceding message K043. Otherwise inform system service.

K147 Sign on for `&USRTYPE` **user** `&USER` **not successful** `&PTRm/&PRNM/&BCAP/&LTRM`
Reason: `&REA7`
Reason: Depends on the &USER value.

&USER	Reason
BCURAxxx	Outbound communication in a cluster configuration or in multiple resource adapter mode. If REA7=U3: Normal behavior Otherwise: Problems during communication between the resource adapter and the proxy container
BCUxxxxx	Outbound communication: Problems during communication between the resource adapter and the proxy container.
BCADMIN	Administration: Problems communicating with the proxy container in the Management Console.
other	Inbound communication: Problems during communication between the EIS partner and the proxy container.

Action: Depends on the &REA7 value.

&REA7	Meaning
U1	Inbound: The USER that is used for the dialog by the EIS partner is not configured in the proxy container. See Section 6.8, "Configuring inbound communication". Outbound: Internal error. Inform system service.
U3	Inbound: You have tried to start more than one parallel dialog with the same user and without commit functionality. Outbound: Internal error. Inform system service.

&REA7	Meaning
U4	Inbound: The password used for the dialog by the EIS partner is not configured in the proxy container. See Section 6.8, "Configuring inbound communication". Outbound: Internal error. Inform system service. Administration: An incorrect password is specified in the Management Console for access to the proxy container.
U17	The administrator has started termination of the proxy container. It is therefore not possible to start the dialog.
other	Inform system service.

K152 Heuristic report: `&COND &MTYPE &OLPAP &USER &LTAC &AAIS &AAID`
Action: Depends on the &COND value.

&COND	Meaning
MIX	Inbound: The application server and the EIS partner are not synchronized with respect to transaction security. Possible database inconsistency. To coordinate this, manual contact to one of the involved databases is required. Outbound: The EIS partner has detected a heuristic MIX.
HAZ	The application server has rolled back the transaction. The termination of this transaction (commit/rollback) is unknown in the EIS partner. Possible database inconsistency. To coordinate this, manual contact to one of the involved databases is required.

K160 The `&TACNTR.` transaction of the `&TCVG` service was reset by `&RCAUSER` (`&RCCC/&RCDC`); (pid: `&TASK`).
If `&TCVG` = KDCGIOPU, there is an outbound communication process.

For inbound communication `&TCVG` contains the service name that is assigned to the message endpoint in the Management Console.

Action with openUTM: Normal behavior, see the process in the resource adapter or EIS partner.

Action with CICS: Normal behavior, see the process in the resource adapter or proxy component openUTM-LU62 Gateway.

K204 XA (&PID**) Precommit requires global rollback; cause:** &XATXT **TA=** &INTTAID
Action: Inform system service if the value of the insert &XATXT is not equal to one of the following values.

&XATXT	Meaning
XA_RBROLLBACK	Rollback for unspecified reason
XA_RBCOMMFAIL	Rollback due to a internal communication error in the Resource Manager
XA_RBDEADLOCK	Rolback due to a deadlock
XA_RBINTEGRITY	Rollback due to a resource inconsistency
XA_RBOTHER	Rollback for unspecified reason
XA_RBPROTO	Rollback due to an internal protocol error in the Resource Manager
XA_RBTIMEOUT	Rollback due to transaction period timeout
XA_RBTRANSIENT	Rollback due to a temporary error

K210 XA (&PID**) Return code:** &XATXT **Open RM** &TEXT32, &INSTNUM
Action: Inform system service.

K211 XA (&PID**) Return code:** &XATXT **Close RM** &TEXT32, &INSTNUM
Action: Inform system service.

K212 XA (&PID**) xa_start(** &XAFLAG**) - Return code** &XATXT **TA=** &INTTAID
Action: Inform system service.

K213 XA (&PID**) xa_end(** &XAFLAG**) - Return code** &XATXT **TA=** &INTTAID
Action: Inform system service.

K214 XA (&PID**) xa_commit() - Return code** &XATXT **TA=** &INTTAID
Cause: Normal behavior if no abnormal termination of the proxy container follows.

Action: Inform system service on abnormal termination.

K215 XA (&PID**) xa_rollback() - Return code** &XATXT **TA=** &INTTAID
Cause: Normal behavior if no abnormal termination of the proxy container follows.

Action: Inform system service on abnormal termination.

K216 XA (&PID**) Return code** &XATXT**, recover PTC list, RM:** &TEXT32,&INSTNUM
Cause: Normal behavior if no abnormal termination of the proxy container follows.

Action:

- Abnormal termination during the start-up phase may be caused by a faulty connection to the resource adapter (e.g. incorrect port number or invalid protocol between the proxy container and the resource adapter). The cause is described in more detail in the messages in the log files `utmp.err.*` and `utmp.out.*` as well as in the proxy container's logging file. If an incorrect port number has been assigned to the resource adapter then this can be set correctly using the Management Console. It is possible to restart the proxy container.

  If the BeanConnect proxy container and the resource adapter have different versions, the resource adapter clears the connection and the proxy container is therefore terminated while still in the start-up phase. The comment on the incorrect version designation is logged in resource adapter logging.

- If the cause of abnormal termination during the start-up phase cannot be identified as described above, please inform the system service.

K217 XA (&PID**) xa_prepare() - Return code** &XATXT **TA=** &INTTAID
Cause: Normal behavior if no abnormal termination of the proxy container follows.

Action: Inform system service on abnormal termination.

K218 XA (&PID**) xa_forget() - Return code** &XATXT **TA=** &INTTAID
Cause: Normal behavior if no abnormal termination of the proxy container follows.

Action: Inform system service on abnormal termination.

K220 XA (&PID**) Error: xa_switch definition not found for specified RM:** &TEXT32
Action: Inform system service.

K221 XA (&PID**) Error: Start parameters not found for defined RM:** &TEXT32
Action: Inform system service.

K222 XA (&PID**) Error: Linked RM is not** &XASPEC **compatible** &TEXT32
Action: Inform system service.

K223 XA (&PID**) Syntax error in start parameters**
Action: Inform system service.

K224 XA (&PID**)** &XACALL **- return code** &XASTAT **from RM instance** &INSTNUM, &TEXT32 **is not XA(CAE) compliant**
Action: Inform system service.

K225 XA (&PID**) recursive call:** &XADBC1 **- Error/signal in DB/XA connection for** &XADBC2
Action: Inform system service.

K230 XA (&PID**) Int. error:** &TEXT32
Action: Inform system service.

K231 XA (&PID**) Int. error: PETA not supported**
Action: Inform system service.

K232 XA (&PID**) Int. error: DBSTAT secondary opcode inconsistent**
Action: Inform system service.

13.14.2.3 P messages

All messages from the OSI-TP protocol stack start with the letter “P”.

P messages with openUTM occur between the proxy container and the EIS partner.

P messages with CICS can occur in the proxy container or in the openUTM-LU62 Gateway. If the term "proxy component" is used for the description of the P message, the proxy container or the openUTM-LU62 Gateway is meant, depending on the component where the message occurred.

P001 Error on OSS call (`&XPFUNC`**):** `&ACPNT, &XPRET, &XPERR, &XP1INFO, &XP2INFO`
Error on OSI-TP communication between the proxy container and the EIS partner or between the proxy components.

If the error has been reported by the transport system, message P012 is also output.

Action: Inform system service.

P002 Error on association establishment (`&XPFUNC`**):** `&ACPNT, &OSLPAP, &XPRET, &XPERR, &XP1INFO, &XP2INFO`
Error on OSI-TP connection generation between the proxy container and the EIS partner or between the proxy components.

If the error has been reported by the transport system, message P012 is also output.

Action: Inform system service.

P003 Association rejected (a_assin()): `&ACPNT`**, reason:** `&XPRJCT`**, length:** `&XPLTH`
Action: Inform system service.

P004 Association rejected (a_assin()): `&ACPNT, &OSLPAP`**, reason:** `&XPRJCT`
This message is issued if a request to establish an association was rejected by the EIS partner.

Action: See the table below.

XPRJCT	Action
1 - 16	Inform system service.
17	Unknown partner application. Action: Check the generation in the proxy container and in the EIS system openUTM resp. in the proxy components. Possibly the generation input which is generated by the Management Console was not activated at one of the partners or at the components.
18 - 33	Inform system service.
34 - 35	Configuration problem. A different number of connections was generated on both sides. Action: Check the generation in the proxy container and in the EIS system openUTM resp. in the proxy components. Possibly the generation input which is generated by the Management Console was not activated at one of the partners or at the components.
36 - 39	Inform system service.
40	Timeout during connection establishment. Connection establishment was started but could not be completed in the specified time. Action with openUTM partner: See the messages of the EIS partner. Alternatively, inform system service. Action with CICS partner: See the messages of the EIS partner and the proxy components and try to find out which component did not respond. Alternatively, inform system service.
41 - 46	Inform system service.

P005 Association rejected (a_assin()): `&ACPNT`**, reason: unknown partner, partner address:**

This message is issued if a request to establish an association was rejected from outside because the remote partner is not known to the local application.

Action: Check the generation in the proxy container and in the EIS system openUTM resp. in the proxy components. Possibly the generation input which is generated by the Management Console was not activated at one of the partners or at the components.

P006 Association rejected (a_assin()): &ACPNT &OSLPAP**, reason: wrong application context name (**&XP00BID, &XP10BID, &XP20BID, &XP30BID, &XP40BID, &XP50BID, &XP60BID, &XP70BID, &XP80BID, &XP90BID**)**
This message is issued if a request to establish an association was rejected from outside. The application context name for the remote partner does not match the application context name generated for this partner in the local application.

Action: Check the generation in the proxy container and in the EIS system openUTM resp. in the proxy components. Possibly the generation input which is generated by the Management Console was not activated at one of the partners or at the components.

P007 Error on association establishment (a_assrs()): &ACPNT, &OSLPAP, &XPRET, &XPERR, &XP1INFO, &XP2INFO
This message is output when a request to establish an association from outside returns an error. If the error has been reported by the transport system, message P012 is also output.

Action: Inform system service.

P008 Association established: &ACPNT, &OSLPAP
This message is issued when an association has been established.

Action: Normal behavior.

P009 Association rejected (a_asscf()): &ACPNT, &OSLPAP**, reason:** &XPRJCT, **length:** &XPLTH
This message is issued when active establishment of an association is rejected because the confirmation from the communication partner cannot be accepted.

Action: Inform system service.

P010 Association rejected (a_asscf()):
&ACPNT,
&OSLPAP,
reason: unknown
partner address: &PARTADDR
(&XASSREF**)**
This message is issued when active establishment of an association is rejected because the remote partner confirms establishment of an association with an address which is unknown to the local application.

Action: Inform system service.

P011 Association rejected (a_asscf()): &ACPNT &OSLPAP**, reason: wrong application context name (**&XP00BID, &XP10BID, &XP20BID, &XP30BID, &XP40BID, &XP50BID, &XP60BID, &XP70BID, &XP80BID, &XP90BID**)**
This message is issued when active establishment of an association is rejected because the remote partner confirms establishment of an association with an application context name other than the one configured for this partner in the local application.

Action: Check the generation in the proxy container and in the EIS system openUTM resp. in the proxy components. Possibly the generation input which is generated by the Management Console was not activated at one of the partners or at the components.

P012 CMX diagnostic information: &XPCTYPE, &XPCCLS, &XPCVAL
Action:

XPCVAL > 15: Error code from the transport system. Inform system service.

XPCVAL <other value>: Inform system service.

P013 Association rejected (a_asscf()): &ACPNT, &OSLPAP**, reason:** &XPCRES, &XPSRC, &XPNDIA, CCR V2 = &XP1BOOL**, Version Incompatibility =** &XP2BOOL**, ContWin Assignment rejected =** &XP3BOOL**, Bid mandatory rejected =** &XP4BOOL**, No reason =** &XP5BOOL
This message is issued when active establishment of a connection to the EIS system resp. between the proxy components is rejected by the remote partner.

Action:

If XP1BOOL, XP2BOOL or XP4BOOL are set to TRUE: Inform system service.

If XP3BOOL = TRUE: Temporary shortage concerning the number of connections.

If XP5BOOL = TRUE: Action depends on XPNDIA, see the table below.

XPNDIA	Action
1	Inform system service.
2 - 10	Configuration problem. Action: Check the generation in the proxy container and in the EIS system openUTM resp. in the proxy components. Possibly the generation input which is generated by the Management Console was not activated at one of the partners or at the components. With openUTM partners: Check that the host name specified in the Management Console for the EIS partner corresponds to the host name used in the connection request. In the event of an error, the correct name is output in message U315 of the proxy container. If message U315 contains the value `*ANY` for the host name, you must add the host name of the openUTM partner application to the host file of the system.
11 - 24	Inform system service.

P014 Error on association disconnection (`&XPFUNC` **):** `&ACPNT, &OSLPAP, &XPRET, &XPERR, &XP1INFO, &XP2INFO`

Action: Inform system service.

P015 Association disconnected (&XPFUNC **):** &ACPNT, &OSLPAP, &XPLNK, &XPSRC, &XPNDIA, &XPINI, &XP1INFO, &XP2INFO

This message is issued when an OSI-TP association is cleared.

Action: See the table below.

XPINI	Meaning
0	Normal behavior.
401	The local transport system has cleared the association. The subsequent message P012 contains the detailed CMX return code.
402	openUTM partner: The EIS partner's transport system has cleared the connection. CICS partner: The openUTM-LU62 proxy component's transport system has cleared the connection. &XP1INFO contains the reason for the connection cleardown. The following is a subset of &XP1INFO values: 0 (T_USER) The communication partner cleared the connection, possibly as a result of a user error on the partner side. 258 (T_RSAP_NOTATT) The partner cleared the connection because the addressed communication endpoint was not registered there. 482 (T_RLNOCONN) The local transport system could not establish the connection because no network connection is available.
403 - 406	Inform system service.
407	The originator is the local proxy component.
408	The EIS partner or the proxy component openUTM-LU62 Gateway initiated the abort.

P016 Association disconnected (a_relin()): `&ACPNT, &OSLPAP, &XPLNK, &XPNDIA`
This message is issued if an association is cleared because a "release indication" was received.

Action: See the table below.

XPNDIA	Meaning
0	NO_REASON_GIVEN
21 - 23	The association is cleared by the EIS partner or by the partner proxy component with release.

P017 OSS decoding error: `&XPDU, &XP1DIA, &XP2DIA, &XP3DIA`
Action: Inform system service.

P018 FSM protocol error: `&ACPNT, &OSLPAP,  &XPPTYP, &XPFSMN`
Action: Inform system service.

P019 APDU contains invalid value: `&ACPNT, &OSLPAP,  &XPAPDU, &XP3INFO`
Action: Inform system service.

13.14.2.4 U messages

U184 DMS error `&DMSE` for file `&FNAM`
Action: See the DMS error code in Section 13.16, "Error codes" or inform system service.

U185 kdcdef not allowed during application run
Action: User error. You can update the proxy container configuration only after you have stopped the proxy container.

U189 `&OBJ1` (`&PTRM`, `&PRNM`): IPC shortage of `&IPCOBJ &IPCREAS`
Action: For the meaning of the inserts &IPCOBJ and &IPCREAS see the table below. For inserts which are not contained in the table inform system service.

&IPCOBJ	&IPCREAS	Meaning	Action
TSAP	tsapname	openUTM network process not yet started. It was not possible to log in at transport system	Normal behavior at application start. Otherwise: Check BCAMAPPL and ACCESS-POINT in openUTM generation.
NET	PROC DEAD	openUTM net process terminated.	See the earlier message U3nn.
LETT	IPC FULL	Data area already too full to take any more data for this connection.	Check openUTM generation or `UTM_IPC_LETTER` environment variable (see note below).
LETT	EXTP FULL	Data area for received messages already fully occupied.	Check openUTM generation or `UTM_IPC_LETTER` environment variable (see note below).
LETT	USED	Data area is occupied.	Check openUTM generation or `UTM_IPC_LETTER` environment variable (see note below).
LETT LETT	MAX ILETT MAX OLETT	Maximum data area per connection in use.	Check openUTM generation or `UTM_IPC_EXPT_LETTER` environment variable (see note below).
SEMA	USED	Maximum number of semaphores in use.	Check openUTM generation or `UTM_IPC_EXPT_LETTER` environment variable (see note below).

i The message with &PCOBJ=LETT refers to the data area in shared memory which is used for the exchange of messages between the proxy container processes.

