X/Open CAE Specification

Structured Transaction Definition Language (STDL)

X/Open Company Ltd.

© *August 1996, X/Open Company Limited*

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior permission of the copyright owners.

X/Open CAE Specification

Structured Transaction Definition Language (STDL)

ISBN: 1-85912-176-4

X/Open Document Number: C611

Published by X/Open Company Ltd., U.K.

Any comments relating to the material contained in this document may be submitted to:

The Open Group
Apex Plaza
Forbury Road
Reading
Berkshire, RG1 1AX
United Kingdom

or by Electronic Mail to:

XoSpecs@tog.org

Contents

Part 1 STDL Specification 1

Chapter 1 Introduction to STDL 3
1.1 The STDL Language 3
1.1.1 Major Features 3
1.1.2 Resource Manager Access 4
1.2 The STDL Model 5
1.2.1 Mapping STDL to Different Client/Server Models 5
1.2.2 Mapping STDL to the X/Open DTP Model 8
1.2.3 Mapping STDL to Existing TP Monitors 9
1.3 Client Programs 10
1.4 Application Flow and Transaction Control 11
1.4.1 Transactions 11
1.4.2 Task Call 11
1.4.3 Submit Task 12
1.4.4 Transactional Task Call 13
1.4.5 Non-transactional Task Call 13
1.4.6 Transactional Submit Task 14
1.4.7 Concurrent Execution 15
1.4.8 Text Messages 16
1.4.9 Exception Handling 16
1.5 Database and File Access 18
1.6 Accessing a Display 19
1.6.1 Non-transactional Presentation Procedures 19
1.6.2 Transactional Presentation Procedures 20
1.7 Communications 21
1.8 STDL Language Components 22
1.8.1 STDL Source Files 22
1.8.2 Execution Environment 26
1.8.3 The STDL Specification 26

Chapter 2 Introduction to Part 1 27
2.1 Scope 27
2.2 Organisation of Part 1 28
2.3 Syntax Notation 29
2.4 Conventions 30
2.4.1 Definitions 31
2.5 Conformance 32
2.5.1 Categories 34
2.5.2 Levels 35

Chapter 3 Concepts **37**
3.1 Transaction 37
3.1.1 Transactional and Durable Resources 37
3.1.2 Isolation Level 38
3.2 TP System Model 40
3.2.1 Task 41
3.2.2 Task Group 42
3.2.3 Processing Procedure 43
3.2.4 Processing Group 43
3.2.5 Display 44
3.2.6 Presentation Procedure 45
3.2.7 Presentation Group 45
3.3 Task Invocation 47
3.3.1 Task Call 47
3.3.2 Task Submit 50
3.3.2.1 Actions on the Submitter TP System 52
3.3.2.2 Actions on the Client TP System 53
3.3.3 Dialogues 54
3.4 Security 56
3.4.1 Principal 56
3.4.2 User Authentication 57
3.4.3 Remote Authentication 57
3.4.4 Authorization 58
3.5 Environment 59
3.5.1 Translation Environment 59
3.5.2 Execution Environment 59
3.5.3 Environmental Information 59
3.6 Record Queuing 60
3.6.1 Ordering 60
3.6.2 Records and Positions 60
3.6.3 Context 60
3.6.4 Queue Operations 61
3.6.4.1 Enqueue Operation 61
3.6.4.2 Dequeue Operation 61
3.6.4.3 Read Queue Operation 61
3.7 Exceptions 62
3.7.1 Types of Exceptions 62
3.7.2 Exception Classes, Sources, Codes and Level 62
3.7.3 How Exceptions Are Generated 64
3.7.4 Exception Handlers 64
3.7.5 TP System Actions when an Exception is Generated 65
3.7.5.1 Exception Handling in Customer-written and External Clients 65
3.7.5.2 Exception Handling in Processing Procedures 66
3.7.5.3 Exception Handling in Presentation Procedures 66
3.7.5.4 Non-transaction Exception Handling in Tasks 66
3.7.5.5 Transient Transaction Exception Handling in Tasks 67
3.7.5.6 Permanent Transaction Exception Handling in Tasks 67
3.7.5.7 Fatal Transaction Exception Handling in Tasks 68

	3.8	Resources	69
	3.8.1	Private Workspace	69
	3.8.2	Shared Workspace	69
	3.8.3	SQL Database	69
	3.8.4	Indexed File	70
	3.8.5	Relative File	70
	3.8.6	Sequential File	71
	3.8.7	Stream File	71
	3.8.8	Transactional Send and Receive	72
	3.8.9	Task Queue	72
	3.8.10	Record Queue	72
	3.8.11	Audit Log	72
	3.8.12	Transactional and Durable Attributes	73
	3.9	Message Group	74
Chapter	**4**	**Common Elements**	**75**
	4.1	Character Sets	75
	4.1.1	Source Character Sets	75
	4.1.2	Execution Character Sets	76
	4.2	Tokens	78
	4.3	Reserved Words	79
	4.4	Predefined Words	80
	4.4.1	<character-set-identifier>	80
	4.4.2	<class-identifier>	80
	4.4.3	System Workspace Identifiers	80
	4.5	OS Names	81
	4.5.1	Broadcast List	81
	4.5.2	Computer Language	81
	4.5.3	Destination	81
	4.5.4	Display	82
	4.5.5	Distribution List	82
	4.5.6	Human Language	82
	4.5.7	Record Queue	83
	4.6	Customer-defined Identifiers	84
	4.6.1	<external-identifier>	84
	4.6.1.1	<data-type-identifier>	85
	4.6.1.2	<message-group-identifier>	85
	4.6.1.3	<message-identifier>	85
	4.6.1.4	<presentation-group-identifier>	85
	4.6.1.5	<presentation-procedure-identifier>	85
	4.6.1.6	<processing-group-identifier>	86
	4.6.1.7	<processing-procedure-identifier>	86
	4.6.1.8	<task-group-identifier>	86
	4.6.1.9	<task-identifier>	86
	4.6.2	<internal-identifier>	86
	4.6.2.1	<field-identifier>	87
	4.6.2.2	<label-identifier>	87
	4.6.2.3	<workspace-identifier>	87

4.6.3 Syntactical References to Identifiers 88
4.7 Workspace Fields 89
4.7.1 <workspace-field> 89
4.7.2 Syntax References 91
4.8 Literals 92
4.8.1 <integer-literal> 92
4.8.2 <decimal-literal> 92
4.8.3 <string-literal> 92
4.8.4 <string-literal> Syntax References 93
4.9 <operator> 94
4.9.1 Relational Operators 94
4.9.2 Arithmetic Operators 94
4.9.3 Assignment Operator 94
4.10 <punctuator> 95
4.11 Message Parameter 95
4.11.1 <message-parameter> 95
4.12 Comment Text and Comment Character 95
4.13 Source File and Source Line 97
4.14 Preprocessing 99
4.14.1 <pre-define> 102
4.14.2 <pre-if> 103
4.14.3 <pre-include> 105
4.14.4 <pre-undef> 106
4.15 <boolean-expression> 107
4.15.1 <boolean-and-expression> 108
4.15.2 <boolean-term> 109
4.15.3 <boolean-comparison> 109
4.15.4 <boolean-negated-condition> 112
4.16 <integer-expression> 113
4.16.1 <multiplicative-expression> 114
4.16.2 <unary-expression> 115
4.16.3 <primary-expression> 116
4.17 Values 117
4.17.1 <absolute-time-value> 117
4.17.2 <delta-time-value> 118
4.17.3 <os-name-value> 119

Chapter 5 Data Type Definitions 121
5.1 <array-type> 121
5.2 <data-type> 123
5.3 <data-type-definition> 124
5.4 <decimal-string-type> 125
5.5 <integer-type> 127
5.6 <octet-type> 128
5.7 <record-type> 129
5.8 <text-type> 130
5.9 <uuid-type> 131