Increase the value for the environment variables UTM_IPC_LETTER (default: 1600) and UTM_IPC_EXTP_LETTER (default: 32) in the following file:

- Solaris and Linux systems:
  <proxy_cont_home>/shsc/startcontainer.sh
- Windows systems:
  <proxy_cont_home>\shsc\startcontainer.cmd

To activate the value, you must restart the proxy. You can find further information in Section 12.1, "Shared memory of the proxy container".

U190 &OBJ1 SHM error (key: &SHMKEY, lth: &SHMLTH): &UERRNO
The insert &UERRNO has the following meaning:

&UERRNO	Meaning	Action
1	The shared memory cannot be established with the specified size.	User error. Action: If necessary, the system has to be tuned. Please refer to the release notes.
2 - 5	The shared memory cannot be established.	Inform system service.

U205 utmtimer: Error &UERRNO during utmtimer run
The insert &UERRNO has the following meaning:

&UERRNO	Meaning	Action
1 - 21	Internal error.	Inform system service.
22	Proxy container is being terminated.	Normal behavior.
29 - 32	Internal error.	Inform system service.

U206 utmtimer: Message with incorrect type received
Action: Inform system service.

U207 utmtimer: Reallocation of timer list from `&DIA1` **to** `&DIA2` **elements**
Action: Inform system service.

U223 `&OBJ1` **UTM application** `&APPL` **still running according to internal status, appfile:** `&OBJ3`
You have tried to start the proxy container more than once.

U227 `&OBJ1` **UTM application** `&APPL` **terminated by kdcrem**
Action: `Remove` script or `Forced Clear` has been activated.

U228 utmmain: Read error on pipe, errno: `&ERRNO`
Action: Inform system service.

U229 utmmain: `&OBJ1` **process died, pid:** `&PID`**,** `&SIGEXIT` **(** `&STRTIME` **)**
Action: Inform system service.

U231 utmmain: utmwork process died, pid: `&PID` **unexpected exit code:** `&EXTCODE` `&SIGEXIT` **(** `&STRTIME` **)**
Action: Inform system service.

U304 `&OBJ1` **(pid:** `&PID, &TNSNAME` **):** `&NETFCT` **call: Error** `&NETERR`
Error during activation of communication endpoint `&NETFCT`.

Action: Configuration problem (see message U305) or inform system service.

U305 `&OBJ1` **(pid:** `&PID`**): CMX application** `&BCAP` **already attached**
Error during activation of communication endpoint `&BCAP`. This communication endpoint is not unique on the host.

Action: The communication endpoints depend on the container name and the generated container port number. The used communication endpoints are written to the file `<proxy_cont_home>/def/input.system.txt` (`BCAMAPPL <communication endpoint>`). You must install the proxy container with another container name or port number.

U306 `&OBJ1` **(pid:** `&PID, &TNSNAME` **): Error** `&UERRNO` **during process run**
Action: See the table below.

The insert `&UERRNO` has the following meaning:

&UERRNO	Meaning	Action
1 - 11	Internal error during container connection setup.	Inform system service.
12	No work process when receiving.	Normal behavior when the proxy container terminates.
13	Internal error during container connection setup.	Inform system service.
14	Concurrent clearing of connections.	Normal behavior.
19 - 21	Internal error during container connection setup.	Inform system service.
22	No work process when sending - application terminated.	Normal behavior when the proxy container terminates.
23 - 24	Internal error during container connection setup.	Inform system service.
25	Connection was cleared down by the container.	Normal behavior.
26	Connection cleardown was identified on sending.	Normal behavior.
27 - 36	Internal error during container connection setup.	Inform system service.
37 - 38	Login for local communication endpoints was unsuccessful (error when activating the communication endpoint)	User error: Change the configuration, see also U304/U305.
41 - 43	Internal error during container connection setup.	Inform system service.
44	Recipient of connection setup request could not be determined.	User error: Change the configuration.
45	Connection already cleared down.	Normal behavior.
51 - 58	Internal error during container connection setup.	Inform system service.
82	Sender of connection setup request could not be determined.	User error: Change the configuration.
83 - 283	Internal error during container connection setup.	Inform system service.

U307 `&OBJ1` **(pid:** `&PID, &TNSNAME` **): Invalid event** `&EVENT`
Action: Inform system service.

U309 `&OBJ1` **(pid:** `&PID, &TNSNAME` **): KDCSTRMA called - reason:**
    `&TRMA ( &STRTIME )`
Action: Inform system service.

13.15 Error messages of the openUTM-LU62 Gateway

This section lists all messages of the openUTM-LU62 Gateway:

- openUTM-LU62 Gateway error messages on start-up
- openUTM-LU62 Gateway error messages at runtime
- openUTM-LU62 Gateway error messages on status queries
- openUTM-LU62 Gateway error messages during administration
- openUTM-LU62 Gateway error messages during configuration

13.15.1 openUTM-LU62 Gateway error messages on start-up

When the openUTM-LU62 Gateway starts, the openUTM-LU62 utility `u62_start` outputs messages to `stdout`.

All the messages start with the string `u62_start <nn>`, where `<nn>` indicates the message number.

- The messages with the following numbers indicate normal behavior:
  `17`, `18`, `24`, `25`, `26`, `28`, `29`
- For information on the messages with the numbers `05`, `06` and `07`, see below.
- If any other `u62_start <nn>` messages which are not listed are output, inform the system service.

05 Directory `&DIRNAME` cannot be read, errno `&ERRNO` (`&ERRTEXT`)
Action: For ERRNO, see Section 13.16.2, "System error codes" and/or inform the system service.

06 Configuration file `&FILENAME` cannot be opened, errno `&ERRNO` (`&ERRTEXT`)
Action: For ERRNO, see Section 13.16.2, "System error codes" and/or inform the system service.

07 Error reading configuration file `&FILENAME`, errno `&ERRNO` (`&ERRTEXT`)
Action: For ERRNO, see Section 13.16.2, "System error codes" and/or inform the system service.

13.15.2 openUTM-LU62 Gateway error messages at runtime

Messages concerning the OSI-TP protocol stack for connections between the proxy container and the openUTM-LU62 Gateway begin with letter P. All of these messages are described in the section "P messages" on page 536.

All other messages of the openUTM-LU62 Gateway begin with the string `u62_tp[<comp>]<nnn>`.

In this prefix, `<comp>` specifies the sub-component of the openUTM-LU62 Gateway and `<nnn>` the message number.

004 Error on signal handling, errno `&ERRNO` (`&ERRTEXT`)
Action: Inform system service.

006 Error opening the configuration file `&FILENAME`, errno `&ERRNO` (`&ERRTEXT`)
Action: For ERRNO see the section "System error codes" on page 562 or inform system service.

010 The instance for the local LU name `&LUNAME` is already running
Action: Inform system service.

012 Internal error occurred
Action: Inform system service.

015 PID file `&PIDFILE` cannot be created, errno `&ERRNO` (`&ERRTEXT`)
Action: For ERRNO see the section "System error codes" on page 562 or inform system service.

016 Write to PID file `&PIDFILE` failed, errno `&ERRNO` (`&ERRTEXT`)
Action: For ERRNO see the section "System error codes" on page 562 or inform system service.

026 Crash of openUTM-LU62 in module `&MODULNAME`
Action: Inform system service. Save the contents of the following directory:

- On Solaris and Linux systems: `/opt/lib/utmlu62/PROT`
- On Windows systems: `<gateway_home>\utmlu62\PROT`

If possible, you should reproduce the error with the instance trace activated.

027 Shutdown by the XAP-TP provider: `&REASON`
Action: Inform system service. Save the contents of the following directory:

- On Solaris and Linux systems: `/opt/lib/utmlu62/PROT`
- On Windows systems: `<gateway_home>\utmlu62\PROT`

If possible, you should reproduce the error with the XAP-TP trace activated.

036 Error in allocating a new shared memory of length `&LEN`**,**
errno `&ERRNO` **(**`&ERRTEXT`**)**
Action: For ERRNO see Section 13.16.2, "System error codes" or inform system service.

040 Error in the configuration of the local LU `&LLUNAME` **or of the partner LU** `&RLUNAME`
Action: Generation error in one of the following proxy components:

- On Solaris systems: SNAP-IX
- On Linux and Windows systems: IBM Communications Server

048 The node is deactivated: No conversations can be established
The node of the SNAP-IX or IBM Communications Server was presumably deactivated by the administrator. It is no longer possible to start conversations to the EIS partner.

Action: Check the following proxy component:

- On Solaris systems: SNAP-IX
- On Linux and Windows systems: IBM Communications Server

302 Conversation rejected, TP Name `&TPNAME`**, job submitted by LU6.2 side:**
Restart of the nominated control instance not completed yet
This can occur when the connection to the EIS partner is re-established after a crash, but the connection between the openUTM-LU62 Gateway and the proxy container has not yet been re-established.
Action: Repeat the job.

303 Incoming conversation (TP Name `&TPNAME`**, job submitted by LU6.2 side)**
supplies initialization data that will be discarded by openUTM-LU62.
Action: Inform system service.

304 Conversation rejected, TP Name `&TPNAME`**, job submitted by LU6.2 side:**
Security data not supported by the OSI TP partner or contain invalid characters.
Action: Inform system service.

305 Conversation rejected, TP Name `&TPNAME`, job submitted by LU6.2 side:
The alias name `&ALIAS` of the partner LU is not identical with the configured name `&CONFALIAS`.
Action: Check the openUTM-LU62 Gateway generation.

306 Conversation rejected, TP Name `&TPNAME`, job submitted by LU6.2 side:
Service TPs using mapped (!) conversations are not supported!
The LU6.2 partner has attempted to start a conversation with a non-printable TP name (i.e. a transaction code between X'00' and X'3F'). The openUTM-LU62 Gateway does not support these types of service TPs, except for the resync-TP X'06F2'. This message appears, for example, when the EXEC CICS START function that addresses the service TP X'02' in the openUTM-LU62 Gateway is used.

307 Conversation rejected, TP Name `&TPNAME`, job submitted by LU6.2 side:
Configuration error: The net name of the local and/or of the partner LU has been changed!
Action: Check the openUTM-LU62 Gateway generation.

308 Conversation rejected, TP Name `&TPNAME`, job submitted by LU6.2 side:
Error in the exchange of the log names between the local and the remote LU!
The LU6.2 partner has opened a conversation to the openUTM-LU62 Gateway with sync-level 2 although the log names have not yet been exchanged between the two LUs, or a fatal error occurred the last time the log names were exchanged. The openUTM-LU62 Gateway reacts to this protocol violation by immediately closing the conversation.
Action: The administrator of the transactional resources must check which actions must be reset. System service cannot check this.

309 `RECEIVE_ALLOCATE` rejected due to configuration error:
The local LU alias name `&LUNAME` is not configured!
Action: Check the generation of

- The openUTM-LU62 Gateway
- On Solaris systems: SNAP-IX
- On Linux and Windows systems: IBM Communications Server

310 `RECEIVE_ALLOCATE` failed: LU6.2 base software is not running!
Action: Start the following proxy component:

- On Solaris systems: SNAP-IX
- On Linux and Windows systems: IBM Communications Server

311 `RECEIVE_ALLOCATE` **failed: system error (errno** `&ERRNO` **) occurred:** `&ERRTEXT`
Error

- On Solaris systems: in SNAP-IX
- On Linux and Windows systems: in the IBM Communications Server

Action: For ERRNO see Section 13.16.2, "System error codes" or inform system service.

312 Allocation of a conversation to the LU6.2 partner rejected, TP Name `&TPNAME`**, job submitted by OSI TP side.**
Action: Check the generation of

- The openUTM-LU62 Gateway
- On Solaris systems: SNAP-IX
- On Linux and Windows systems: IBM Communications Server

313 Allocation of a conversation to the LU6.2 partner rejected, TP Name `&TPNAME`**, job submitted by OSI TP side: The LU6.2 base software is not running!**
Reason: The proxy component is not started.

Action: Start the following proxy component:

- On Solaris systems: SNAP-IX
- On Linux and Windows systems: IBM Communications Server

314 Allocation of a conversation to the LU6.2 partner rejected, TP Name `&TPNAME`**, job submitted by OSI TP side: system error (errno** `&ERRNO` **) occurred:** `&ERRTEXT`
Error

- On Solaris systems: in SNAP-IX
- On Linux and Windows systems: in the IBM Communications Server

Action: For ERRNO see Section 13.16.2, "System error codes" or inform system service.

315 Actively allocated conversation to LU6.2 partner deallocated (TP Name `&TPNAME`**, job submitted by OSI TP side) again due to a configuration error: The net name of the local and/or of the partner LU has been changed!**
Action: Check the generation of

- The openUTM-LU62 Gateway
- On Solaris systems: SNAP-IX
- On Linux and Windows systems: IBM Communications Server

320 Allocation of a conversation to LU6.2 partner failed, TP Name `&TPNAME`, job submitted by OSI TP side: Allocation error (code = `&ERRTEXT`)
No conversations to the TP `&TPNAME` of the LU6.2 partner can be actively opened at the moment. (`&TPNAME` is in this case the CICS transaction code, for example.) The error code `&ERRTXT` then contains the return code of the corresponding LU6.2 call in plain text.

Action: Start the CICS application.

321 Allocation of a conversation to LU6.2 partner failed, TP Name `&TPNAME`, job submitted by OSI TP side: Inconsistency in the configurations of openUTM-LU62 and the LU6.2 base software
Action: Check the generation of

- The openUTM-LU62 Gateway
- On Solaris systems: SNAP-IX
- On Linux and Windows systems: IBM Communications Server

322 Allocation of a conversation to LU6.2 partner failed, TP Name `&TPNAME`, job submitted by OSI TP side: Security data invalid
Action: Inform system service.

332 Conversation to the LU6.2 partner (TP Name `&TPNAME`, job submitted by OSI TP side) terminated by shutdown of the LU6.2 base software!
Action: Check the diagnostic information and restart

- On Solaris systems: SNAP-IX
- On Linux and Windows systems: IBM Communications Server

333 Conversation to the LU6.2 partner (TP Name `&TPNAME`, job submitted by LU6.2 side) terminated by shutdown of the LU6.2 base software!
Action: Check the diagnostic information and restart

- On Solaris systems: SNAP-IX
- On Linux and Windows systems: IBM Communications Server

334 Conversation to the LU6.2 partner (TP Name `&TPNAME`, job submitted by OSI TP side) terminated by conversation error!
Possibly caused by an administrative shutdown of all sessions.
Action: Inform the administrator or system service.

335 Conversation to the LU6.2 partner (TP Name `&TPNAME`, job submitted by LU6.2 side) terminated by conversation error!
Possibly caused by an administrative shutdown of all sessions.
Action: Inform the administrator or system service.

336 User control data received on a conversation to the LU6.2 partner
(TP Name `&TPNAME`, job submitted by OSI TP side)
=> Termination of the conversation by openUTM-LU62!
Action: Inform system service.

337 User control data received on a conversation to the LU6.2 partner
(TP Name `&TPNAME`, job submitted by LU6.2 side)
=> Termination of the conversation by openUTM-LU62!
Action: Inform system service.

338 Return code `AP_STATE_CHECK` received on a conversation to the LU6.2 partner
(TP Name `&TPNAME`, job submitted by OSI TP side)
=> Termination of the conversation by openUTM-LU62!
Action: Inform system service.

339 Return code `AP_STATE_CHECK` received on a conversation to the LU6.2 partner
(TP Name `&TPNAME`, job submitted by LU6.2 side)
=> Termination of the conversation by openUTM-LU62!
Action: Inform system service.

340 Incoming conversation to the LU6.2 partner rejected,
(TP Name `&TPNAME`, job submitted by LU6.2 side)
Expiration of the timer for the supervision of the association allocation
This maximum wait time is configurable in the openUTM-LU62 Gateway using the parameter `ALLOC-TIME`, where the default value is 30 seconds. If this message appears often, you should either increase the session limit or decrease the number of parallel connections for the openUTM-LU62 Gateway. Alternatively, inform system service.

341 Incoming conversation to the LU6.2 partner rejected,
(TP Name `&TPNAME`, job submitted by OSI TP side)
Expiration of the timer for the supervision of the association allocation
This maximum wait time is configurable in the openUTM-LU62 Gateway using the parameter `ALLOC-TIME`, where the default value is 30 seconds. If this message appears often, you should either increase the session limit or decrease the number of parallel connections for the openUTM-LU62 Gateway. Alternatively, inform system service.

350 XAP-TP provider rejects association allocation
(TP Name `&TPNAME`): result = `&RES`, source = `&SRC`, reason = `&RSN`
The proxy container rejects the association.
Action: See the proxy container messages.

351 XAP-TP provider reports loss of association
(TP Name `&TPNAME`): source = `&SRC`, reason = `&RSN`, event = `&EVT`
The proxy container rejects the association.
Action: See the proxy container messages.

352 OSI-TP partner (XAP-TP user) rejects begin dialogue request:
TP Name `&TPNAME`**, job submitted by LU6.2 side**
The proxy container rejects the dialog request.
Action: See the corresponding proxy container message or inform system service.

353 OSI-TP partner (XAP-TP user) rejects begin dialogue request:
TP Name `&TPNAME`**, job submitted by OSI TP side: Reason =** `&RSN`
The proxy container rejects the dialog request.
Action: See the corresponding proxy container message or inform system service.

354 Diagnostic information in the initialization data of
AP_TP_BEGIN_DIALOGUE_CNF: `0xhhhh (d,d)`
`hhhh` represents the hexadecimal value and `d,d` represent the according decimal values.

For UTM as OSI-TP partner, the initialization data contains further information on the reason, why the dialog has been rejected. The first value shows whether the fault is transient (1) or permanent (2).

The second value shows the detailed reason for rejection.

For the meaning of these values see Section 13.14.2.2, "K messages", description DIA3 in message K119 for DIA1=2.

361 Incoming dialogue rejected, job submitted by OSI-TP side:
Cannot decode local TPSU title
Action: Inform system service.

373 Incoming dialogue rejected, TP Name `&TPNAME`**, job submitted by OSI-TP side:**
Restart of the nominated control instance not completed yet
Action: Repeat the job.

374 Incoming dialogue rejected, TP Name `&TPNAME`**, job submitted by OSI-TP side:**
Remote application process title is not consistent with the configuration
Action: Check the generation of the proxy container and the openUTM-LU62 Gateway.

375 Incoming dialogue rejected, TP Name `&TPNAME`**, job submitted by OSI-TP side:**
Remote application entity qualifier is not consistent with the configuration
Action: Check the generation of the proxy container and the openUTM-LU62 Gateway.

390 OSI-TP dialogue aborted by partner (XAP-TP provider),
TP Name `&TPNAME`**, job submitted by LU6.2 side**
Action: See the corresponding proxy container message or inform system service.

391 OSI-TP dialogue aborted by partner (XAP-TP provider),
TP Name `&TPNAME`**, job submitted by OSI TP side**
Action: See the corresponding proxy container message or inform system service.

392 OSI-TP dialogue aborted by partner (XAP-TP user),
TP Name `&TPNAME`, job submitted by LU6.2 side
Action: See the corresponding proxy container message or inform system service.

393 OSI-TP dialogue aborted by partner (XAP-TP user),
TP Name `&TPNAME`, job submitted by OSI TP side
Action: See the corresponding proxy container message or inform system service.

408 OSI-TP partner indicates heuristic mix.
TP Name `&TPNAME`, job submitted by `&PARTNER` side
Probably the transaction between the proxy container and openUTM-LU62 Gateway is inconsistent.
Action: You usually need to change the participating database manually in this case to recoordinate the inconsistent database records. Inform system service.

409 OSI-TP partner indicates heuristic hazard.
TP Name `&TPNAME`, job submitted by `&PARTNER` side
Probably the transaction between the proxy container and openUTM-LU62 Gateway is inconsistent.
Action: You usually need to change the participating database manually in this case to recoordinate the inconsistent database records. Inform system service.

410 Syncpoint request received from LU6.2 job-receiving service.
TP Name `&TPNAME`. Transaction is aborted.
A job receiving program on the CICS side has requested a syncpoint (e.g. `EXEC CICS SYNCPOINT`) without having been requested to do so by the proxy container. This is prohibited. The transaction is aborted. `&TPNAME` specifies the transaction code on the CICS side.
Action: Change the CICS application program.

420 Error during resynchronization of a transaction. LU6.2 partner does not accept the state `&STATE`.
A transaction has been interrupted in the commit phase due to a loss of the connection on the LU6.2 side. An error has occurred during resynchronization with the LU6.2 partner. The transaction cannot be reset or aborted.
Action: You usually need to change the participating database manually in this case to recoordinate the inconsistent database records. The administrator must abort the transaction manually.

421 Log name error during resynchronization of a transaction.
A transaction has been interrupted in the commit phase due to a loss of the connection on the LU6.2 side. An error has occurred during resynchronization with the LU6.2 partner. The transaction cannot be reset or aborted.
Action: You usually need to change the participating database manually in this case to recoordinate the inconsistent database records. The administrator must abort the transaction manually.

423 Protocol error of LU6.2 partner:
Compare States `&LUWSTATE &RRI` **received in state** `&STATE1/&STATE2`**.**
Malfunction of the EIS partner.
Action: You usually need to change the participating database manually in this case to recoordinate the inconsistent database records. Inform the EIS partner's administrator.

424 LU6.2 partner indicates a protocol error. State: `&STATE1/&STATE2`**,**
TP Name `&TPNAME`**, job submitted by** `&PARTNER` **side.**
Action: Inform system service.

425 Invalid log record (error type &ERROR). Transaction is removed.
A faulty log record was read during a warm start of the openUTM-LU62 Gateway. The transaction recorded in this log record can therefore not be resynchronized.
Action: You usually need to change the participating database manually in this case to recoordinate the inconsistent database records. The administrator must abort the transaction manually.

426 Problem at the XAP-TP interface. `&EVENT` **received.**
Action: Inform system service.

427 Error `&ERRNO` **when issuing** `&CALL`**.**
An error has occurred in a Solaris or Linux system call. This can lead to the abnormal termination of the openUTM-LU62 Gateway.
Action: For ERRNO see Section 13.16.2, "System error codes" or inform system service.

510 The LU (alias name = `&ALIASNAME`**, net name =** `&NETNAME`**) has returned an abnormal reply to the Exchange Logname command.**
Action: Inform system service.

511 A cold start has been attempted by LU (alias name = `&ALIASNAME`**, net name =** `&NETNAME`**), but the local LU has logical units of work that are awaiting resynchronization from the previous activation.**
Action: Inform system service.

13.15.3 openUTM-LU62 Gateway error messages on status queries

When checking the availability of the openUTM-LU62 Gateway in the Management Console, the openUTM-LU62 `u62_sta` utility outputs messages to `stdout`.

All these messages begin with the prefix `u62_sta <nn>`, where `<nn>` specifies the message number.

- For messages with the numbers `06`, `09`, `12` and `13` see below.
- For all the other `u62_sta <nn>` messages that are not listed above, inform system service.

06 Instance `&INST`:
Instance is still initializing or terminating.
The proxy component openUTM-LU62 Gateway is currently being started or terminated.
Action: Repeat the state request.

09 Instance `&INST`:
Instance does not respond in time.
Delay which can occur with high load and in large configurations.
Action: Try checking again. Otherwise increase the preset waiting time of 20 seconds in the following script:

- Solaris and Linux systems: `<proxy_cont_home>/shsc/checkgateway.sh`
- Windows systems: `<proxy_cont_home>\shsc\checkgateway.cmd`

12 No openUTM-LU62 instance active.
The proxy component openUTM-LU62 Gateway has not been started yet.
Action: Start the openUTM-LU62 Gateway.

13 The given instance is not active.
The proxy component openUTM-LU62 Gateway has not been started yet.
Action: Start the openUTM-LU62 Gateway.

13.15.4 openUTM-LU62 Gateway error messages during administration

When administering the openUTM-LU62 Gateway in the Management Console (activating/deactivating traces) the openUTM-LU62 `u62_adm` utility outputs messages to `stdout`.

All these messages begin with the prefix `u62_adm <nn>`, where `<nn>` specifies the message number.

- Messages with the following numbers indicate normal behavior:
  `20` to `45`, `56`
- For messages with the numbers `06`, `09`, `12`, `13`, `52`, `53` and `59` see below.
- For all the other `u62_adm <nn>` messages that are not listed above, inform system service.

06 Instance `&INST`:
Instance is still initializing or terminating.
The proxy component openUTM-LU62 Gateway is currently being started or terminated.
Action: Repeat the action.

09 Instance `&INST`:
Instance does not respond in time.
Delay which can occur with high load and in large configurations.
Action: Repeat the action.

12 No openUTM-LU62 instance active.
The proxy component openUTM-LU62 Gateway has not been started yet.
Action: Start the openUTM-LU62 Gateway.

13 The given instance is not active.
The proxy component openUTM-LU62 Gateway has not been started yet.
Action: Start the openUTM-LU62 Gateway.

52 Error opening the trace file `&FILENAME`, errno `&ERRNO` (`&ERRTXT`)
Action: For ERRNO see Section 13.16.2, "System error codes" or inform system service.

53 Error reading from trace file `&FILENAME`, errno `&ERRNO` (`&ERRTXT`)
The specified file is not a trace file of the openUTM-LU62 Gateway.
Action: Specify the correct file.

59 Extracting the protocol trace may take some time. Please wait ...
Action: Wait for trace extraction to be completed.

13.15.5 openUTM-LU62 Gateway error messages during configuration

When configuring the openUTM-LU62 Gateway in the Management Console (update configuration), the openUTM-LU62 `u62_gen` utility outputs messages to `stdout`.

All these messages begin with the prefix `u62_gen <nn>`, where `<nn>` specifies the message number.

- A message with the number `u62_gen 60` indicates normal behavior.
- For the messages `51`, `58`, and `59` see below.
- For all the other `u62_gen <nn>` messages that are not listed above, inform system service.

51 File `&FILENAME` cannot be opened, errno `&ERRNO` (`&ERRTEXT`)
Action: For ERRNO see Section 13.16.2, "System error codes" or inform system service.

58 Error in writing to file `&FILENAME`, errno `&ERRNO` (`&ERRTEXT`)
Action: For ERRNO see Section 13.16.2, "System error codes" or inform system service.

59 Error in reading from file `&FILENAME`, errno `&ERRNO` (`&ERRTEXT`)
Action: For ERRNO see Section 13.16.2, "System error codes" or inform system service.

13.16 Error codes

This section provides information on the following topics:

- Error codes during file processing (DMS error codes)
- System error codes

13.16.1 Error codes during file processing (DMS error codes)

In conjunction with file processing, return codes in the form yxxx occur in the event of an error. These are also known as DMS errors and have the following meanings:

y The first character y denotes the function in which the error occurred. y may have the following values:

`A`	Error in loading shared memories into the address space
`C`	Error in close call
`D`	Error in signing off from a shared memory area
`F`	Error in fstat/stat call
`G`	Error in allocating shared memory
`L`	Error in lseek call
`O`	Error in open call
`R`	Error in read call
`W`	Error in write call
`X`	Error in create call

xxx The three characters xxx represent, in printable form, the error number which is stored by the operating system in the external variable `errno`. The meanings of the individual error numbers are described in the operating system manuals and in the `errno.h` header file. You will find some of the error codes which occur most often in Section 13.16.2, "System error codes".

In addition, the errors `CONS`, `LERR`, `OERR`, `REND`, `RERR`, `WERR`, `SXDE`, `SDDE` and `SDFE` can occur. These have the following meaning:

`CONS` The data contents are inconsistent.

`LERR` Iseek could not be positioned at the desired point.

`OERR` An attempt was made to open a directory as a normal file.

`REND` End-of-file reached on reading from a file.

`RERR` Insufficient bytes could be read.

`WERR` Insufficient bytes could be written.

`SXDE` A directory could not be created.

`SDDE` A directory could not be deleted.

`SDFE` A file could not be deleted.

13.16.2 System error codes

Several messages from the BeanConnect proxy container and the openUTM-LU62 Gateway contain as an insert a message number which is returned by the operating system. The actions in response to these messages depend on the error code.

The following table provides information on the most frequent reasons for errors. All other reasons not listed in this table are severe errors. Inform system service if these errors occur.

ERRNO	Meaning
2	File or directory not available.
12	Storage bottleneck in the system.
13	No access rights for file or directory.
17	File already exists.
28	Disk bottleneck.
36	Operating facility deleted.

14 Cobol2Java

BeanConnect clients can communicate with a BS2000/OSD-COBOL application using a number of different protocols. Byte arrays or strings are used for the exchange of information. The structure of the information is defined by the COBOL application. As a result, it is difficult for application developers to exchange data with the legacy service since this demands precise knowledge of the data structure of the COBOL program. Cobol2Java helps simplify the integration of BS2000/OSD-COBOL applications and BeanConnect clients.

This chapter contains information on the following subjects:

- Mapping COBOL data types to Java classes
- Converting COBOL data types
- Programming reference
- Example
- Error messages and error handling

14.1 Mapping COBOL data types to Java classes

Cobol2Java permits the object-oriented mapping of COBOL data structures to Java classes.

Cobol2Java consists of the following parts:

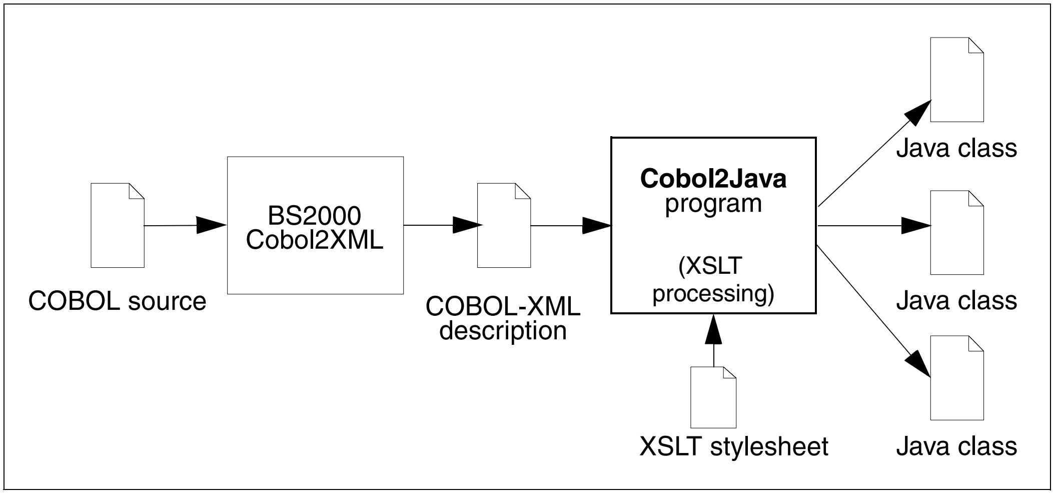
- Cobol2XML, a BS2000 tool for generation of an XML description of a BS2000 COBOL structure
- a framework with Cobol2Java conversion classes
- a program generator for the generation of application-specific classes on the basis of this framework

The program generator makes use of an XSLT stylesheet to generate application-specific Java classes. This is performed on the basis of an XML description of the COBOL service that you can generate using the Cobol2XML tool in BS2000/OSD.

The Cobol2Java classes are able to process the byte arrays of a COBOL program or generate a byte array that can be interpreted by the COBOL service. To access the individual data fields within the byte array, Cobol2Java provides data access methods appropriate for the data type as it is defined in the COBOL structure.

The figure below illustrates the generation operation.

Figure 69: Cobol2Java generation procedure



The Cobol2XML tool in BS2000/OSD can be used to generate an XML file from the BS2000/OSD-COBOL source. Cobol2Java then uses this file as the basis for creation of the Java classes.

It is also possible to use Cobol2Java with COBOL programs developed for other platforms, such as UNIX systems. For this to be possible, the XML description must be generated on a BS2000 system. It is not, however, possible to guarantee full functionality since the memory allocation is dependent on the COBOL compiler.