Chapter 6 Message Definition Language **133**
6.1 <message-group-definition> 133
6.2 <message-group-attribute-list> 134
6.2.1 <message-group-language> 135
6.2.2 <message-group-uuid> 136
6.3 <message-list> 137
6.3.1 <message-definition> 138
6.3.2 <message-text> 139

Chapter 7 Presentation Group Specification **141**
7.1 <presentation-group-specification> 141
7.2 <presentation-group-attribute-list> 142
7.2.1 <presentation-initialization-procedure> 143
7.2.2 <presentation-source-language> 144
7.2.3 <presentation-termination-procedure> 145
7.3 <presentation-procedure-list> 146
7.3.1 <presentation-procedure-interface> 147

Chapter 8 Processing Group Specification **151**
8.1 <processing-group-specification> 151
8.1.1 <processing-source-language> 152
8.2 <processing-procedure-list> 153
8.2.1 <processing-procedure-interface> 154

Chapter 9 Task Group Specification **157**
9.1 <task-group-specification> 157
9.2 <task-group-attribute-list> 158
9.2.1 <task-group-uuid-attribute> 159
9.2.2 <task-group-version-attribute> 160
9.3 <task-interface-list> 162
9.3.1 <composable-task-interface> 163
9.3.2 <noncomposable-task-interface> 164
9.3.3 <task-interface-argument-list-definition> 165

Chapter 10 Task Definition Language **167**
10.1 <task-definition> 168
10.1.1 <composable-task> 169
10.1.2 <noncomposable-task> 171
10.1.3 <task-argument-list-definition> 172
10.2 <task-attribute-list> 174
10.2.1 <restartability> 175
10.2.2 <send-display> 176
10.2.3 <workspace-list-definition> 177
10.3 <statement-list> 182
10.3.1 <assignment> 184
10.3.2 <audit> 186
10.3.3 <call-presentation-procedure> 188
10.3.4 <call-procedure> 193

	10.3.5	<call-task>	196
	10.3.6	<cancel-submit>	201
	10.3.7	<concurrent-block>	203
	10.3.8	<control-field>	205
	10.3.9	<dequeue-record>	207
	10.3.10	<enqueue-record>	212
	10.3.11	<exception-handler>	214
	10.3.12	<exception-information>	215
	10.3.13	<exit-block>	216
	10.3.14	<exit-task>	217
	10.3.15	<get-message>	218
	10.3.16	<go-to>	222
	10.3.17	<if>	223
	10.3.18	<raise-exception>	225
	10.3.19	<read-queue-record>	226
	10.3.20	<reraise-exception>	231
	10.3.21	<restart-transaction>	232
	10.3.22	<select-first>	233
	10.3.23	<statement-block>	235
	10.3.24	<submit-task>	237
	10.3.24.1	<submit-repeat>	241
	10.3.24.2	<submit-trigger>	242
	10.3.25	<transaction-block>	244
	10.3.26	<transaction-control>	246
	10.3.27	<while>	247
Chapter	**11**	**Processing Procedures**	**249**
	11.1	Processing Procedure	249
	11.2	Top-level Processing Procedure	250
	11.3	Top-level Processing Procedure Arguments	250
	11.4	Nested Processing Procedure	250
	11.5	Nested Processing Procedure Arguments	251
	11.6	Context	252
	11.7	File Access	254
	11.8	SQL Access	255
	11.9	File Transfer	256
	11.10	Error Handling Rules	256
Chapter	**12**	**Presentation Procedures**	**257**
	12.1	Send Presentation Procedure	257
	12.1.1	General Rules	257
	12.1.2	Transactional Send Procedure	257
	12.1.3	Broadcast Send Procedure	257
	12.2	Receive Presentation Procedure	259
	12.2.1	General Rules	259
	12.3	Transceive Presentation Procedure	259
	12.3.1	General Rules	259
	12.4	Presentation Context	260

	12.5	Customer-written Presentation Procedures	261
	12.5.1	Context	261
	12.6	Vendor-supplied Forms	262
	12.7	Initialization and Termination Procedures	262
Chapter	**13**	**Client Programs**	**263**
	13.1	Customer-written Client Programs	263
	13.1.1	Exceptions	263
	13.2	Menu Client Programs	264
	13.3	Processing Procedures	265
	13.4	External Clients	266
	13.4.1	Exceptions	266
	13.5	Stub Procedures	267
Chapter	**14**	**Use of COBOL**	**269**
	14.1	Data Type Mapping Information	269
	14.2	Exception Information	270
	14.2.1	Raising Exceptions	270
	14.2.2	Examining Exceptions Returned	271
	14.2.3	Exception Information COPY Files	271
	14.3	Processing Procedures	272
	14.3.1	File Access	272
	14.3.2	Restrictions	272
	14.4	Top-level Processing Procedure	274
	14.4.1	Arguments	274
	14.5	Presentation Procedures	275
	14.5.1	Arguments	275
	14.5.2	Restrictions	275
	14.6	STDL Identifier to COBOL Word Conversion	276
	14.7	Generated COBOL COPY Files	277
Chapter	**15**	**Use of C**	**279**
	15.1	Data Type Mapping Information	279
	15.2	Exception Information	280
	15.2.1	Raising Exceptions	280
	15.2.2	Examining Exceptions Returned	281
	15.2.3	Exception Information Header Files	281
	15.3	Processing Procedures	282
	15.3.1	File Access	282
	15.3.2	Restrictions	282
	15.4	Top-level Processing Procedures	284
	15.4.1	Arguments	284
	15.5	Presentation Procedures	285
	15.5.1	Arguments	285
	15.5.2	Restrictions	285
	15.6	STDL to C Identifier Conversion	286
	15.7	Generated C Header Files	287

Part	**2**	**STDL Environment, Execution and Protocol Mapping Specification**	**289**
Chapter	**16**	**Introduction to Part 2**	**291**
	16.1	Scope	291
	16.2	Organisation of Part 2	291
	16.3	Syntax Notation	291
	16.3.1	Definitions	291
	16.4	Conformance	292
Chapter	**17**	**Environmental Information**	**293**
	17.1	Translation Information	293
	17.1.1	Translating Source Files	293
	17.1.2	Procedures	294
	17.1.2.1	File Descriptor	295
	17.1.2.2	Data Type Definition	296
	17.1.2.3	SQL Access Information	296
	17.1.3	Transferring a Task Group Specification	297
	17.1.4	Moving an Application	297
	17.2	Execution Information	298
	17.2.1	TP Entity	298
	17.2.2	TP System Entity Containment	299
	17.2.3	TP System	301
	17.2.4	Audit Log	303
	17.2.5	Broadcast List	303
	17.2.6	Record Queue	304
	17.2.7	Display	305
	17.2.8	Distribution List	306
	17.2.9	Destination	306
	17.2.10	File	308
	17.2.11	Incoming Authentication	309
	17.2.12	Presentation Group	309
	17.2.13	Presentation Procedure	310
	17.2.14	Processing Group	310
	17.2.15	Top-level Processing Procedure	310
	17.2.16	SQL Environment	311
	17.2.17	Task Group	312
	17.2.17.1	Task	312
	17.2.18	Task Queue	315
	17.2.18.1	Submission Request	316
	17.2.19	User Profile	317
Chapter	**18**	**Application Execution**	**319**
	18.1	Introduction	319
	18.2	Scope	320
	18.3	Modelling Method	321
	18.3.1	State Machines and State Machine Instances	321
	18.3.2	Requests and Indications	322

18.3.3 State Machine Instance Context 322
18.3.4 Thread Contexts 323
18.3.5 Execution Contexts 323
18.4 Conventions 325
18.4.1 Mapping of Terms and Concepts 325
18.4.2 State Machine Description Conventions 325
18.5 Model 328
18.5.1 State Machine Categories 328
18.5.1.1 Client Program Call 329
18.5.1.2 Task Execution 329
18.5.1.3 Submitted Task Operations 330
18.5.1.4 Communication 330
18.5.1.5 Resource Manager 331
18.5.1.6 Procedure Execution 331
18.5.2 Relationships Among State Machines 331
18.6 Client Program Call State Machine 333
18.7 Task Execution State Machines 338
18.7.1 Task State Machine 338
18.7.2 Non-transactional Statement List State Machine 349
18.7.3 Transactional Statement List State Machine 369
18.7.4 Concurrent Block State Machine 396
18.7.5 Concurrent Thread State Machine 404
18.8 Submitted Task Operations State Machines 407
18.8.1 Task Dequeuer State Machine 407
18.8.2 Task Forwarder State Machine 412
18.8.3 Task Enqueuer State Machine 416
18.9 Communication State Machines 423
18.9.1 RPC Interfaces Used 423
18.9.1.1 Interfaces for Calling Tasks 424
18.9.1.2 Interface for the Task Enqueuer 426
18.9.1.3 Exceptions 430
18.9.2 Call State Machine 433
18.9.3 Transaction State Machine 447
18.10 Resource Manager State Machines 454
18.10.1 Transactional Receive State Machine 454
18.10.2 Resource Manager State Machine 458
18.11 Procedure Execution State Machines 468
18.11.1 Processing Procedure State Machine 468
18.11.2 Presentation Procedure State Machine 474

Part 3 Appendices **479**

Appendix A Architectural Constants **481**
- A.1 Architectural Limits 481
- A.2 Exception Class Values 486

Appendix B Audit Data Type Definition **487**

Appendix C Exception Classes **489**

Appendix D Complete BNF **501**
- D.1 Data Type Definition 501
- D.2 Message Definition Language 503
- D.3 Presentation Group Specification 504
- D.4 Processing Group Specification 506
- D.5 Task Group Specification 507
- D.6 Task Definition Language 509

Appendix E Data Type Mapping **523**
- E.1 Introduction 523
- E.2 Definition of Data Type Mapping 524

Appendix F Human Language Mapping **527**
- F.1 Introduction 527
- F.2 Language Names and Execution Character Sets 528

Appendix G Encoding Rules of STDL Data Types **529**
- G.1 Introduction 529
- G.2 Mapping of APDU Data Types to ASN.1 530
- G.3 APDU Data Type Encoding Rules 532
  - G.3.1 Katakana Character Set Encoding Rules 532
  - G.3.2 Kanji Character Set Encoding Rules 532
  - G.3.3 UUID Encoding Rules 532

Appendix H Use of DCE RPC Protocol **533**
- H.1 Introduction 533
- H.2 Scope 534
  - H.2.1 Introduction 534
  - H.2.2 Subjects within the Scope of the Appendix 534
  - H.2.3 Subjects Not within the Scope of the Appendix 534
- H.3 RTI Model Subset 535
  - H.3.1 Introduction 535
  - H.3.2 RTI Model Subset Components 535
    - H.3.2.1 RTI Service User Invocation 535
    - H.3.2.2 RTI Protocol Machine 535
  - H.3.3 RTI Model Subset Component Relationships 536
- H.4 RTI Service Primitive Subset 537
  - H.4.1 Introduction 537

H.5 RTI Service Primitive Mapping 538
H.5.1 Introduction 538
H.5.2 Kernel Functional Unit 538
H.5.2.1 RTI-ESTABLISH-CONTEXT.req 538
H.5.2.2 RTI-CALL-TASK.req 539
H.5.2.3 RTI-CALL-TASK.ind 539
H.5.2.4 RTI-CANCEL-CALL.req 540
H.5.2.5 RTI-CANCEL-CALL.ind 540
H.5.2.6 RTI-CALL-FAILURE.ind 540
H.5.2.7 RTI-CALL-RESULT.req 540
H.5.2.8 RTI-CALL-RESULT.ind 541
H.5.3 Non-transactional Functional Unit 541
H.5.3.1 RTI-RELEASE-CONTEXT.req 541
H.5.3.2 RTI-RELEASE-CONTEXT.ind 541
H.6 Optional Use of DCE Name Service 542
H.6.1 Introduction 542
H.6.2 Name Service Primitives 542
H.7 Mapping of APDU Data Types to NDR 544

Index **545**

List of Figures

1-1 STDL Resource Managers 4
1-2 STDL Model 5
1-3 Mapping STDL Groups to a Client and Server 6
1-4 Mapping STDL Groups for Server-Server Communications 7
1-5 External Client 7
1-6 Mapping the STDL Model to the X/Open DTP Model 8
1-7 Mapping STDL to Existing TP Monitors 9
1-8 On-line Task Calls 11
1-9 Submitted Task Invocation 12
1-10 Composable Task Call 13
1-11 Non-composable Task Call 14
1-12 Submitting a Composable Task with Dependent Work 15
1-13 Concurrent Block Execution 16
1-14 Exception Processing 17
1-15 Processing Procedure Invocation 18
1-16 Non-transactional Receive and Send 19
1-17 Transactional Presentation Procedures 20
1-18 STDL Language Model 22
1-19 STDL Source Files 25
2-1 Conformance Categories and their Functional Capabilities 34
3-1 TP System Model 40
3-2 Task Submission, Forwarding and Calling 51
3-3 Record Queue 60
14-1 EINFO Data Type Definition 271
15-1 EINFO Data Type Definition 281

17-1 File and Database Descriptors for a Procedure 294
17-2 Entities Managed Within a TP System 300
18-1 Requests and Indications 322
18-2 Overall Relationships Between State Machines 329
18-3 Relationships Among All State Machines 332
H-1 RTI Model Subset 535
H-2 Relationship between RTI and DCE RPC Services 537

List of Tables

1-1 Major STDL Statements 24
2-1 Level List by Difficulty of Implementation 35
3-1 TP System Resource Types 37
3-2 Transaction Isolation Level 39
3-3 Dialogue Termination 54
3-4 Transactional and Durable Attributes of Resources 73
4-1 Reserved Words 79
18-1 Client Program 337
18-2 Task (Null State) 347
18-3 Task (Composable, Remote Client, Statements and Cancel States) 347
18-4 Task (Composable, Remote Client, Inactive States) 347
18-5 Task (Composable, Local Client States) 348
18-6 Task (Non-composable Task State) 348
18-7 Non-transactional Statement List (Null State) 360
18-8 Non-transactional Statement List (Block State) 360
18-9 Non-transactional Statement List (Handler State) 361
18-10 Non-transactional Statement List (Conditional States) 362
18-11 Non-transactional Statement List (Loop States) 363
18-12 Non-transactional Statement List (Work State) 364
18-13 Non-transactional Statement List (Cancel State) 364
18-14 Non-transactional Statement List (Transaction Block State) 365
18-15 Non-transaction Statement List (Transaction Block Standard Commit States) 366
18-16 Non-transaction Statement List (Transaction Block Task Exit Commit States) 366
18-17 Non-transaction Statement List (Transaction Block Go To Commit State) 366
18-18 Non-transaction Statement List (Transaction Block Standard Rollback States) 367
18-19 Non-transaction Statement List (Transaction Block Task Exit Rollback States) 367
18-20 Non-transaction Statement List (Transaction Block Go To Rollback States) 367
18-21 Non-transaction Statement List (Transaction Block Non-exception Cancel States) 367
18-22 Non-trans. Stmt. List (Trans. Block Rollback for Restart, Handler, Exception, Cancel States) 368
18-23 Non-transaction Statement List (Transaction Block