14.1.1 System requirements

The following operating systems are supported as Cobol2Java target platforms:

- Solaris, Linux and other UNIX systems
- Windows

The following programs must be installed before you can use Cobol2Java:

- Java SDK
- Ant (supplied)

Cobol2XML must be installed on a BS2000/OSD system.

14.1.2 Installation

The components required for Cobol2Java are made available together with the BeanConnect tools.

The installation of Cobol2Java is described in Section 3.5, "Installing the BeanConnect tools".

The following directory structure is created on installation:

`/`	`build.xml`	Ant script
	`cobol2java.properties`	Configuration file
	`runAnt.sh`	Ant script (UNIX system)
	`runAnt.bat`	Ant script (Windows system)
	`xml2java.sh`	Script (UNIX system)
	`xml2java.cmd`	Script (Windows system)
`/api`		JavaDoc
`/BS2Files/ftp`	`COB2XML.LIB`	BS2000 tool Cobol2XML for file transfer with FTP
`/BS2Files/openFT`	`COB2XML.LIB`	BS2000 tool Cobol2XML for file transfer with openFT
`/Docs`	`Copyright.htm`	Copyright notes for the employed openSource products
	`Readme.pdf`	Release Notice
`/lib`	`ant.jar`	Ant
	`ant-launcher.jar`	
	`BeanConnectCob2java.jar`	Transfer tool: XML to Java
	`BeanConnectCob2java_ext.jar`	Necessary for the execution of the generated Cobol2Java classes if Cobol2Java is used in a separate program without BeanConnect
	`BeanConnectEncoding.jar`	Code conversion
	`functions.xslt`	XSLT stylesheet
	`mkcob2java.xslt`	XSLT stylesheet
	`newformat.dtd`	Document Type Definition
`samples`	`*.xml`	Sample XML files generated using Cobol2XML within BS2000/OSD
	`newformat.dtd`	Document Type Definition

14.2 Converting COBOL data types

Mapping COBOL data types to Java classes comprises the following steps:

- Creating an XML description for a COBOL program in BS2000
- Generating Java classes on UNIX or Windows systems

14.2.1 Creating an XML description for a COBOL program in BS2000

The BS2000/OSD tool Cobol2XML is used to convert a COBOL program or COPY element to XML. The following steps are required:

- Transferring the LMS library `COB2XML.LIB` to BS2000/OSD
- Converting the data structures from COBOL programs or COPY elements to XML using the procedure `D.XMLPROG` or `D.XMLCOPY`
- Transferring the XML descriptions in text format to UNIX systems or Windows systems for further processing with Cobol2Java.

14.2.1.1 Transferring the LMS library to BS2000/OSD

Cobol2XML is a BS2000/OSD tool. You must therefore transfer the library `COB2XML.LIB` to a BS2000/OSD user ID. Depending on the transfer mode, you can either transfer the file `COB2XML.LIB` under `cobol2java/BS2Files/ftp` or the file `COB2XML.LIB` under `cobol2java/BS2Files/openFT`.

ℹ When transferring with ftp, use `binary` mode.
When transferring with openFT, use the file type `binary`.
If the BS2000 system does not possess sufficient storage space, with ftp you may get the error message `DD33` (File not present).

⚠ Do not rename the library in the BS2000/OSD system since the conversion procedure accesses elements of this library.

14.2.1.2 Converting the data structures

The conversion procedure can only be called under the user ID where the library `COB2XML.LIB` is stored.

Before calling the conversion procedure, assign the link names `COBLIB<n>` (<n>= 1, .., 9) to the COPY libraries required for compilation of the source. For example, for a UTM COBOL program assign a link name to the library containing the UTM-COPY elements in order to ensure that the UTM-COPY elements can be found. You do this using the command `ADD-FILE-LINK`:

```
/ADD-FILE-LINK COBLIB1, <copy_library>
```

(see also the manual “COBOL2000 (BS2000/OSD), COBOL-Compiler”).

i The link name `COBLIB` is used internally.

You can then start the procedures `D.XMLPROG` or `D.XMLCOPY` from the library `COB2XML.LIB`. The parameters are described in the sections below.

14.2.1.3 D.XMLPROG

`D.XMLPROG` generates an XML description of the data structures used in a COBOL program on the basis of the corresponding program source.

Start `D.XMLPROG` with the command `CALL-PROCEDURE` as follows:

```
CALL-PROC
FROM-FILE=*LIBRARY-ELEMENT(LIBRARY=COB2XML.LIB,ELEM=D.XMLPROG)
,PROCEDURE-PARAMETERS=(
{SRC=FILE,TSTNAM=<cobol_source>|SRC=LIB,LIB=<cobol_lib>, TSTNAM=<cobol_
source>},
[XMLOUT=<ouptput_file>,]
[COBRUN=<compiler_options>] )
```

Description of the parameters:

SRC=FILE,TSTNAM=<cobol_source>
The COBOL program from whose data structures an XML description is to be generated can be found in the file `<cobol_source>`.

SRC=LIB,LIB=<cobol_lib>, TSTNAM=<cobol_source>
The COBOL program from whose data structures an XML description is to be generated can be found in the element `<cobol_source>` in the LMS library `<cobol_lib>`.

XMLOUT=<output_file>
Name of the file to which the data structures are written in XML format. The name must have the suffix .xml. If this parameter is not specified and the link name XMLLINK is not defined then the XML description is written to the file XMLFIL.COBOL.<progID>.XML where <progID> stands for the program name defined in PROGRAM-ID.

COBRUN=<compiler_options>
Specification of compiler options in the form of a string of maximum 121 characters. You separate the individual options with commas. You can specify, for example:

COMMENT=YES/NO

Comment lines are not output if the option COMMENT=NO is set.

DTD=YES/NO

The reference to the Document Type Definition is not output if the option DTD=NO is set.

14.2.1.4 D.XMLCOPY

D.XMLCOPY generates an XML description of the data structures defined in a COBOL COPY element on the basis of the corresponding COPY element.

You start D.XMLCOPY with the command CALL-PROCEDURE as follows:

```
CALL-PROC
FROM-FILE=*LIBRARY-ELEMENT(LIBRARY=COB2XML.LIB,ELEM=D.XMLCOPY)
,PROCEDURE-PARAMETERS=(
LIB=<cobol_lib>,
ELEM=<cobol_copy>,
[XMLOUT=<output_file>,]
[COBRUN=<compiler_options>] )
```

Description of the parameters:

LIB=<cobol_lib>, ELEM=<cobol_copy>
The COBOL COPY element from whose data structures an XML description is to be generated is available as the element <cobol_copy> in the LMS library <cobol_lib>.

XMLOUT=<output_file>
Name of the file to which the data structures are written in XML format. The name must have the suffix .xml. If this parameter is not specified and the link name XMLLINK is not defined then the XML description is written to the file XMLFIL.COBOL.<cobol_copy>.XML where <cobol_copy> stands for the name of the COPY element.

COBRUN=<compiler_options>
Specification of compiler options in the form of a string of maximum 121 characters. You separate the individual options with commas. You can specify, for example:

`COMMENT=YES/NO`

Comment lines are not output if the option `COMMENT=NO` is set.

`DTD=YES/NO`

The reference to the Document Type Definition is not output if the option `DTD=NO` is set.

14.2.1.5 Example call

```
/PROC A
/REMARK is necessary for converting UTM programs
/ADD-FILE-LINK COBLIB1,$TSOS.SYSLIB.UTM.062.COB
/ADD-FILE-LINK COBLIB2,COBOLCOPY.LIB
/CALL-PROC F-F=*LIB(LIB=COB2XML.LIB,ELEM=D.XMLPROG),P-P=(SRC=FILE -
/   ,TSTNAM=COBTAC.CBL -
/   ,XMLOUT=COBTAC.XML),LOG=N
/ENDP
```

14.2.1.6 Generated files

The output from `D.XMLPROG` and `D.XMLCOPY` can be found in the file which is either

- assigned previously to the link name `XMLLINK` using the `ADD-FILE-LINK` command, or
- specified in the `XMLOUT` parameter.

If you do not make use of any of these possibilities, the output from `D.XMLPROG` and `D.XMLCOPY` is written to the file `XMLFIL.COBOL.<progID>.XML`. Here `<progID>` stands for the program name defined in `PROGRAM-ID` (`D.XMLPROG`) or the name of the COPY element (`D.XMLCOPY`).

You have to transfer the generated files containing the XML description in text format to UNIX or Windows systems for further processing with Cobol2Java.

14.2.2 Generating Java classes on UNIX or Windows systems

You can generate Java classes with or without Ant.

14.2.2.1 Generating Java classes with Ant

Java classes are generated by means of an Ant script. The parameters are given with the file `cobol2java.properties` as follows:

```
xml.file=<xml_file>
cobol.struct=<list_of_structure_names>
package.name=<package_name>
doc.dir=<doc_directory>
jar.dest=<jarfile_name>
code.convention={java|cobol}
undef.pic9=<undef_value>
```

Description of the parameters:

xml.file=<xml_file>
The parameter `xml.file` specifies the name of the file containing the XML description of the COBOL structure that is to be processed.

⚠ Make sure that the DTD `newformat.dtd` is located in the same directory as the XML file. You can find a copy of `newformat.dtd` in the directories `lib` and `samples`.

cobol.struct=<list_of_structure_names>
In the parameter `cobol.struct`, specify a space-separated list of COBOL structures. Cobol2Java searches all the elements (records as well as fields) in the input file for the names contained in `<list_of_structure_names>`. For each found element a Java class is generated. At least one structure name must be specified.

Certain specifications for the parameter `cobol.struct` are of little use. You should therefore observe the following restrictions:

- Do not specify two structures that are nested one within the other in a single call. Reason: The class of the lower-level structure is generated twice (as root level and as sub-level class). The class that is generated later overwrites the first and errors occur during the compilation and use of the classes.

- Do not enter any recurrent structure or field names (occurs, Array). Reason: It is only possible to access the first element of the array via the generated class. Instead, you should specify higher-level structures.
- The specification of recurrent structure or field names is not permitted. A message is output and no Java classes are generated.

package.name=<package_name>
The parameter `package.name` contains the name of the package under which the generated Java classes are to be grouped.

doc.dir=<doc_directory>
`doc.dir` contains the name of the directory for JavaDoc.

jar.dest=<jarfile_name>
`jar.dest` contains the name of the JAR file that is to be generated.

code.convention={java|cobol}
`code.convention` specifies the naming convention that is to be used by the Java classes. If `cobol` is specified then all the names are taken over from the COBOL program wherever possible. Otherwise, all the names are adapted to the Java naming conventions for the naming of classes, variables, methods and objects (see also Section 14.3.2, "Naming conventions").

undef.pic9=<undef-value>
`undef.pic9` specifies the specific value in Pic9 fields that is to indicate "undefined". Here, `<undef-value>` can have the following form:

`0x<nn>` where `<nn>` is the hexadecimal byte value, or

`"'<char>'"` where `<char>` is a printing character with a 1-byte representation.

To obtain the value `'<char>'` in the Java class, the parameter must also be specified in quotes ("), e.g. `undef.pic9="'''"`

If `undef.pic9` is not specified then '0' is the predefined value for undefined Pic9 fields.

Caution! If you specify the byte value with encoding active then you may need to take account of code conversion. See also Section 14.3.3.2, "Reading a data field".

If you compile generated Java classes, it is important that the class and file names correspond in terms of the use of case-sensitive notation. Since the Windows file system does not make any distinction between uppercase/lowercase, the Java compiler may report the following error when compiling, for example, a class `Benid` within a file `BENID.java`:

```
"[javac] BENID.java:19:class Benid is public,
should be declared in a file named Benid.java"
```

This error occurs if you create the same COBOL structure with `code.convention=cobol` and then again with `code.convention=java`. If you want to modify `code.convention`, you must first delete the sources that were created beforehand in the `src` directory.

To start the program you call the script `runAnt.sh` (UNIX system) or `runAnt.bat` (Windows). This starts generation using the Ant program supplied with Cobol2Java. This generation process results in the Java sources being created in the `src` directory, compiled, and stored in the JAR file specified in `jar.dest`. If you do not want to use the Ant supplied, call Ant in the directory containing the `build.xml` file.

Some of the tools used here require a very large amount of RAM. You should therefore use the options`-Xss (stack size)` and `-Xmx (heap size)` to make the adaptations necessary for the large data structures. In UNIX systems, it may also be necessary to adapt the `stack size limit` using the `ulimit` command.

If `runAnt` is to run correctly, it must be possible to call the Java programs `javac` and `javadoc`; i.e. the Java SDK program directory must be present in the environment variable `PATH`.

It is necessary to use the separator "/" or "\\" when specifying the path names for `xml.file`, `doc.dir` and `jar.dest` since Ant interprets "\" and the following character as a control character.

Example 26 Example for a cobol2java.properties file

```
# Properties to set for Cobol2Java Program
# used by Ant

# Name of Source XML generated by the BS2000 COBOL Compiler
# Make sure the DTD File is available!!
xml.file=cobkb.xml

# Name of the COBOL Records
# Space separated list:
# cobol.struct=RECORD1 RECORD3 will create Java classes for
# RECORD1 and RECORD3
cobol.struct=MPUT-MSG

# Name of the package to be generated
package.name=de.siemens.cob2java.cobkb
#

# Directory for JavaDoc
doc.dir=doc/cobkb

# Jar file name
jar.dest=cobkb.jar

# Determines what code convention the generated code will use
#
# code.convention=java
# code.convention=cobol

code.convention=java

# defines a non numeric value which marks a PIC9 field as undefined
#
undef.pic9=0x20
```

14.2.2.2 Generating Java classes without Ant

The generation procedure described above is usable limited for batch processing and bulk data operations.

If you are in the Cobol2Java home directory you can call the following command in order to perform a large number of generations on the basis of your own specifications.

```
java -Xss8m -Xmx512m -classpath lib/BeanConnectCob2java.jar
de.siemens.cob2java.Cob2Java [-undef.pic9=<default-value>] <xml_file>
<package_name>
<code_convention>
<cobol_struct1> [<cobol_struct2> ...]
```

Description of the parameters:

undef.pic9=<default-value>
`undef.pic9` specifies the specific value for Pic9 fields that is to represent "undefined". For details, see the description of `undef.pic9` in Section 14.2.2.1, "Generating Java classes with Ant".

<xml_file>
Name of the XML file containing the description of the COBOL data structures.

<package_name>
Package name for the generated classes.

<code_convention>
Convention used by the generated classes: `cobol` or `java`.

When using this parameter, please note the comments concerning the `code.convention` parameter on page 572.

<cobol_struct>[1...<n>]
Space-separated list of COBOL structures that are to be converted. At least one name must be specified.

i Since some of the tools used here require a large amount of memory, it is generally necessary to use the options `-Xss (stack size)` and `-Xmx (heap size)` and make the required adaptations for large data structures. In UNIX systems, it may also be necessary to adapt the `stack size limit` using the `ulimit` command.

Example 27 Generating Java classes

```
java -Xss8m -Xmx512m -classpath lib/BeanConnectCob2java.jar
de.siemens.cob2java.Cob2Java cobkb.xml de.cobol java MPUT-MSG BENID
```

This program simply generates the Java classes in the subdirectory `src`. It does not perform any compilation or create any JAR files or documentation.

An example can be found in the script `xml2java.sh` on UNIX systems or `xml2java.cmd` on Windows systems.

⚠ Make sure that the DTD `newformat.dtd` is located in the same directory as the XML file. You can find a copy of `newformat.dtd` in the directories `lib` and `samples`.

14.3 Programming reference

This section contains a presentation of the framework that is used to convert COBOL data types to Java and vice versa.

14.3.1 Type assignment

The table below presents an overview of the Java classes present in the framework together with a brief description of the COBOL types for which the classes are used.

Java class	Description
`DataType`	Basic class for all conversion classes
`CobolRecord`	Basic class for COBOL structures
`PicX`	For alphabetical/alphanumerical COBOL types: `PIC X(n)`
`PicN`	For national COBOL types: `PIC N(n)`
`Pic9`	For positive, integer numerical COBOL types: `PIC 9`
`Pic9COMP`	For integer numerical COBOL types: `PIC S9(n)` and `PIC 9(n) USAGE [COMP, BINARY, COMP-5]`
`PicU`	For COBOL types for which Cobol2Java offers no special data conversion classes. The data is made available in the Java program without conversion as a byte array.