	Exception States)	368
18-24	Transactional Statement List (Null State)	386
18-25	Transactional Statement List (Block State)	387
18-26	Transactional Statement List (Concurrent Block State)	388
18-27	Transactional Statement List (Exception Handler State)	389
18-28	Transactional Statement List (Conditional Evaluation State)	390
18-29	Transactional Statement List (Conditional Execute State)	391
18-30	Transactional Statement List (Loop Boolean State)	392
18-31	Transactional Statement List (Loop Execute State)	393
18-32	Transactional Statement List (Internal Work State)	394
18-33	Transactional Statement List (External Work State)	395
18-34	Transactional Statement List (Cancel State)	395
18-35	Concurrent Block (Null State)	402
18-36	Concurrent Block (Execute, Exit and Non-transaction Exception States)	402
18-37	Concurrent Block (Transaction Exception and Cancel States)	403
18-38	Concurrent Thread	406
18-39	Task Dequeuer (Null, Dequeue and Execute States)	411
18-40	Task Dequeuer (Commit, Rollback and Shutdown States)	411
18-41	Task Forwarder (Null, Dequeue and Forward States)	415
18-42	Task Forwarder (Commit, Rollback and Shutdown States)	415
18-43	Task Enqueuer (Null State)	422
18-44	Task Enqueuer (Non-transactional Forwards)	422
18-45	Task Enqueuer (Transactional Forwards)	422
18-46	Call (Null State)	445
18-47	Call (Local Call States)	445
18-48	Call State Transitions (Client to Remote Server)	446
18-49	Call (Server from Remote Client)	446
18-50	Transaction (Null State)	452
18-51	Transaction (Active States)	452
18-52	Transaction (Local Termination States)	452
18-53	Transaction (Distributed P1 Commit States)	452
18-54	Transaction (Distributed P2 Commit States)	453
18-55	Transaction (Distributed Rollback States)	453
18-56	Transactional Receive	457
18-57	Resource Manager (Null and Operating States)	467
18-58	Resource Manager (Commit and Rollback States)	467
18-59	Processing Procedure	473
18-60	Presentation Procedure	478
A-1	TP System Resource Minimum Limits	482
A-2	STDL Minimum Limits	483
A-3	I/O Minimum Limits	484
A-4	Environment Execution Minimum Limits	485
C-1	Exception Types	499
H-1	RTI Service Primitives Required for Non-transactional RTI Service	537
H-2	Summary of RTI to DCE RPC Service Primitive Mapping	538
H-3	rpc_binding_from_string_binding service	538
H-4	rpc_binding_set_auth_info service	539

H-5 pthread_cancel service 540
H-6 rpc_binding_free service 541
H-7 Summary of RTI to DCE RPC Name Service Mapping 542
H-8 rpc_ns_binding_import_begin service 542
H-9 rpc_ns_binding_import_next service 542
H-10 rpc_ns_binding_import_done 542

Preface

The Open Group

In February 1996, X/Open and the Open Software Foundation (OSF) joined forces to become The Open Group, which represents one of the leading authorities in open systems. It is supported by most of the world's largest information systems suppliers, user organisations and software companies. By combining their complementary strengths — X/Open in providing specifications and its trade mark branding scheme, and OSF in facilitating collaboration among customers and system/software vendors toward the development of new open systems technologies — The Open Group is well positioned to assist vendors and users to develop and implement products which support adoption and proliferation of open systems.

Established in 1984, X/Open is an organisation dedicated to the identification, agreement and widescale adoption of Information Technology standards which provide compatibility (portability and interoperability) between software products, and so help users realise the business benefits of open information systems — lower costs, increased choice and greater flexibility. It achieves this by combining existing and emerging standards into an evolving set of integrated, high-value and usable open system specifications, which form a Common Applications Environment (CAE). It licenses a trade mark — called the X/Open Brand — for use on products which vendors have demonstrated are conformant to this CAE. The X/Open brand is recognised by users worldwide as the guarantee of compliance to open systems standards. X/Open is also responsible for the management of the UNIX trade mark on behalf of the industry.

Founded in 1988, the Open Software Foundation (OSF) delivers technology innovations in all areas of open systems, including interoperability, scalability, portability and usability. OSF is a worldwide coalition of vendors and customers in industry, government and academia, who work together to provide advanced open-systems technology solutions for use in a distributed computing environment. It runs programmes in collaborative research and development, to deliver vendor-neutral technology (software source code and supporting documentation) in key IT areas. These include OSF/1, Distributed Computing Environment (DCE), OSF/Motif and Common Desktop Environment (CDE).

X/Open Technical Publications

X/Open publishes a wide range of technical literature, the main part of which is focused on specification development, but which also includes Guides, Snapshots, Technical Studies, Branding/Testing documents, industry surveys and business titles.

There are two types of X/Open specification:

- *CAE Specifications*

  CAE (Common Applications Environment) specifications are the stable specifications that form the basis for X/Open-branded products. These specifications are intended to be used widely within the industry for product development and procurement purposes.

  Anyone developing products that implement an X/Open CAE specification can enjoy the benefits of a single, widely supported industry standard. In addition, they can demonstrate product compliance through the X/Open brand trade mark. CAE specifications are published as soon as they are developed, so enabling vendors to proceed with development of conformant products without delay.

- *Preliminary Specifications*

  These specifications usually address an emerging area of technology and consequently are not yet supported by multiple sources of stable conformant implementations. They are published for the purpose of validation through implementation of products. A Preliminary specification is not a draft specification; rather, it is as stable as can be achieved, through applying X/Open's rigorous development and review procedures.

  Preliminary specifications are analogous to the *trial-use* standards issued by formal standards organisations, and developers are encouraged to develop products on the basis of them. However, experience through implementation work may result in significant (possibly upwardly incompatible) changes before its progression to becoming a CAE specification. While the intent is to progress Preliminary specifications to corresponding CAE specifications, the ability to do so depends on consensus among X/Open members.

In addition, X/Open publishes:

- *Guides*

  These provide information that X/Open believes is useful in the evaluation, procurement, development or management of open systems, particularly those that are X/Open-compliant. X/Open Guides are advisory, not normative, and should not be referenced for purposes of specifying or claiming X/Open conformance.

- *Technical Studies*

  X/Open Technical Studies present results of analyses performed on subjects of interest in areas relevant to X/Open's Technical Programme. They are intended to communicate the findings to the outside world so as to stimulate discussion and activity in other bodies and the industry in general.

- *Snapshots*

  These provide a mechanism for X/Open to disseminate information on its current direction and thinking, in advance of possible development of a Specification, Guide or Technical Study. The intention is to stimulate industry debate and prototyping, and solicit feedback. A Snapshot represents the interim results of a technical activity. Although at the time of its publication, there may be an intention to progress the activity towards development of a Specification, Guide or Technical Study, the ability to do so depends on consensus among X/Open members.

Versions and Issues of Specifications

As with all *live* documents, CAE Specifications require revision to align with new developments and associated international standards. To distinguish between revised specifications which are fully backwards compatible and those which are not:

- A new *Version* indicates there is no change to the definitive information contained in the previous publication of that title, but additions/extensions are included. As such, it *replaces* the previous publication.
- A new *Issue* indicates there is substantive change to the definitive information contained in the previous publication of that title, and there may also be additions/extensions. As such, both previous and new documents are maintained as current publications.

Corrigenda

Readers should note that Corrigenda may apply to any publication. Corrigenda information is published on the World-Wide Web at **http://www.xopen.org** under Sales and Ordering.

Ordering Information

Full catalogue and ordering information on all Open Group publications is available on the World-Wide Web at **http://www.xopen.org** under Sales and Ordering.

This Document

This document contains:

- an informative introduction to STDL
- a description of the syntax and semantics of STDL
- a description of the environment in which STDL programs execute
- state tables that model STDL execution
- mapping of STDL to the TxRPC and DCE RPC protocols.

STDL is derived from the Digital Equipment Corporation task definition language.