The directory `api-doc` contains the documentation for these classes.

The table below indicates the support provided for the various COBOL types and clauses:

COBOL clause	Java support
Data structures with level numbers	`CobolRecord`
Pic X (n) (alphanumerical)	`PicX`
Combinations of A, X, 9 (not only 9)	`PicX`
Pic 9 (n) (numerical)	`Pic9`
Pic N (n) (national)	`PicN`

`BINARY, COMP, COMP-5`	`Pic9Comp` for `PIC9(1)`to `PIC9(18)`
	Not supported:
	`PIC 9(19)` to `PIC 9(31)` is mapped to `PicU`

`COMP-1, COMP-2, COMP-3`	Not supported.
Alphanumerical, ready for printing Alphabetical, ready for printing	Mapped to `PicX`. The preparation mask is ignored.
Numerical, ready for printing	Not supported. Mapped to `PicU`.
`BLANK WHEN ZERO`	Not supported. Mapped to `PicU`.
`INDEX`	Not supported.
`POINTER; PROCEDURE POINTER, OBJECT REFERENCE`	Not supported.
`JUSTIFIED RIGHT`	`PicX`, ignored.
`SYNCHRONIZED`	Supported.
Level number 77	Supported.
`OCCURS`	Limited support. ● Dynamic arrays (`OCCURS DEPENDING ON`) are created with a fixed length. ● `OCCURS INDEXED BY`, `OCCURS KEY IS` is not supported.
`REDEFINES`	Supported.
`RENAMES`	Supported.

14.3.2 Naming conventions

This section describes how the names in the Java classes are constructed for the different code conventions.

When `cobol` is used as code convention, the following mapping rules apply:

- The COBOL name is retained as far as possible.
- Hyphens are replaced by underscores.
- All arrays are indexed with 0.

If `java` is used as code convention then the following mapping rules apply:

- All uppercase are converted to lowercase.
- Letters following a hyphen are written in uppercase. All hyphens are removed.
- The first letter of a class name is written in uppercase.
- The names of `get` methods are formed from `get` plus the attribute name and the first letter of the attribute name is written in uppercase.
- The names of `set` methods are formed from `set` plus the attribute name and the first letter of the attribute name is written in uppercase.
- All arrays are indexed with 0.

Example 28 Name assignment for the different code conventions

Depending on the code convention, the following names are formed for the COBOL field named `EMPLOYEE-RECORD`:

Code convention	Java attribute name	Java get/set method name

`cobol`	`EMPLOYEE_RECORD`	`getEMPLOYEE_RECORD ()`
		`setEMPLOYEE_RECORD ()`

`java`	`employeeRecord`	`getEmployeeRecord()`
		`setEmployeeRecord()`

If the data structure contains substructures with the name `FILLER` or if the program contains multiple substructures with the same name then these structures are numbered in order in accordance with their sequence in the xml input file. When the Java classes are generated, this number `n` is appended to the generated names in the form `_R_n`.

If the data structure contains fields with the name FILLER then these fields are numbered in order in accordance with their sequence in the xml input file. When the Java classes are generated, this number `n` is appended to the generated names in the form `_n`.

14.3.3 Accessing COBOL fields

The hierarchical COBOL data structure is mapped to a hierarchical class structure.

Consequently, a field `<XXX>` can be addressed as follows:

Read access:	`level01.getLevel02().getLevel03().get<XXX>()`
Write access:	`level01.getLevel02().getLevel03().set<XXX>()`

The construction can be used to achieve high-performance access to deeply nested fields.

Instead of

```
in.getArray2(1).getLineTab().getLinex(3).getKeyx().setData1
(1,"Value1");
in.getArray2(1).getLineTab().getLinex(3).getKeyx().setData2
(1,"Value2");
```

the fields can be accessed level by level:

```
Keyx keyx;
keyx = in.getArray2(1).getLineTab().getLinex(3).getKeyx();
keyx.setData1(1,"Value1");
keyx.setData2(1,"Value2");
```

14.3.3.1 Writing a data field

To write a data field `<XXX>` it is necessary to call the method s`etXXX()`. An object of the COBOL data field type is passed as the parameter.

```
EmployeeRecord out = new EmployeeRecord();
out.setLastName(new PicX("LastName"));
```

For each COBOL data type, there are additional methods with parameters of the type `String`, or `int` and `long`:

for `PicX, PicN`:	`setXXX(String): out.setLastName("LastName");`
for `Pic9`:	`setXXX(int), setXXX(long)`

14.3.3.2 Reading a data field

To read a data field `<XXX>` it is necessary to call the method `get<XXX>()`. The return value supplied by this method is an object of the same type as the COBOL data field. Each object possesses methods adapted to the type that are used to extract the data:

for `PicX`:	`toString()`
for `Pic9`:	`longValue()`

When COBOL is used, it is possible that a numerical data field (`PIC 9(n)`) is initialized with a non-numerical value, e.g. blanks or `'X00'`. To avoid the output of any `NumberFormatException` when accessing this type of "undefined" field, the following methods are available to check the content:

for `Pic9`:	`isUndefined(), isUndefined(byte)`

The default value for "undefined" can be specified when the Java class is created. (see Section 14.2.2, "Generating Java classes on UNIX or Windows systems").

If a data field is only initialized with the default value for "undefined" (`isUndefined()` returns `true`) and if this default value is not numerical, then the value 0 is returned when the field is read.

14.3.3.3 Replacement data type PicU

The replacement data type `PicU` is used for non-supported data fields. This permits transparent access to the data. The application programmer can then process this data with his/her own resources. The unchanged data of the partner application is provided as byte array.

This type possesses the following methods:

Read access:	`getBytes()`
Write access:	`setBytes()`

14.3.3.4 Setting and reading the data for the entire structure (for sending and receiving)

The data for an entire record or data group is read using the `getBytes()` method and written using the `setBytes()` method.

With the communication methods of BeanConnect for sending and receiving data (`sndRecord()`, `rcvRecord()` and `call()` methods with the `ByteContainer` parameter) it is also possible to specify the Java objects directly because all classes representing COBOL structures generated by Cobol2Java implement the `ByteContainer` interface.

Example 29 Sending and receiving data

```
// Java object which was created by Cobol2Java
EmployeeRecord emplRecord = new EmployeeRecord();
// Set encoding of the connection
emplRecord.setEncoding( connection.getEncoding() );
emplRecord.setEncodingActive( connection.isEncodingActive() );
// Connection object: sndRecord/rcvRecord method
connection.sndRecord(emplRecord);
connection.rcvRecord(emplRecord);
```

With the `sndRecord()` call the data from `emplRecord` is sent on the connection and with the `rcvRecord()` call the data from the connection is stored in `emplRecord`.

Detailed information on encoding is provided in the Chapter 11, "Encoding and national language support".

i To modify a data field `XXX`, it is not sufficient simply to modify the object obtained via the `get<XXX>` method. Instead, the field has to be modified using one of the `set<XXX>` methods.

Example 30 Getting and storing data

```
// Data is received and stored in the EmployeeRecord
connection.rcvRecord(in);           (1)
PicX       lastName;
String     newName = "MyName";
lastName = in.getLastName();        (2)
lastName.setString(newName);        (3)
in.setLastName(lastName);           (4)
// or in.setLastName(newName);      (5)
// The data stored in EmployeeRecord is sent connection.sndRecord(in);
```

where:

(1) Java class which was created by Cobol2Java from a COBOL data structure

(2) Return data field as `PicX` object

(3) Modify `PicX` object

(4) Modify data field via `set` method with `PicX` parameter

(5) Modify data field via `set` method with `String` parameter

14.3.4 Java/EBCDIC conversion

For conversion, proceed as follows:

1. Create the Cobol2Java objects with an empty constructor.

   ```
   cob2javaclass cob2java = new cob2javaclass();
   ```

2. Then set the encoding of the connection:

   ```
   cob2java.setEncoding( connection.getEncoding() );
   cob2java.setEncodingActive( connection.isEncodingActive() );
   ```

3. Specify the Cobol2Java objects directly in the communication methods:

   ```
   sndRecord(ByteContainer), rcvRecord(ByteContainer),
   call(ByteContainer, ByteContainer)
   ```

Detailed information on encoding is provided in the Chapter 11, "Encoding and national language support".

14.3.5 Formatted mode support

Cobol2Java provides restricted formatted mode support. The `Kcat` class is used to simplify the use of +formats. This class contains constants for `KDCS ATTRIBUTE` for formatted mode support.

14.4 Example

This section provides an example for the conversion of the data types in a COBOL program into Java classes. It contains information on how to convert a simple COBOL program in a standard case. Below you will find information on the:

- COBOL example program
- Creating the XML description
- Generating the Java classes
- Use of the generated classes

14.4.1 COBOL example program

The following program `employee.cbl` is used as an example.

```
IDENTIFICATION DIVISION.
   PROGRAM-ID. EMPLOYEE.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
   01 EMPLOYEE-RECORD
      05 EMPLOYEE-NUMBER   PIC X(08).
      05 FIRST-NAME        PIC X(20).
      05 LAST-NAME         PIC X(20).
      05 PAY-METHOD        PIC X.
      05 SALARY-INFO.
         10 ANNUAL-SALARY  PIC 9(5).
LINKAGE SECTION.
      COPY KCKBC.
      ...
      COPY KCPAC.
PROCEDURE DIVISION
...
```

14.4.2 Creating the XML description

The XML description `employee.xml` can be generated in BS2000/OSD with Cobol2XML on the basis of the COBOL program `employee.cbl`. The file `employee.xml` must then be transferred to the UNIX or Windows system.

Detailed information on Cobol2XML is provided in Section 14.2.1, "Creating an XML description for a COBOL program in BS2000"

Transferring the LMS library COB2XML.LIB to BS2000/OSD

Send the file `COB2XML.LIB` to BS2000/OSD (see Section 14.2.1.1, "Transferring the LMS library to BS2000/OSD").

Converting the data structures from the COBOL program to XML

1. Log on to the BS2000/OSD system under the user ID where the files `COB2XML.LIB` and `EMPLOYEE.CBL` are located.
2. Assign the link name `COBLIB<n>` (<n>= 1, .., 9) to the COPY libraries required for converting the source:

   ```
   /ADD-FILE-LINK COBLIB1, $TSOS.SYSLIB.UTM.062.COB
   ```

3. Start the `D.XMLPROG` procedures from the `COB2XML.LIB` library:

   ```
   CALL-PROC
   FROM-FILE=*LIBRARY-ELEMENT(LIBRARY=COB2XML.LIB,ELEM
                             =D.XMLPROG)
   ,PROCEDURE-PARAMETERS=(
   SRC=FILE,TSTNAM=employee.cbl,
   XMLOUT=employee.xml)
   ```

   The `D.XMLPROG` procedure generates the XML description of the data structures used in the COBOL program in the file `employee.cbl`. The XML description is stored in the file `employee.xml`.

Transferring the XML descriptions to the UNIX system or Windows system

Transfer the XML description `employee.xml` in text format to the UNIX system or Windows system, for example to the `samples` subdirectory of Cobol2Java, for further processing.

14.4.3 Generating the Java classes

The EmployeeRecord class is generated by Cobol2Java in UNIX systems or Windows on the basis of the XML file employee.xml. The classes are generated by Ant, compiled, and packed in the JAR file employee.jar, which can then be used as a basis for a BeanConnect client. You call Ant with the script runAnt.

Detailed information on generation is provided in Section 14.2.2, "Generating Java classes on UNIX or Windows systems".

Defining the configuration

Edit the parameter file cobol2java.properties for Ant as follows:

```
xml.file=samples/employee.xml
cobol.struct=EMPLOYEE-RECORD
package.name=de.siemens.cob2java.test
doc.dir=doc
jar.dest=employee.jar
code.convention=java
```

Setting the PATH variable

To enable runAnt to run, it must be possible to call the Java programs javac and javadoc.

Extend your PATH environment variable by adding the Java SDK program directory.

Starting generation

1. Switch to the Cobol2Java home directory.
2. Call runAnt.sh (UNIX systems) or runAnt.bat (Windows systems).

During generation, the Java sources are created in the src directory, compiled, and stored in the JAR file employee.jar. The JavaDoc of the created classes can be found in the doc directory.

14.4.4 Use of the generated classes

It is possible to create applications on the basis of the generated classes. The following class provides an example of a BeanConnect client on the basis of the class `EmployeeRecord`.

```
package net.fsc.jca.beanconnect.qa;

import javax.ejb.SessionBean;
import javax.ejb.SessionContext;

import de.siemens.cob2java.cobtypes.*; // Runtime of Cobol2Java
import de.siemens.cob2java.test.EmployeeRecord

public class EmployeeServiceBean implements SessionBean
{
   private net.fsc.jca.communication.EISConnectionFactory cf;
   public void ejbCreate() throws javax.ejb.CreateException
   {
      try {
          javax.naming.Context ic = new
          javax.naming.InitialContext();
          cf =(net.fsc.jca.communication.EISConnectionFactory)
               ic.lookup
          ("java:comp/env/eis/myEIS");

      } catch (javax.naming.NamingException ex) {
         throw new javax.ejb.CreateException
         ("NamingException:"+ex);
      }
   }

   public void ejbActivate()
   {
   }

   public void ejbPassivate()
   {
   }

   public void ejbRemove()
   {
   }

   public void setSessionContext(SessionContext ctx)
   {
   }
```

```
public String addSalary(String employeeNr, int
                        salaryIncrease)
{
   String retValue = "";
   net.fsc.jca.communication.EISConnection con = null;
  try {
     con = cf.getConnection();
     con.setServiceName("EMPLOYEE");

     // Create EmployeeRecord and accept the encoding setting
     // of the connection
     EmployeeRecord employee = new EmployeeRecord();
     try
     {   employee.setEncoding( con.getEncoding() );  }
     catch (net.fsc.beanta.encoding.EncoderException encEx) {
         // todo  Error handling
     } // catch EncoderException
     employee.setEncodingActive( con.isEncodingActive() );

     // Fetch the required EmployeeData
     employee.setEmployeeNumber( employeeNr );
     con.sndRecord( employee );
     con.rcvRecord( employee );

     // Increase the salary and send modification
     int oldSalary =employee.getSalaryInfo().
                    getAnnualSalary().intValue();
     employee.getSalaryInfo().setAnnualSalary( oldSalary+
                    salaryIncrease );
     con.sndRecord( employee );
     con.rcvRecord( employee );

     retValue="Salary for "+employee.getLastName()+"
              increased from "+oldSalary +" to "+
              employee.getSalaryInfo().getAnnualSalary();

     con.close();

  } // try
  catch (net.fsc.jca.communication.EISConnectionException
         eisEx) {
     if (con != null) {
        try {    con.close();
        } catch(Throwable thr) { }
     }
     throw new javax.ejb.EJBException
                        ("EISConnectionException:"+eisEx);
     } // catch EISConnectionException
     catch (de.siemens.cob2java.cobtypes.
            Cob2JavaException cobEx) {
        if (con != null) {
           try {    con.close();
           } catch(Throwable thr) { }
        }
        throw new javax.ejb.EJBException("Cob2JavaException:
                                         "+cobEx);
     } // catch Cob2JavaException
     return retValue;
   } // addSalary
}
```

14.5 Error messages and error handling

The following table provides an overview of the error messages that may be output by the tool Cobol2Java:

No	Errors	Error handling
1	`No compatible XSLT Processor found.`	Check whether all the required JAR files are present in the Cobol2Java `lib` directory.
2	`TransformerFactory-ConfigurationError`	See 1.
3	`Could not create file <name>`	Please make sure that you possess the necessary data access authorizations for the data medium.
4	`No Record/Field <name> found in specified XML document`	The name of the data structure is incorrect. Please check.
5	`WARNING! Multiple occurrence of <name>`	The document contains multiple structures with the specified name. In this case, there is no generation.

If a data element that is not supported by Cobol2Java is encountered then a generic data element of type `PicU` is generated. Developers can access the information relating to this element by means of `getBytes/setBytes`.

Error handling in the Cobol2Java classes is based on COBOL and is extremely tolerant. Input data longer than the field it is to be stored in is truncated to the destination field length. Invalid inputs, for example the input of an invalid number, result in the generation of a `NumberFormatException`.