This specification was submitted to the X/Open fast-track process by SPIRIT (Service Providers' Integrated Requirements for Information Technology), under the auspices of the Network Management Forum (NMF). It has now been formally adopted by X/Open as a CAE Specification. It supersedes the following documents published by X/Open:

STDL Preliminary Specification
X/Open Preliminary Specification, December 1995, Structured Transaction Definition Language (STDL) (ISBN: 1-85912-120-9, P536.

SPIRIT Issue 2.0, Volume 3
X/Open NMF SPIRIT Documentation, December 1994, SPIRIT STDL Language Specification (SPIRIT Issue 2.0, Volume 3) (ISBN: 1-85912-063-6, J403).

SPIRIT Issue 2.0, Volume 4
X/Open NMF SPIRIT Documentation, December 1994, SPIRIT STDL Environment, Execution and Protocol Mapping (SPIRIT Issue 2.0, Volume 4) (ISBN: 1-85912-064-4, J404).

Structure

The Structured Transaction Definition Language (STDL) Specification contains:

- Part 1, STDL Specification

  This part describes the syntax and semantics of Structured Transaction Definition Language (STDL). STDL is a high-level language for developing portable transaction processing applications in a multi-vendor environment. This part is a detailed introduction to STDL.

- Part 2, STDL Environment, Execution and Protocol Mapping Specification

  This part details the execution environment for STDL applications, models STDL execution, and maps STDL to the X/Open TxRPC (Document Number C505) and DCE RPC (Document Number C309) protocols.

Intended Audience

Part 1, STDL Specification is intended for users and implementors of STDL. It should be read by those interested in using STDL or in evaluating its potential use.

Part 2, STDL Environment, Execution and Protocol Mapping Specification is intended for implementors of STDL.

Typographical Conventions

The following typographical conventions are used throughout this document:

- *Italic* strings are used for emphasis or to identify the first instance of a word requiring definition.
- Syntax is shown in `fixed width` font. Brackets shown in this font, `[ ]`, are part of the syntax and do *not* indicate optional items. In syntax the `|` symbol is used to separate alternatives, and ellipses (`. . .`) are used to show that additional arguments are optional.
- Variables within syntax statements are shown in *`italic fixed width font`*.
- Syntax notation conventions used in this document are detailed in Section 2.3 on page 29.

Trade Marks

X/Open® is a registered trade mark, and the ‘‘X’’ device is a trade mark, of X/Open Company Limited.

Acknowledgements

X/Open gratefully acknowledges the work of and significant contribution made by the NMF SPIRIT team in the development of this specification.

Referenced Documents

The following documents are referenced in this specification.

These references are considered normative; that is, they are external documents that are requisite for a complete understanding of this document.

Some of the references listed are standards specifications. These standards contain provisions that, though referenced as external documents, also constitute provisions of this specification. All relevant versions of a standard are referenced and, at the time of publication, considered to be valid.

X/Open Documents

DCE RPC
: X/Open CAE Specification, August 1994, X/Open DCE: Remote Procedure Call (ISBN: 1-85912-041-5, C309).

DTP
: X/Open Guide, November 1993, Distributed Transaction Processing: Reference Model, Version 2 (ISBN: 1-85912-019-9, G307).

ISAM
: X/Open Specification, February 1992, Data Management, Issue 3 (ISBN: 1-872630-40-5, C215); this specification was formerly X/Open Portability Guide, Volume 5, August 1988 (ISBN; 1-13-685876-7, XO/XPG/89/006).

SQL, Version 2
: X/Open CAE Specification, March 1996, Data Management: Structured Query Language (SQL), Version 2 (ISBN: 1-85912-151-9, C449).

TX
: X/Open Preliminary Specification, October 1992, Distributed Transaction Processing: The TX (Transaction Demarcation) Specification (ISBN: 1-872630-65-0, P209).

TxRPC
: X/Open CAE Specification, October 1995, Distributed Transaction Processing: The TxRPC Specification (ISBN: 1-85912-115-2, C505).

XFTAM
: X/Open CAE Specification, January 1994, FTAM High-level API (XFTAM) (ISBN: 1-85912-010-5, C304).

Non-X/Open Documents

ISO 1989
: ISO 1989: 1985, Programming Languages — COBOL (technically identical to ANSI standard X3.23-1985).

ISO 1989/Amendment 1
: ISO 1989/Amendment 1:1992, Intrinsic Function Module (technically identical to ANSI standard X3.23a-1989).

ISO 3166
: ISO 3166: 1993, Codes for the Representation of Names of Countries, Bilingual edition (Edition 4).

ISO 6093
: ISO 6093: 1985, Information Processing — Representation of Numerical Values in Character Strings for Information Interchange.

ISO 639
: ISO 639: 1988, Codes for the Representation of Names of Languages, Bilingual edition (Edition 1).

ISO Alphanumeric
: ISO/IEC 646: 1991, Information Processing — ISO 7-bit Coded Character Set for Information Interchange (Edition 3).

ISO ASN.1 BER
: ISO/IEC 8825: 1990, Information Technology — Open Systems Interconnection — Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1).

ISO C
: ISO/IEC 9899: 1990, Programming Languages — C (technically identical to ANSI standard X3.159-1989).

ISO Latin 1
: ISO 8859-1: 1987, Information Processing — 8-bit Single-byte Coded Graphic Character Sets — Part 1: Latin Alphabet No. 1.

ISO Latin 2
: ISO 8859-2: 1987, Information Processing — 8-bit Single-byte Coded Graphic Character Sets — Part 2: Latin Alphabet No. 2.

ISO UCS-2
: ISO/IEC 10646-1: 1993, Information Technology — Universal Multiple-octet Coded Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane.

JIS Kanji
: JIS X0208: 1983, Code of the Japanese Graphic Character Set for Information Interchange (in Japanese).

JIS Katakana
: JIS X0201: 1976, Code for Information Interchange, Table 3, Katakana Graphic Character Set (in Japanese).

JIS XO212: 1990
: JIS X0212: 1990, Code for the Supplementary Japanese Graphic Character Set for Information Interchange (in Japanese).

JIS X0221: 1995
: JIS X0221: 1995, Universal Multiple-octet Coded Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane.

OSI DTP
: ISO/IEC 10026: 1992, Information Technology — Open Systems Interconnection — Distributed Transaction Processing.

Structured Transaction Definition Language (STDL)

Part 1:

STDL Specification

X/Open Company Ltd.

Chapter 1

Introduction to STDL

This chapter is informative.

The Structured Transaction Definition Language (STDL) is a high-level transaction processing language designed for creating portable and modular distributed TP applications.

1.1 The STDL Language

STDL is a TP language that achieves multi-platform portability and modularity by providing functionality not available in standard C, COBOL or SQL. The interfaces to these functions (such as transaction control and exception handling) are incorporated within the STDL syntax. STDL also defines the environment in which the TP applications execute.

As a unique TP language, STDL represents a practical method of providing portability and modularity for complete TP functionality, leveraging existing language standards to the greatest extent possible. STDL provides interoperability by mapping to standard communication protocols for remote procedure calls.

Programmers use STDL to create STDL applications. STDL applications execute procedures written using STDL and C or COBOL with embedded SQL. An STDL task is a procedure that demarcates transactions and performs a sequence of STDL operations within each transaction.

STDL tasks call standard C and COBOL procedures for complex processing, for all operations on databases and files, and for display and I/O device access. Isolating transactional features in STDL removes the necessity of adding transactional features to standard C and COBOL, thus increasing the portability of TP applications, and facilitating the mapping of STDL onto existing TP monitors.

1.1.1 Major Features

STDL provides the following features:

- transaction demarcation
- flexible exception handling
- defined interfaces to resource managers
- queued as well as on-line TP
- transactional workspaces accessible by the programmer
- transactional presentation messaging for reliable interaction with displays or other devices
- ability to call and be called by C and COBOL programs
- ability to spawn independent transactions from within a transaction
- ability to separate procedures within a transaction and execute them concurrently
- ability to enqueue and dequeue data
- data typing
- multi-lingual defining of user messages.