Glossary

access point

See service access point.

ACID properties

Acronym for the fundamental properties of a transaction: atomicity, consistency, isolation and durability.

advanced program-to-program communication (APPC)

Another name for the LU6.2 protocol and the underlying architecture.

APPC

See advanced program-to-program communication (APPC).

appender

Log4j message destination. The logging messages transferred to a logger are output by the appender(s) assigned to the logger.

application entity

An application entity represents all the aspects of a real application which are relevant to communications. An application entity is identified by a globally unique name ("globally" is used here in its literal sense, i.e. worldwide), the application process title (APT). Every application entity represents precisely one application process. One application process can encompass several application entities.

application entity qualifier (AEQ)

According to the OSI standard, the application entity qualifier identifies a service access point within an application.

application entity title

An application entity title is a globally unique name for an application entity ("globally" is used here in its literal sense, i.e. worldwide). It is made up of the application process title (APT) of the relevant application process and the application entity qualifier (AEQ).

application process

The application process represents an application in the OSI reference model. It is uniquely identified globally by the application process title (APT).

application process title (APT)

According to the OSI standard, the application process title is used for the unique identification of applications on a global (i.e. worldwide) basis.

association (OSI)

An association is a communication relationship between two application entitys.

application server

An application server (Java EE Server) is the basic component of an EJB architecture. The application server offers the services of enterprise applications to the EJB clients.

application server cluster

Cluster consisting of multiple application servers. In such a cluster, n instances of the application server can be assigned to m proxy instances.

asynchronous job

Job carried out by the job receiver at a later time. Processing is carried out independently of the job submitter.

asynchronous service

A service in openUTM which processes a background job.

authentication

See system access control.

authorization

See data access control.

Basic Communication Access Method (BCAM)

BCAM forms the basis for the data communication system for BS2000 hosts or in the BeanConnect proxy container.

basic conversation

A basic conversation is a type of APPC conversation in which the CICS application must add control bytes to the application data for transmission to the partner.

BCAM

see Basic Communication Access Method (BCAM).

BeanConnect Management Console

The BeanConnect Management Console is the tool for administering and configuring the BeanConnect components. It provides a graphical user interface and a command line interface.

BeanConnect proxy

A BeanConnect proxy is the BeanConnect component which communicates with the resource adapter within the application server as well as with the EIS.

A BeanConnect proxy consists of a proxy container based on the transaction monitor openUTM.

During communication with an EIS partner of type CICS, a BeanConnect proxy consists additionally of an openUTM-LU62 Gateway and the communication service.

BeanConnect proxy container

The BeanConnect proxy container is the core of the BeanConnect proxy. The proxy container comprises the definitions of the objects used for outbound communication and inbound communication and the configuration of the communication partner (resource adapter and EIS partners).

BeanConnect resource adapter

The BeanConnect resource adapter is the BeanConnect component that is running on the application server and that implements the JCA-compliant interfaces. In the case of outbound communication via the OSI-TP protocol and in the case of outbound communication, the BeanConnect proxy is additionally needed for the connection between the EIS and the resource adapter.

CCI

See common client interface (CCI).

CICS

See Customer Information Control System (CICS).

cluster

A cluster is a number of networked computers that are seen as a single computer from the outside for many purposes. As a rule, the individual elements in a cluster are connected to each other over a fast network. The aim of establishing a cluster is to increase CPU capacity or availability.

See also application server cluster and proxy cluster.

CMX

See Communications Manager for UNIX Systems (CMX).

common client interface (CCI)

CCI is a component of the Java EE connector architecture (JCA) and provides an EIS independent client API for accessing EISs.

communication service

Service required by a BeanConnect proxy for communication between the proxy container and a CICS partner. The communication service works directly with the openUTM-LU62 Gateway and is provided by the IBM Communications Server or SNAP-IX independently of the platform. SNAP-IX or the IBM Communications Server represent software prerequisites for BeanConnect if communication is to be performed with CICS partners. However, they are not supplied with BeanConnect.

Communications Manager for UNIX Systems (CMX)

CMX is the basic product for communication software running on operating systems like Solaris. A license is required for its use.

component

Reusable software unit with standardized interfaces that can usually be manipulated in a development environment.

configuration property

Configuration properties are used to configure the resource adapter. A configuration property is set in a configuration file by means of XML tags.

connection factory

A connection factory is used by an EJB of an application server to open a connection to an external data source (EIS). The connection factory is provided when the application server is started in the JNDI name directory.

connection pooling

Connection pooling manages connections that are expensive to create and destroy. Connection pooling is used to improve scalability and performance in an application environment.

connector architecture

See Java EE connector architecture (JCA).

container.properties

Configuration file of a BeanConnect proxy. Many of the changes you make using the BeanConnect Management Console are saved in the `container.properties` file.

contention winner / contention loser

Each connection between two partners is managed by one of the partners. This partner is called the contention winner, while the other partner is referred to as the contention loser. Jobs can be initiated by both partners. If both partners submit a job at the same time, priority is given to the contention winner.

control point (CP)

The control point is responsible for managing the end node and its resources in an APPN (advanced peer-to-peer-networking) network.

conversation

A conversation is a logical connection between two transaction programs in an LU6.2 session. A conversation begins with Allocate and ends with Deallocate. A conversation allocates a session for its entire life and locks out all other users.

CP

See control point (CP).

Customer Information Control System (CICS)

CICS is the IBM transaction monitor. CICS is available for various platforms such as CICS/ESA for the z/OS operating system. An OLTP application based on CICS is called a CICS application.

data access control

The application server checks whether a communication partner/client is authorized to access a particular business method. The access rights are set as part of the configuration.

data link control (DLC)

Data link control is the service provided by the data link layer. The data link layer corresponds to layer 2 of the Open Systems Interconnection model for network communication (LAN, Enterprise Extender for example).

deployment descriptor

A deployment descriptor is a file in XML format that is used for the deployment of EJBs or resource adapters. A deployment descriptor provides configuration information that is not contained in the EJB code or the resource adapter code.

dialog service

Service which processes a job interactively (synchronously) in conjunction with the job submitter. A dialog service processes dialog messages received from the job submitter and generates dialog messages to be sent to the job submitter. A dialog service comprises at least one transaction. In general for openUTM, a dialog service encompasses at least one dialog step.

dialog step

A dialog step starts when a dialog message is received by the openUTM application. It ends when the openUTM application responds.

distributed program link (DPL)

DPL is a program interface that enables a CICS program to call another CICS program which may reside on a remote CICS system. DPL is like calling a subprogram.

distributed transaction

See global transaction.

distributed transaction processing (DTP)

with openUTM partners:

Transaction-oriented distributed processing with global transactions. Distributed processing means that jobs are processed by several different applications.

with CICS partners:

Distributed transaction processing is a programming interface that enables a CICS transaction to invoke another CICS transaction (possibly in another CICS system). DTP supports the handling of global transactions. It is similar to the client-sever programming model.

DLC

See data link control (DLC).

DPL

See distributed program link (DPL).

DTP

See distributed transaction processing (DTP).

EIS

See Enterprise Information System (EIS).

EJB

See Enterprise JavaBeans (EJB).

EJB container

Runtime environment for EJB components. It is embedded in an application server.

Enterprise Information System (EIS)

This is a term used to describe external data sources such as ERP systems (Enterprise Resource Planning systems such as SAP), OLTP applications such as openUTM applications/CICS applications or database systems such as Oracle DB, SESAM, UDS.

An EIS that communicates with the application server by means of BeanConnect is called an EIS partner.

Enterprise JavaBeans (EJB)

Enterprise JavaBeans^{TM} (EJB) is a component technology that allows the development of cross-platform, multitier, distributed server applications within a modular architecture.

function unit commit

A function group in the OSI-TP protocol that is required to create distributed transactions. Whether or not the functional unit commit may be used is negotiated when an association is set up between the two partners. OSI-TP dialogs can run with or without the functional unit commit in an association in which the functional unit commit was agreed to. An association is a communication relationship between two applications.

function unit handshake

A function group in the OSI-TP protocol that can be used by the communication partners to coordinate the processing of a dialog at application level. This function makes it possible to request processing confirmations and send positive or negative confirmations. No inter-application transaction management is linked to this function.

global transaction

A global transaction is a transaction extending over more than one application.

IBM Communications Server

IBM Communications Server is an IBM product that connects applications in SNA networks with applications in TCP/IP networks.

In BeanConnect, the IBM Communications Server is used for communication between a BeanConnect proxy and a CICS partner application using the LU6.2 protocol on Linux and Windows systems.

IDE

Integrated Development Environment. In the case of BeanConnect, this is the open source development environment NetBeans IDE.

inbound communication

Inbound communication is communication from an EIS to a Java EE application server.

inbound message endpoint

An inbound message endpoint is an endpoint of the inbound communication within the Java EE application server.

For each inbound message endpoint in the application server, an identically named inbound message endpoint must be configured in the BeanConnect proxy by means of the BeanConnect Management Console. A BeanConnect proxy may have several endpoints.

inbound service

Inbound services represent the objects addressed by the EIS partners during inbound communication. The inbound services known to the Management Console are defined implicitly by the inbound message endpoints.

Exactly one inbound message endpoint is assigned to each inbound service.

inbound user

An inbound user is a user name and password that can be delivered from the EIS to the BeanConnect proxy during inbound communication.

J2EE®

Old name for Java EE.

Java EE

Java Platform, Enterprise Edition, abbreviated to Java EE (previously J2EE), is the specification of a software architecture for the transaction-based execution of applications programmed in Java.

Java EE connector architecture (JCA)

Java EE connector architecture

Defines a standard architecture for connecting the Java EE platform to heterogeneous Enterprise Information Systems.

Java Development Kit (JDK)

Standard development environment from Sun Microsystems for developing applications written in Java.

Java Naming and Directory Interface (JNDI)

JNDI is a standard Java extension that provides a uniform API for accessing the directory and naming services of different vendors.

JCA

See Java EE connector architecture (JCA).

JDK

See Java Development Kit (JDK).

JMX

Java Management Extensions (JMX) is a specification developed by the Java Community Process (JSR-3) for managing and monitoring Java applications.

JMX client

Client that can access a JMX server and make use of its services. In BeanConnect, the Management Console represents the JMX client implementation.

JMX server

Instance in a Java application that provides services for monitoring the application. In a BeanConnect environment, the application server implements the JMX server functionality.

JNDI name

Name of a Java object in a programming environment in which the Java Naming and Directory Interface (JNDI) is used.

job-receiving service

A job-receiving service is a service started by a job-submitting service of another server application.

job-submitting service

A job-submitting service is a service which requests another service from a different server application (job-receiving service) in order to process a job.

Jython

Jython (formerly JPython) is a pure Java implementation of the Python programming language and therefore makes it possible to execute Python programs on all Java platforms. Python is a universal, usually interpreted, high-level programming language.

KDCA

Default name of the KDCFILE.

KDCDEF

openUTM generation tool resp. generation tool of the BeanConnect proxy container.

KDCFILE

Configuration file of a BeanConnect proxy container. The file contains data required by the container application for execution. The file is created with the generation tool KDCDEF.

with openUTM partners:

Configuration file of an openUTM partner application. The file contains data required by the openUTM partner application for execution. The file is created with the openUTM generation tool KDCDEF.

KDCS

Universal openUTM program interface compliant with the national DIN 66 265 standard and which includes some extensions. KDCS (compatible data communications interface) allows dialog services to be created, for instance, and provides calls for distributed processing. In BeanConnect, KDCS is only used as an internal interface.

Log4j

BeanConnect uses the software product Log4j to provide trace and logging functionality. Log4j is a component of the Apache Jakarta project. Log4j provides interfaces for logging information (runtime information, trace records etc.) and for configuring the log output.

Log4j is configured by means of the BeanConnect Management Console.

logger

A logger is a Log4j message source. Programs that must write logging information retrieve logger objects with predefined names from Log4j and output their messages via these objects. The destination to which Log4j sends the messages is transparent to the program.

logical unit (LU)

A logical unit is a logical virtual port which provides a user with access to network services in an SNA network. The logical unit corresponds with the control point (CP) and a partner logical unit representing, for example, a user program.

LU

See logical unit (LU).

LU6.2 protocol

The LU6.2 protocol is a component of the IBM network. LU6.2 defines methods for program-to-program communication between applications on different computers.

Management Console Command Line Interface (MC-CLI)

Management Console command line interface (MC-CLI)

Set of Jython which can be used to start BeanConnect Management Console functions from a Java script.

mapped conversation

A mapped conversation is a type of APPC conversation in which the data passed to and received from another APPC application is simply user data. The user is not concerned with the internal data formats demanded by the architecture.

MBean

Managed Bean which represents a resource of a JMX server.

MC-CmdHandler

The MC-CmdHandler is a BeanConnect component which is required in order to administer another, remote BeanConnect component using the BeanConnect Management Console. In this case, the function scope is the same as when administering a local component.

message

A message is a data packet that consists of a header and a body. The header contains addressing data, the network routing and possibly data on the message format. The body contains the actual message in the form of business data or system messages.

message-driven bean

A message-driven bean is a component of the Enterprise JavaBeans specification from Sun Microsystems.

Message-driven beans are beans for the reception of messages. Message-driven beans are only called by the EJB container and therefore have no home or remote interfaces.

message endpoint

message endpoint

A message endpoint is a message-driven bean application deployed in an application server.

message listener

message listener

Message consumer. The message listener object is sent messages as soon as they become available. Message-driven beans are message listeners.

message listener interface

message listener interface

Interface that a message listener must implement. Inbound resource adapters provide specific message listener interfaces which message listeners must implement if they are to consume messages from this resource adapter.

mode name

The mode name is a symbolic name for a list of session properties and is required for all interconnection requests over SNA. The mode name is used by the initiator of a session.

multiple resource adapter mode

Configuration in which a BeanConnect proxy interacts with multiple resource adapters. These resource adapters may be located on different application servers.

multi-step transaction

A multi-step transaction is a transaction which comprises more than one dialog step.

naming

Mapping of names to object references. The mapping is usually performed via a naming service.

network name

A network name is a name identifying an SNA network and is a component of the logical unit name, see logical unit (LU).

OLTP message-driven bean

OLTP message-driven beans are EJBs that receive and process jobs from OLTP applications (openUTM/CICS applications). The OLTP application addresses the OLTP message-driven bean via an inbound message endpoint.

openUTM

Transaction monitor from Fujitsu Siemens Computers and basic component of the BeanConnect proxy container.

openUTM application

An OLTP application based on the transaction monitor openUTM of Fujitsu Technology Solutions.

openUTM-LU62 Gateway

openUTM-LU62 is a BeanConnect proxy component which acts as the interconnection with partner applications supporting the SNA protocol LU6.2, particularly with CICS applications.

openUTM socket protocol (USP)

A protocol used by openUTM partner applications to convert byte streams into messages that require the TCP/IP transport system.

Oracle WebLogic Server

Java EE 6-compliant application server from Oracle Corporation.

OSI-LPAP partner

OSI-LPAP partners are the addresses of the OSI-TP partners generated in the BeanConnect proxy container. In the case of distributed processing via the OSI-TP protocol, an OSI-LPAP partner for each partner application must be configured in the proxy container. In the case of openUTM partners, the OSI-LPAP partner in the proxy container represents the partner application and in the case of CICS partners, it mirrors the openUTM-LU62 Gateway instance. During communication, the partner application is addressed by the name of the assigned OSI-LPAP partner and not by the application name or address.

OSI reference model

The OSI reference model provides a framework for standardizing communications in open systems. ISO, the International Organization for Standardization, described this model in the ISO IS7498 standard. The OSI reference model divides the necessary functions for system communication into seven logical layers. These layers have clearly defined interfaces to the neighboring layers.

OSI-TP

Open **S**ystem **I**nterconnection **T**ransaction **P**rocessing.

A communication protocol defined by ISO for distributed transaction processing.

A partner of an application that communicates with the BeanConnect proxy container via the OSI-TP protocol is called an OSI-TP partner.

In the case of openUTM partners this is the EIS partner and in the case of CICS partners it is the openUTM-LU62 Gateway.

With CICS, this protocol is used for communication between the BeanConnect proxy container and the openUTM-LU62 Gateway.

OSS

OSI **S**ession **S**ervice

OSS forms the basis for OSI-TP data communication in the BeanConnect proxy container.

outbound communication

Outbound communication is communication from the Java EE application server to the EIS.

outbound communication endpoint

An outbound communication endpoint is a symbolic name representing a service of the partner EIS.

outbound service

An outbound service object describes a service (transaction code) inside the EIS partner for outbound communication.

PCMX

The basis for communication software which runs on the Solaris, Linux and Windows operating systems.

physical unit (PU)

Every node in an SNA network contains a physical unit as an addressable SNA instance. Before two logical units (LUs) can open a communication relationship in the SNA network, a communication relationship must first be opened between the corresponding PUs.

proxy cluster

Cluster consisting of more than one BeanConnect proxy and which is administered via the BeanConnect Management Console.

proxy container

See BeanConnect proxy container.

PU

See physical unit (PU).

resource adapter

Resource adapters (also referred to as connectors) connect the application server to an EIS, see also BeanConnect resource adapter.

resource manager

Resource managers (RMs) manage data resources. Database systems are examples of resource managers.

RFC1006

A protocol defined by the IETF (Internet Engineering Task Force) belonging to the TCP/IP family that implements the ISO transport services (transport class 0) based on TCP/IP.

schema

see XML Schema.

service

Services process the jobs that are sent to a server application. Services can be requested by clients or by other servers. A service of an openUTM application or of a CICS application comprises one or more transactions. The first transaction is called with the service TAC or transaction program name.

With openUTM, there are two types of services: dialog services and asynchronous services. openUTM provides the program units of a service with common data areas.

service access point

In the OSI reference model, a layer has access to the services of the layer below at the service access point. In the local system, the service access point is identified by a selector. During communication, the openUTM application links up to a service access point. A connection is established between two service access points.

session

A session is understood to be a communication relationship between two LUs, more generally between two addressable SNA instances.

single-step transaction

transaction which encompasses precisely one dialog step.

SNA

See Systems Network Architecture (SNA).

SNAP-IX

SNAP-IX is a product from Data Connection that connects applications in SNA networks with applications in TCP/IP networks.

In BeanConnect, SNAP-IX is used for communication between a BeanConnect proxy and a CICS partner application using the LU6.2 protocol on Solaris systems.

synchronization level (sync-level)

Designation in LU6.2 that characterizes the transaction security for distributed processing:

- For sync-level 0 (none), only net data and error messages may be sent. Acknowledgments are not allowed.
- For sync-level 1 (confirm), simple acknowledgments may also be sent in addition to net data and error messages.
- For sync-level 2 (sync point), full transaction security is activated for distributed transactions.

synchronization point (sync point)

A logical point within the flow of a distributed process at which the common resources are brought to a defined state. The term "end of transaction" is used instead in openUTM.

system access control

This involves the application server checking whether a user ID is authorized to work with the application server.

Systems Network Architecture (SNA)

SNA is the designation for a series of communication protocols defined by IBM.

TAC

See transaction code (TAC).

transaction

Processing section within a service which has the ACID properties. If, during the course of a transaction, changes are made to the application information, they are either made consistently and in their entirety or not at all (all-or-nothing rule). The end of the transaction forms a synchronization point (sync point).

transaction code (TAC)

Name by means of which a service of an openUTM application can be called.

user ID

Identifier for a user defined in the configuration for the openUTM / CICS application (with an optional password for system access control) and to whom special data access rights (data access control) have been assigned. A client must specify this ID (and any password which has been assigned) when signing on to the openUTM application/CICS application.

USP

See openUTM socket protocol (USP).

UTM

See openUTM.

UTM application

See openUTM application.

Virtual Telecommunications Access Method (VTAM)

The component in an IBM host system that is responsible for remote data processing.

Web Service Description Language (WSDL)

The Web Services Description Language (WSDL) defines a platform, programming language and protocol-independent XML specification for the description of network services (Web services) for message exchange.

XML

XML (e**X**tensible **M**arkup **L**anguage) is a metalanguage standardized by W3C (WWW Consortium) in which the interchange formats for data and the associated information can be defined.

XML Schema

XML Schema is a W3C recommendation for the definition of XML document structures. The structure is described in the form of an XML document. In addition, a large number of data types are supported.

Index

A
aborted application run 520
address an EIS application 395
administrable proxy 154
administration 255
  functions 160
  MC-CmdHandler 236
  user password 175
Ant 565, 566, 571, 586
appender 462
  BeanConnectLoggingFile 463, 485, 491
  BeanConnectLoggingFileXML 486
  BeanConnectManagementConsole 487
  BeanConnectMCSocketAppender 487
  BeanConnectShortLoggingFile 484
  BeanConnectSysout 490
  BeanConnectSysoutShort 483, 490
  configuring 468
application process title 184
application recommendations
  BeanConnect-specific interfaces or CCI 380
application run aborted 520
application server
  deployment descriptor 116, 131
  generating statistics 284
  monitoring with MBean clients 279
application-managed authentication 110, 396
associated connections 392
asynchronous communication 39, 391, 402, 411
asynchronous request
  durability 455
  inbound communication 456
authentication
  application-managed 110, 396
  container-managed 110, 396
  user ID and password 396
availability
  EIS partners 277
  MC-CmdHandler 276
  resource adapters 273, 275
availability of the proxy 271

B
BCAM configuration 250
BCAM trace 497
BeanConnect
  communication variants 36
  components 17, 28
  features 30
  installing under Linux 52
  installing under Solaris 44
  installing under Windows 60
  target group 18
BeanConnect Management Console siehe Management Console
BeanConnect proxy see proxy
BeanConnect resource adapter see resource adapter
BeanConnect tools 28, 82
  uninstalling 82
BeanConnect-specific interfaces
  application recommendations 380
  connection factory interfaces 382
  connection interfaces 383
  for inbound communication 415
  for outbound communication 382
  program framework for inbound communication 419
  program framework for outbound communication 399

beanconnect_i18n.properties 444
BeanConnect_Install.ini 67
BeanConnectI18N.jar 444
BS2000/OSD as EIS partner 250
BS2000/OSD COBOL application 563
bufferedIO, property 103
byte array 384
byte container 384

C
CCI 394
  application recommendations 380
  asynchronous communication 402
  Common Client Interface (CCI) 40
  connection factory interfaces 394
  connection interfaces 394
  dialog communication 400
  inbound communication 421
  outbound communication 394
  program framework for inbound communication 421
  program framework for outbound communication 400
  programming information 421
charset
  custom 441
check availability
  proxy cluster 271
CICS 23
  configuration 252
  input file 253
CICS application
  inbound communication 412
  outbound communication 397
CICS program
  invoking by another CICS program 398
cluster
  adding proxy 187
  displaying 186
  generating 185
  master 261
  removing 188
  removing proxy from 188
CMX
  installing under Linux 53
CMX trace 498
COB2XML.LIB 566, 567
COBOL application
  integrating with BeanConnect 563
COBOL clause
  supported by Cobol2Java 578
COBOL COPY element
  converting to XML 567
COBOL data structure
  converting to XML 568
COBOL data type
  mapping to Java class 567
  supported by Cobol2Java 578
COBOL program
  converting to XML 567
Cobol2Java 563
  code conventions 579
  conversion class 564
  directory structure 566
  error handling 590
  example 584
  formatted mode support 583
  generation procedure 564
  installing 565
  Java-EBCDIC conversion 583
  javadoc 566
  naming conventions 579
  programming reference 577
  supported COBOL types and clauses 578
  system requirements 565
Cobol2XML 564, 567
  example 585
  library 567
  system requirements 565
code conversion 393, 430
code samples
  inbound communication 423
  outbound communication 406
code table
  IBM 430
  OSD_EBCDIC_DF03_IRV 435
  OSD_EBCDIC_DF04_1 437
  OSD_EBCDIC_DF04_15 439

OSD_EBCDIC_DF04_DRV 431
predefined 433
standard 430, 433
user-defined 441
command line installation
proxy container 67
Common Client Interface see CCI
communication
asynchronous 39, 391, 411
dialog 39, 386, 411
inbound 37, 41, 410
non-transactional 40
outbound 37, 40, 219, 382
transactional 39, 391
communication methods 384
communication protocol 36
communication service 31
starting with the Management Console 257
configuration
activating 234
CICS 252
EIS 247
EIS partner 158, 198
error messages 513
files 166
proxy 157
saving 233
configuration file for EIS partner 216
connection
between BeanConnect and EIS partner 251
between BeanConnect and openUTM
via OSI-TP protocol 248
via RFC1006 protocol 250
via socket protocol 250
via UPIC protocol 249
connection factory 382
for deployment in the application server 117, 131
interfaces 382, 394
connection group see associated connections
connection interface 383
communication methods 384
inbound communication 415, 421
outbound communication 383
outbound communication (CCI) 394
connection management 25
connection pooling 109, 125
ConnectionFactory object 116
connections, associated 392
connectionURL, property 104
OSI-TP communication 104
UPIC communication 120
console.properties.xml 166
container-managed authentication 110, 396
custom charset 441

D
D.XMLCOPY 569
D.XMLPROG 568
data exchange
based on OltpMessagePart objects 387, 388
based on OltpMessageRecord objects 388, 390
DEBUG 461
deployment descriptor
application server 116, 131
EJB 116, 130, 381
message-driven bean 381
resource adapter 117, 131, 381
diagnosis 156
diagnosis support 162
diagnostics
dumps and diagnostic dumps 495
IBM Communications Server (Linux) 509
IBM Communications Server (Windows) 511
Log4j 460
Management Console 500
openUTM-LU62 Gateway 502
proxy container 490
resource adapter 481
SNAP-IX 507
stderr log 493
stdout log 493
dialog communication 39, 386, 400, 411
display
MBean 281
MBean attributes 282
DMS error 517, 561

return codes 561
DPL (Distributed Program Link) 398
DTP (Distributed Transaction Processing) 398
dumps 495

E
EBCDIC 430
EIS
configuration 247
EIS application
addressing 395
querying information 397
EIS partner 155
adding to the Management Console 199, 208
availability 277
configuration 158
configuration files 216
configuration via the Management Console 198
inbound communication connections 454
EJB
code file 116, 130
deploying 115, 129
deployment descriptor 116, 130
placing BeanConnect calls 395
ejb-jar.xml 116, 381
example 136
encoding 430
encoding, property
inbound 134
outbound 105, 122
encodingActive, property
inbound 135
outbound 106, 123
endConversation
XATMI outbound 405
Enterprise Java Bean see EJB
ERROR 461
error codes 561
error message
configuration 513
openUTM-LU62 Gateway 548
proxy 513
runtime 514
u62_adm 559
u62_gen 560
u62_sta 558
u62_start 548
u62_tp 549
error number, DMS errors 561
establish connection
to MBean server 280
expert mode 183

F
FATAL 461

G
general configuration property 87
generic work context 25
global configuration property
inboundListenerPort 91
proxyReconnectCount 146
proxyReconnectInterval 146
proxyURL 88
resourceAdapterAddresses 145
group, dump error code K060 520

H
high availability 447

I
IBM code tables 430
IBM Communications Server 31
diagnostics for Linux 509
diagnostics for Windows 511
starting with the Management Console 257
traces for Linux 509
inbound communication 37, 41, 93
BeanConnect-specific interfaces 415
CCI 421
CICS application 412
communication types 38
interfaces 41
non-openUTM application 414
openUTM application 411
program 410
selecting the interface to be used 381

inbound message endpoint 155
  adding to the Management Console 224
inbound programming
  XATMI partners 414
inbound user 155
inboundListenerPort, property 91
INFO 461
install
  BeanConnect under Linux 52
  BeanConnect under Solaris 44
  BeanConnect under Windows 60
  CMX under Linux 53
  Management Console under Linux 58
  Management Console under Solaris 50
  Management Console under Windows 65
  master installation under Linux 52
  master installation under Solaris 44
  master installation under Windows 60
  openUTM under Linux 54
  openUTM under Solaris 45
  openUTM under Windows 61
  parameter file 67
  PCMX under Solaris 45
  PCMX under Windows 60
  proxy container under Linux 56
  proxy container under Solaris 48
  proxy container under Windows 64
  proxy container via Windows command line 67
  resource adapter 69
installation
  updating under Linux 74
  updating under Solaris 72
  updating under Windows 76
installation program
  starting an update on Solaris systems 72
  starting under Linux 55
  starting under Solaris 47
  starting under Windows 60
  starting update under Linux 74
  starting update under Windows 76
interface 379
  BeanConnect-specific 382, 415
  for inbound communication 415
  for outbound communication 382
  javax.resource.cci 394
  javax.resource.cci.MessageListener 41, 421
  net.fsc.jca.communication 40
  net.fsc.jca.communication.cci 40
interface net.fsc.jca.communication.
  AsyncOltpMessageListener 41, 419
  EISConnection 383, 384
  EISConnectionByteArray 384
  EISConnectionByteContainer 384
  EISConnectionFactory 382
  EISConnectionString 385
  EISOltpConnection 383, 384, 385
  EISOltpConnectionFactory 382
  EISUpicConnection 383, 385
  EISUpicConnectionFactory 382
  EncodingDef 385
  OltpMessageListener 41, 419
interfaces and programming 379
internationalization 443

J

J2EE application server 24
Java class
  example for generating from COBOL XML file 586
  for internationalization 443
  generating with Ant 571
    example 574
    parameters 571
  generating without Ant 575
    example 576
    parameters 575
  used for COBOL type 577
Java EE
  application server 23
javax.resource.cci.Connection interface 394
javax.resource.cci.MessageListener interface 421
Jave EE
  Oracle WebLogic Server Connection Architecture 24
JCA 1.5 specification 25
JCA contract 25

Common Client Interface 26
connection management 25
generic work context 25
lifecycle management 25
message inflow 25
security management 25
security work context 26
transaction inflow 25
transaction management 25
work management 25

K

K messages 516
K009 516
K017 516
K036 516
K040
pagepool warning level 517
K041 517
K043 517
K049
start error 518
K055 519
K060 520
K065 524
K075 524
K078 525
K104 526
K119 528
K124 529
K128 530
K135 531
K139 531
K147 531
K152 532
K160 532
K204 533
K210 533
K211 533
K212 533
K213 533
K214 533
K215 533
K216 534
K217 534
K218 534
K220 534
K221 534
K222 534
K223 534
K224 534
K225 535
K230 535
K231 535
KDCDEF
statements for OSI-TP communication 248
statements for socket or RFC1006 connection 250
statements for UPIC communication 249
kdcshut 290

L

language
changing via system properties 443
for message output 443
of online help system 151
lifecycle management 25
LINK command
parameter COMMAREA 398
parameter LENGTH 398
parameter PROGRAM 398
parameters 398
Linux
installing a proxy container 56
installing BeanConnect 52
installing CMX 53
installing Management Console 58
installing openUTM 54
installing packages 52
uninstalling BeanConnect 79
update installation 74
listener port 470
LMS library 567, 568, 569
localization 443
log window 152
Log4j 460
appender 462
configuration file in XML format 469

configuring appenders 468
configuring loggers 467
level 461
listener port 470
logger 460
rolling file appender 463
root logger 460
suppressing messages 473, 474
XML logging file 475
log4j.properties.xml 166
logger 460
activating/deactivating the display of messages 473
BeanConnect 488, 492
BeanConnect.c 492
BeanConnect.Datasources.OLTP 492
BeanConnect.in 488
BeanConnect.info 488, 492
BeanConnect.kdcs 492
BeanConnect.out 488
BeanConnect.ui 488
configuring 467
level 461
name space 460
net.fsc 488, 492
net.fsc.beanta.encoding 488
net.fsc.beanta.encoding.EncoderImpl 492
net.fsc.tpbasics.util.L 488, 492
root logger 492
logging
Log4j 460
proxy 493
resource adapter 481
logging configuration, predefined 490
logging file (XML) 475
logging files 463
logging level 474
logging output 472
example 472
LU6.2 protocol 36