STDL includes supplementary environmental information required for full application portability (that is, features that the programmer can count on but are not contained in the language itself, such as a transaction timeout or presentation procedure timeout limit).

The STDL environment can be mapped to various existing vendors' TP environments, and the STDL modularity provides flexibility of configuration.

1.1.2 Resource Manager Access

Several resource managers (RMs) are defined for STDL. Most are transactional, some are non-transactional, and others are transactional at the programmer's option. A communication resource manager (CRM) is an RM whose function is to provide remote communications for applications.

Figure 1-1 illustrates the RMs for which STDL coordinates transactional operations, including optionally transactional RMs. The shaded area indicates transactional context. A client program is part of the transactional context when the client performs the call within a transaction.

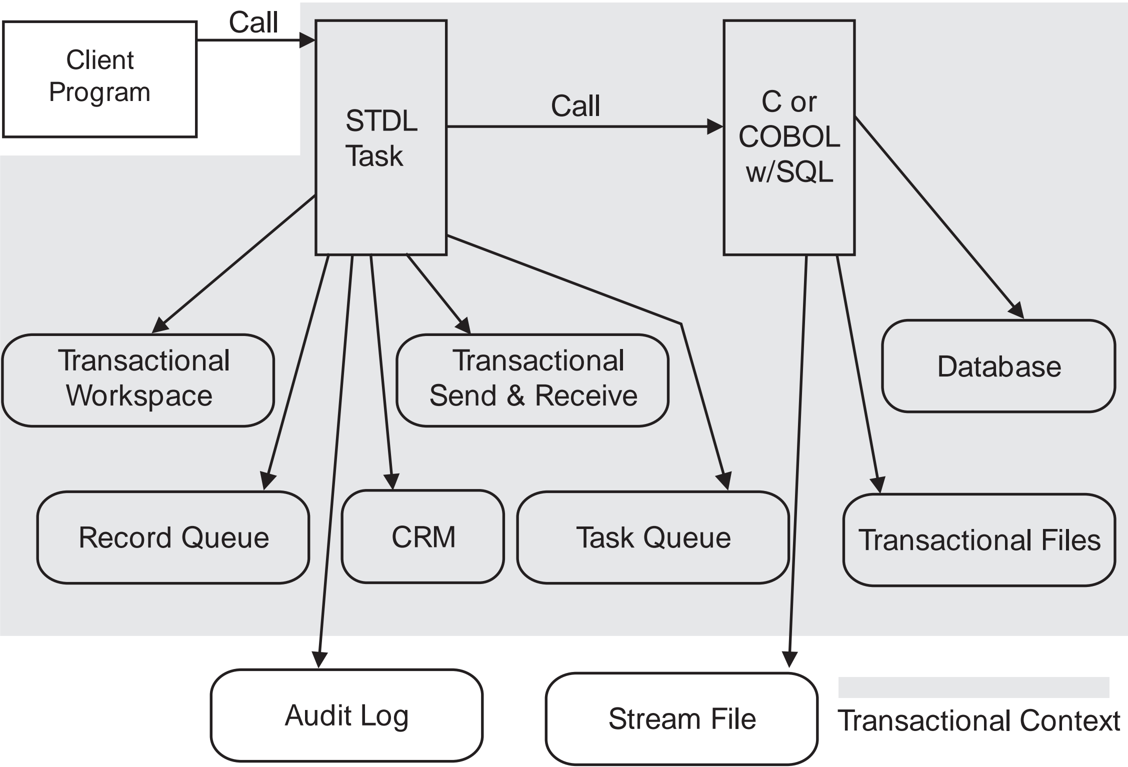


Figure 1-1 STDL Resource Managers

As shown in Figure 1-1, some of the RMs that STDL accesses do not support transactional operations. Others optionally support them (not shown).

1.2 The STDL Model

STDL implements a transaction processing model that divides procedures into three modules called groups. Each group is designed for a specific purpose: presentation (communication with the end-user), application flow and transaction control, and database and file access.

Communications between the groups of procedures is accomplished using a procedure call. A remote procedure call (RPC) is used between groups of tasks to enable distributed processing. Figure 1-2 illustrates the STDL model.

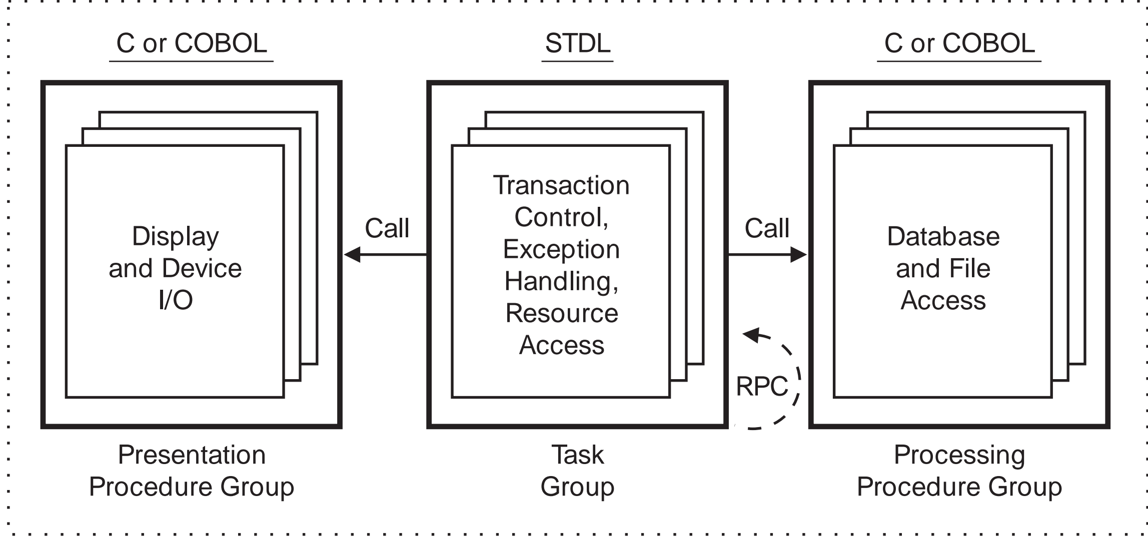


Figure 1-2 STDL Model

Each group of procedures can be created, maintained and replaced separately. The interface to each group of procedures is defined separately, so that procedures in one of the groups can be changed without changing procedures in the other groups, as long as the interfaces remain the same. Separating a procedure from its interface also is consistent with the principles of encapsulation, providing a path for future migration to object-oriented technology.

The mechanism used to perform the local procedure call between the different types of groups is defined according to each vendor's system environment. The STDL specification does not define it, although the procedure call syntax is defined.

1.2.1 Mapping STDL to Different Client/Server Models

Because of its modularity, the STDL model maps easily to various configurations of the client/server model.

Figure 1-3 on page 6 illustrates the case where the STDL presentation group and task group are placed on a computer acting as the application client. An STDL task group and a processing group are placed on another computer to act as the application server.

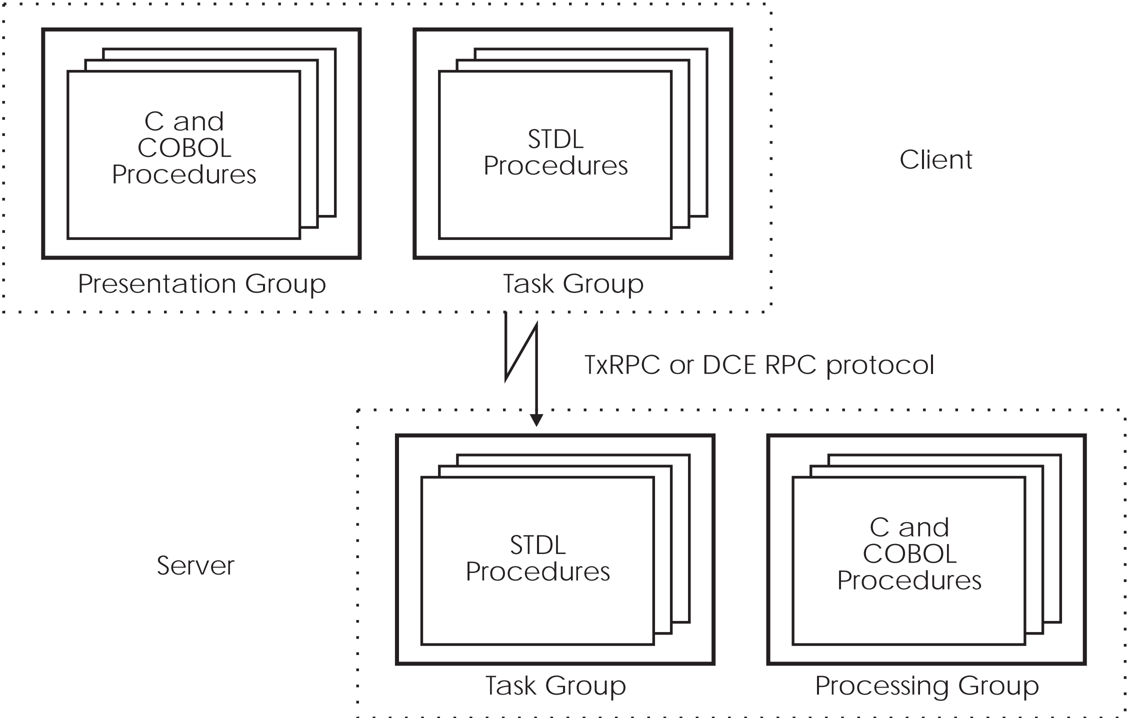


Figure 1-3 Mapping STDL Groups to a Client and Server

The point of communication between an STDL client and an STDL server is the task group, in which a portable remote STDL task call is defined.

As shown in Figure 1-4 on page 7, STDL groups can be placed in a combination to support server-to-server, or distributed TP application configurations.

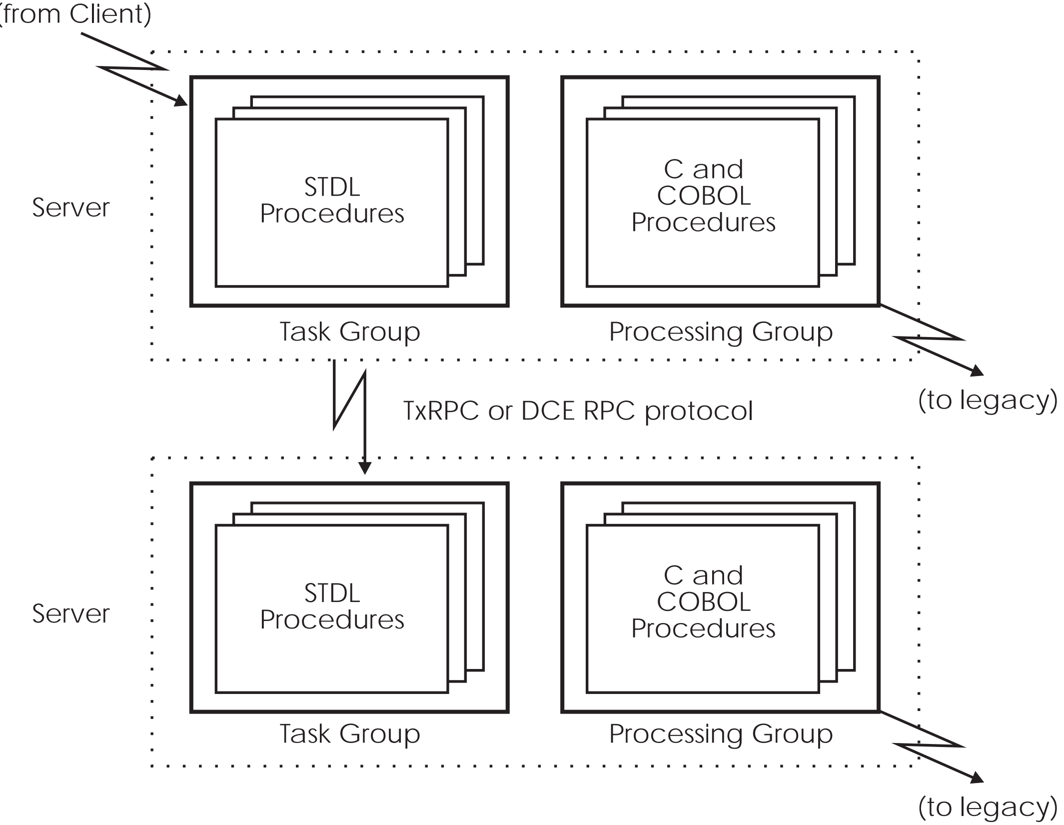


Figure 1-4 Mapping STDL Groups for Server-Server Communications

Use of a legacy protocol is possible in C or COBOL procedures for interoperability with non-STDL applications.

STDL also defines an external client; that is, a client program running externally to the STDL environment, as shown in Figure 1-5.

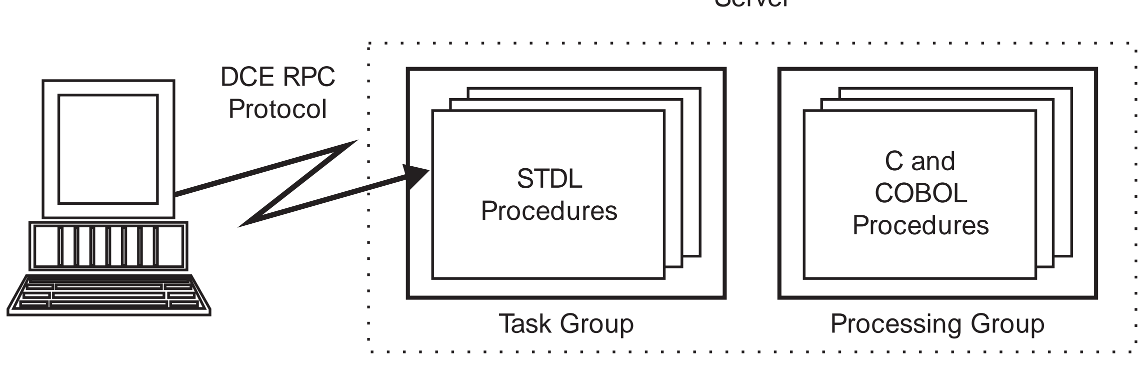


Figure 1-5 External Client

In the external client model, an external display device, such as a PC or workstation, replaces the function of the presentation group and runs a client program that performs STDL task calls using the DCE RPC protocol.

1.2.2 Mapping STDL to the X/Open DTP Model

There are many points of similarity and general compatibility between the STDL and X/Open DTP models. For example, both models are based on OSI TP.

Figure 1-6 illustrates the relationship of STDL to the X/Open DTP model. Basically, STDL represents an application in the X/Open model. STDL is a full-function TP language and includes features additional to those defined within the X/Open DTP model. Figure 1-6 illustrates how the STDL model relates to the X/Open model when STDL is mapped to an X/Open DTP-compliant TP system.