M

Management Console 17, 149, 256
administration functions 160
advanced features 156
as Log4j socket reader 470
diagnosis support 162
diagnostics 500
expert mode 183
installing under Linux 58
installing under Solaris 50
installing under Windows 65
language 151
logging file (XML) 475
managed objects 154
MC-CmdHandler 235
message output 443
navigation area 152, 153
outputting messages 472
overview 32
protocol window 152
proxy navigation tree 170
shutting down 151
starting online help 150
starting under Solaris and Linux 150
starting under Windows 150
status bar 152
todo topics 156, 162
user interface 152
work area 152
manual extension file 18
master
proxy cluster 261
master installation
under Linux 52
under Solaris 44
under Windows 60
MBean
displaying 281
operations 289
subscribing to notifications 286
MBean attributes
displaying 282
modifying 283
MBean clients 279
MBean server
establishing a connection 280
MC-CLI 17, 149, 255

logging 500
MC-CmdHandler 32, 235
administering 236
availability 276
installing 71
security and privileges 235
shutting down 237
starting 236
uninstalling 82
message
BeanConnect proxy container 513
u62_adm 559
u62_gen 560
u62_sta 558
u62_start 548
u62_tp 549
message file 444
message inflow 25
message length
XATMI outbound 405
message listener interface 37, 41
message output
determining the language 443
message types 514
message-driven bean see OLTP message-driven bean
messageEndpoint, property 133
messages
of the proxy 472
outputting 472
suppressing 473, 474
messaging-type, property 133
modify MBean attributes 283
monitor
application server with MBean clients 279

N
navigation area 152, 153
net.fsc.jca.communication 40
net.fsc.jca.communication.cci 40
non-openUTM application
inbound communication 414
non-transactional communication 40
notational conventions 22
notification
from MBeans 286
subscribing 287

O
OFF
logging output 461
OLTP Message-Driven Bean
Deployment 132
OLTP message-driven bean 410
asynchronous communication 411
code samples 423
dialog communication 411
programming information 415, 421
properties 132
OltpMessage object 388
OltpMessagePart object 387, 388
OltpMessageRecord object 390
online help
starting 150
online help system, language 151
openSEAS 17
openUTM 23, 30, 248
installing under Linux 54
installing under Solaris 45
installing under Windows 61
openUTM application
inbound communication 411
openUTM dump 495
openUTM partner 248
configuring in proxy 199
openUTM socket protocol 38, 251
openUTM-LU62 Gateway 31
administering 267
diagnostic information 505
diagnostics 502
displaying status information 268
error messages 548
starting 267
starting via the Management Console 257
stopping 268
traces 502
operate the proxy 255
operations on MBeans 289

Oracle WebLogic Server 27
  Java EE Connection Architecture 24
orion-ejb-jar.xml 116
OSD_EBCDIC_DF03_IRV 435
OSD_EBCDIC_DF04_1 437
OSD_EBCDIC_DF04_15 439
OSD_EBCDIC_DF04_DRV 431
OSI-SCRATCH-AREA 457
OSI-TP error 528
OSS trace 496
outbound communication 37, 40
  BeanConnect-specific interfaces 382
  CCI 394
  CICS application 397
  code samples 406
  configuring with the Management Console 219
  connection factory interfaces 382
  connection factory interfaces (CCI) 394
  connection interface 383, 394
  program framework 399
  programming 382
  programming information 395
  selecting the interface to be used 381
outbound communication endpoint 155, 219
  configuring in the Management Console 222
outbound programming
  XATMI partners 405
outbound service 155, 219
  configuring in the Management Console 219
outputting messages 470

P
P messages 536
package net.fsc.jca.communication 40
pagepool area 453
parameter file for installation 67
password for authentication 396
PCMX
  installing under Solaris 45
  installing under Windows 60
PENDER 522
place BeanConnect calls in an EJB 395
port number 49, 58
port number interval, proxy container 64
predefined code table 433
privileges of MC-CmdHandler 235
process workload,workload 450
program 379
  inbound communication 410
  outbound communication 382
program framework
  AsyncOltpMessageListener, OltpMessageListener 419
  BeanConnect-specific interfaces 399
  inbound communication 419
  inbound communication (CCI) 421
  javax.resource.cci.MessageListener interface 421
  outbound communication 399
  outbound communication (CCI) 400
programming information 395
  OLTP message-driven bean 415
  OLTP message-driven Beans (CCI) 421
  outbound communication 395
properties.xml
  resource adapter 481
properties_debug.xml
  resource adapter 481
properties_default.xml
  resource adapter 481
properties_error.xml
  resource adapter 481
property
  bufferedIO 103
  connectionURL 104, 120
  displayName 104
  encoding (inbound) 134
  encoding (outbound) 105, 122
  encodingActive (inbound) 135
  encodingActive (outbound) 106, 123
  inboundListenerPort 91
  logLevel 107
  messageEndpoint 133
  messaging-type 133
  proxyReconnectCount 146
  proxyReconnectInterval 146
  proxyURL 88

- reconnectThreshold 125
- redeliveryThreshold 135
- resourceAdapterAddresses 145
- RevisionNumber 92
- timeout 107, 124, 125
- transactional 108
- transactionLogDir 91
- transactionLogging 89

protocol window 152

proxy 17, 154
- adding to Management Console 170
- adding to Management Console automatically 172
- administrable 154
- availability 271
- BCAM trace 497
- CMX trace 498
- components 31
- configuration steps 168
- configuring 157, 167
- context menu 256
- diagnostics 490
- error messages 513
- functions 30
- general information 173
- ID 174
- log files 493
- Log4j socket reader 470
- managing via the Management Console 256
- message output 443
- name in the Management Console 174
- navigation tree 170
- operating 255
- OSS trace 496
- outputting messages 472
- performance settings 183
- predefined logging configuration 490
- removing from the Management Console 172
- restarting via a local script 264
- restarting via the Management Console 161, 259
- restarting via the Windows program group 264
- shutting down 160
- starting 160
- starting after abnormal termination 263
- starting as Windows service 259, 263
- starting via a local script 262
- starting via the Management Console 257
- starting via the Windows program group 262
- stopping as Windows service 265
- stopping in cluster 290
- stopping via a local script 265
- stopping via the Management Console 259
- stopping via the Windows program group 265
- timer settings for connection surveillance 183
- traces 496

proxy cluster
- checking availability 271
- defining 185
- master 261
- removing 188
- synchronizing 261

proxy component
- configuring 173
- starting via the Management Console 257

proxy container
- asynchronous processing 455
- cache 453
- high availability 447
- installing under Linux 56
- installing under Solaris 48
- installing under Windows 64
- installing under Windows via command line 67
- number of processes 450
- OSI-SCRATCH-AREA 457
- pagepool area 453
- port number 64
- semaphores 458
- shared memory 448
- starting via the Management Console 257
- workload 450

proxyReconnectCount, property 146

proxyReconnectInterval, property 146

proxyURL, property 88

R
ra.xml
  Beispiel 97
  example 96
  setting the global configuration properties 87
RAR file
  declaring 94
reconnectThreshold, property 125
redeliveryThreshold, property 135
Resource Adapter
  Deployment 94
resource adapter 17, 24, 154
  availability 273, 275
  configuration properties for outbound communication 103, 120
  configuring appenders 468
  configuring loggers 467
  deployment descriptor 117, 131
  diagnostics 481
  extend logging 482
  functions 30
  installing 69
  logging 481
  logging output 470
  message output 443
  predefined logging configuration 483
  switching to other proxy 290
  undeploying 96
  update deployment 95
resource types 115, 129
resourceAdapterAddresses, property 145
restart proxy 161
RFC1006 application 251
RFC1006 connection
  KDCDEF statements 250
RFC1006 protocol 38
rolling file appender 463
root logger 460
runAnt.bat (Windows system) 566, 573
runAnt.sh (UNIX system) 566, 573
runtime
  error messages 514

S
scalability 447
script
  change 264
  shutcontainer 265
  startcontainer 262
security
  MC-CmdHandler 235
security management 25
security settings
  for connecting to EIS 109
security work context 26
select the interfaces to be used 381
semaphores
  number 458
shared memory 448
shut down
  Management Console 151
  MC-CmdHandler 237
SNA daemon 269
SNA implementation 31
SNAP-IX 31
  diagnostics 507
  starting with the Management Console 257
  traces 507
socket connection
  KDCDEF statements 250
socket listener port 470
socket reader
  Log4j 470
Solaris
  installing a proxy container 48
  installing BeanConnect 44
  installing Management Console 50
  installing openUTM 45
  installing packages 44
  installing PCMX 45
  uninstalling BeanConnect 78
  update installation 72
standard code table 430, 433
standard conversion 430, 433
start
  Management Console 150

Management Console under Solaris and Linux 150
Management Console under Windows 150
MC-CmdHandler 236
online help 150
proxy 160
statistical information 156
statistics
for the application server 284
statistics collector 284
status bar 152
stderr 493
stdout 493
stop
proxy in cluster 290
stop proxy 160
with wait time 290
string 385
structure of the documentation 18
structure of the manual 19
subscribe
notifications 287
subscribe notifications 287
switch
resource adapter to other proxy 290
synchronize.
proxy cluster 261
SYSLOG 494
system error codes
ERRNO 562
system log file 494

T

timeout, property 107, 124, 125
todo topics 156, 162
tools 17
uninstalling 82
TPOOL statement 249
TRACE 461
traces
BCAM trace 497
CMX trace 498
IBM Communications Server (Linux) 509
openUTM-LU62 Gateway 502
OSS trace 496
proxy 496
SNAP-IX 507
transaction inflow 25
transaction management 25
transaction monitor openUTM 30
transaction-support 126
transactional communication 39, 391
transactional, property 108
transactionLogDir, property 91
transactionLogging, property 89
TRMA 520
TSEL-FORMAT statement 249
typed buffer 405
types, of messages 514

U

U messages 543
u62_adm
messages 559
u62_gen
messages 560
u62_sta
messages 558
u62_start
messages 548
u62_tp
messages 549
Unicode 430, 433
uninstall 82
BeanConnect under Linux 79
BeanConnect under Solaris 78
BeanConnect under Windows 80
MC-CmdHandler 82
update installation
under Linux 74
under Solaris 72
under Windows 76
UPIC application 251
UPIC message 531
UPIC protocol 38
user ID for authentication 396
user interface 152
user-defined code table 441

V

VTAM
  input file 253

W

WARN 461
weblogic-ra.xml 93, 381
  example 111
weblogig-ra.xml
  example (UPIC) 126
Windows
  installing a proxy container 64
  installing a proxy container via command line 67
  installing BeanConnect 60
  installing Management Console 65
  installing openUTM 61
  installing PCMX 60
  master installation 60
  uninstalling BeanConnect 80
  update installation 76
work area 152
work management 25

X

XATMI partners
  configure 218
  configure proxy for 184
  inbound programming 414
  outbound programming 405
XML
  logging file 475
XML files 166
XML format
  configuration file 469
XSLT stylesheet 564, 566

List of Examples

Example 1: Installing multiple packages 45

Example 2: Installing multiple packages 53

Example 3: Configuration properties in the file ra.xml 96

Example 4: Configurations- properties in the weblogic-ra.xml file 97

Example 5: Connection pooling 109

Example 6: Configuration properties in the file weblogic-ra.xml 111

Example 7: Configuration properties in the file weblogic-ra.xml 126

Example 8: ejb-jar.xml file 136

Example 9: weblogic-ejb-jar.xml file 139

Example 10: Example of an URL 249

Example 11: Verifying that the OSI-TP protocol is used for communication (openUTM partner) 383

Example 12: Information on the conversation with the EIS application 397

Example 13: Dialog communication using the EISConnection interface 406

Example 14: Dialog communication using the CCI 407

Example 15: Associated connections using the EISConnectionGroup interface 408

Example 16: OLTP message-driven bean for dialog communication 423

Example 17: OLTP message-driven bean for asynchronous communication 425

Example 18: OLTP message-driven bean (CCI) 426

Example 19: Using predefined code tables 434

Example 20: Logger name space 461

Example 21: Logging level 461

Example 22: Output of logging events through the appender 462
Example 23: Logging files 464
Example 24: Entries in the LogWriter file. 480
Example 25: Control of logging output. 489
Example 26: Example for a cobol2java.properties file 574
Example 27: Generating Java classes. 576
Example 28: Name assignment for the different code conventions 579
Example 29: Sending and receiving data 582
Example 30: Getting and storing data 582

List of Figures

Figure 1: Oracle WebLogic Server – Java EE Connection Architecture 27

Figure 2: BeanConnect components 29

Figure 3: BeanConnect components for outbound communication with openUTM partners via UPIC protocol. 29

Figure 4: BeanConnect proxy components of a CICS proxy 31

Figure 5: Relationships between the BeanConnect components in standard operation and without a cluster 34

Figure 6: BeanConnect components in multiple resource adapter mode with 2 resource adapters and one proxy 35

Figure 7: Outbound communication for openUTM partners 37

Figure 8: Outbound communication for CICS partners 37

Figure 9: Inbound communication for openUTM partners 38

Figure 10: Inbound communication for CICS partners 38

Figure 11: BeanConnect components in cluster operation 42

Figure 12: User interface of the Management Console 152

Figure 13: Navigation area 153

Figure 14: Starting the configuration wizard 159

Figure 15: Configuration wizard window 159

Figure 16: Proxies within the Management Console 170

Figure 17: Adding a new proxy 171

Figure 18: General information on a proxy 174

Figure 19: Description of the proxy components for CICS partners 177

Figure 20: Configuring the openUTM-LU62 Gateway 178

Figure 21: Configuring a communication service. 180
Figure 22: Configuring the properties of a communication service 181
Figure 23: Generating a new proxy cluster 186
Figure 24: Adding a proxy to a proxy cluster 188
Figure 25: Adding a resource adapter without cluster operation 190
Figure 26: Configuring the properties of an MC-CmdHandler 192
Figure 27: Adding a resource adapter in cluster operation 193
Figure 28: Editing the configuration file ra.xml. 197
Figure 29: Displaying and configuring EIS partners 198
Figure 30: General properties of an EIS partner of Type openUTM 200
Figure 31: Properties of the openUTM partner 202
Figure 32: Properties of the EIS partner for the availability check 206
Figure 33: General properties of an EIS partner of type CICS 209
Figure 34: Communication service properties of a CICS partner (single proxy) 211
Figure 35: Communication service properties of a CICS partner in a proxy cluster . . . 211
Figure 36: Properties of the CICS partner on a z/OS mainframe 213
Figure 37: Properties of the CICS type EIS partner for the availability check 215
Figure 38: Configuring outbound services. 220
Figure 39: Configuring outbound communication endpoints 222
Figure 40: Configuring inbound message endpoints 225
Figure 41: Configuring an inbound service 229
Figure 42: Configuring users for inbound communication 231
Figure 43: Configuring the inbound error prefix 232
Figure 44: Setting up a JMX client for resource adapters 243
Figure 45: Proxy context menu 256
Figure 46: Administering a BeanConnect proxy cluster. 260

Figure 47: Checking the availability of a BeanConnect proxy 272

Figure 48: Checking the availability of a resource adapter 273

Figure 49: Checking the availability of a resource adapter - result details. 274

Figure 50: Checking the availability of an openUTM-LU62 Gateway. 275

Figure 51: Availability of an MC-CmdHandler instance 276

Figure 52: Checking the availability of an EIS partner. 278

Figure 53: Navigation tree for an MBean. 279

Figure 54: MBean - object name subtree 281

Figure 55: MBean - attribute table 282

Figure 56: MBean - statistical values. 285

Figure 57: MBean operations 289

Figure 58: Command Line Interface (CLI) and the Management Console graphical user interface (GUI) 291

Figure 59: Configuring outbound communication with an openUTM application 373

Figure 60: Configuring inbound communication with an openUTM application. 375

Figure 61: Data exchange based on OltpMessagePart objects (openUTM partners). . 387

Figure 62: Data exchange based on OltpMessagePart objects (CICS partners) 388

Figure 63: Data exchange based on OltpMessageRecord objects (OSI-TP protocol with openUTM partners) 389

Figure 64: Data exchange based on OltpMessageRecord objects (CICS partners) . . . 389

Figure 65: Data exchange based on OltpMessageRecord objects (UPIC protocol) . . . 390

Figure 66: Configuring the loggers (example of BeanConnect proxy). 467

Figure 67: Configuring the appenders (example of BeanConnect proxy) 468

Figure 68: Logging output 472

Figure 69: Cobol2Java generation procedure 564