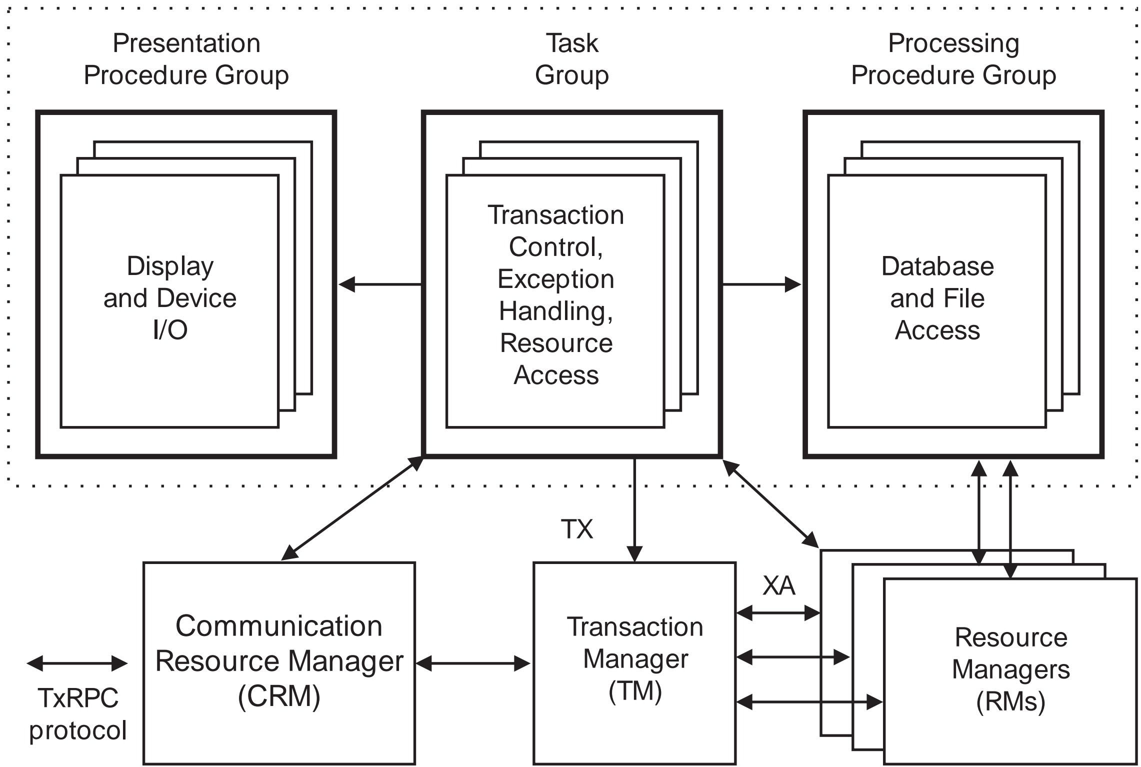


Figure 1-6 Mapping the STDL Model to the X/Open DTP Model

Figure 1-6 shows that the STDL model expands the Application Program (AP) portion of the X/Open DTP model. A C or COBOL processing procedure provides an interface to a resource manager (RM); for example, an SQL database manager. Access to the transaction manager (TM) is through STDL, and can be accomplished using the X/Open TX interface. Also, access to the communication RM (CRM) is accomplished directly via STDL, as is access to RMs such as the task queue, record queue and audit log.

The AP defines transaction boundaries through the TX interface, accesses resources from multiple RMs, and communicates with remote APs using the TxRPC or DCE RPC protocol. (STDL does not require using the TxRPC API, although using the TxRPC API is compatible with STDL because the underlying TxRPC protocol is the same.)

Although STDL fits well with the X/Open DTP model, it can also be mapped onto vendor-proprietary TP monitors.

1.2.3 Mapping STDL to Existing TP Monitors

Because STDL is a high-level language, it can be mapped onto existing TP monitors. Figure 1-7 illustrates that STDL can be adapted differently to different TP monitors to provide portability for the application.

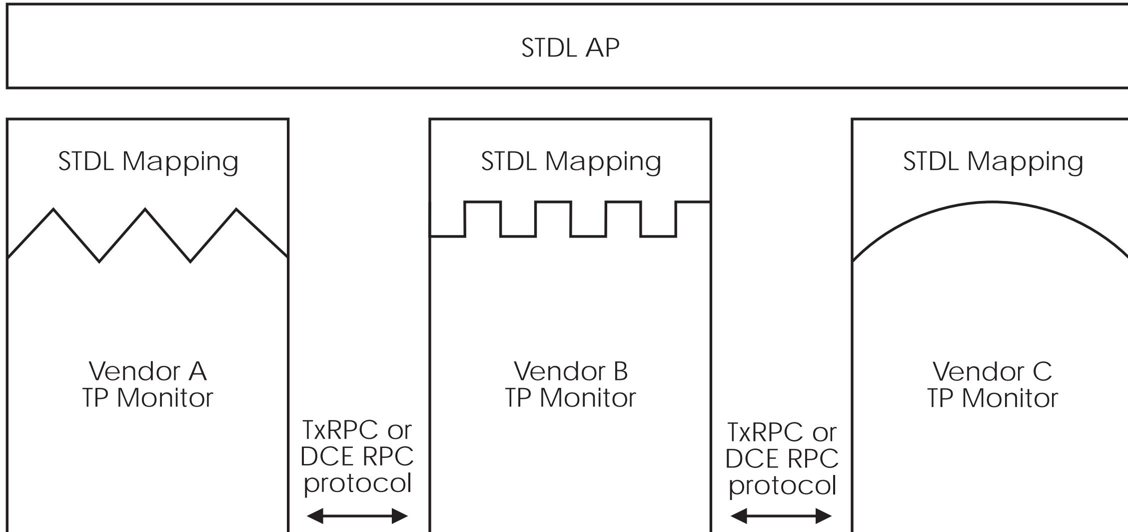


Figure 1-7 Mapping STDL to Existing TP Monitors

After STDL is mapped to an existing TP monitor, applications written using STDL together with standard C or COBOL are portable, and can be distributed according to the various groupings of modules STDL supports.

1.3 Client Programs

A client program is any program that includes an STDL task call. A client program is always the first procedure to call an STDL task. After the initial call, an STDL task itself can act as a client program by calling another task. While several different types of client programs are identified for STDL, a procedure call mechanism is always used to invoke an STDL task. Some of the procedure calls are local and others are remote. Client programs are divided into two basic types:

Internal Within the STDL environment.

External Outside the STDL environment.

The STDL environment defines a vendor-specific set of supporting services, restrictions and execution attributes which are required in order to execute STDL programs. A client program that is operating within this STDL environment uses a vendor-specific procedure call mechanism. A client program that operates outside of this STDL environment, or calls a remote task, uses the call mechanism and communication protocol defined for multi-vendor interoperability. A client program always uses the same basic mechanism (a procedure call) to invoke a task. An internal procedure call is vendor-specific. An external procedure call uses either the DCE RPC or TxRPC protocol.

1.4 Application Flow and Transaction Control

STDL provides application flow and transaction control by using block-structuring mechanisms that demarcate transactions, sequence procedure calls, group operations on resource managers, and handle exceptions. Once an STDL task is called to initiate a transaction, all application flow and transactional work is controlled by STDL until the end of the task.

1.4.1 Transactions

A transaction is a sequence of operations that is atomic with respect to recovery. An STDL transaction is defined as a series of transactional operations that either complete successfully and are committed, or do not complete successfully and are rolled back. STDL transactions can access a variety of resource managers. Resource managers that provide support for transactional operations can participate in an STDL transaction. STDL provides for local as well as remote transactional control.

When an STDL task is called, transaction control is started or continued, and all operations on transactional data participate in the transaction (that is, operations on data maintained by resource managers capable of supporting transactional operations). At the programmer's option, an STDL task can perform non-transactional work, such as a non-transactional RPC or a non-transactional write to a workspace, audit log or file.

1.4.2 Task Call

An STDL task can perform a direct call to a local task or a remote task. A direct call means the client waits for the server to complete. As shown Figure 1-8, successful execution of the server task returns data to the client. Unsuccessful execution returns an exception to the client (not shown).

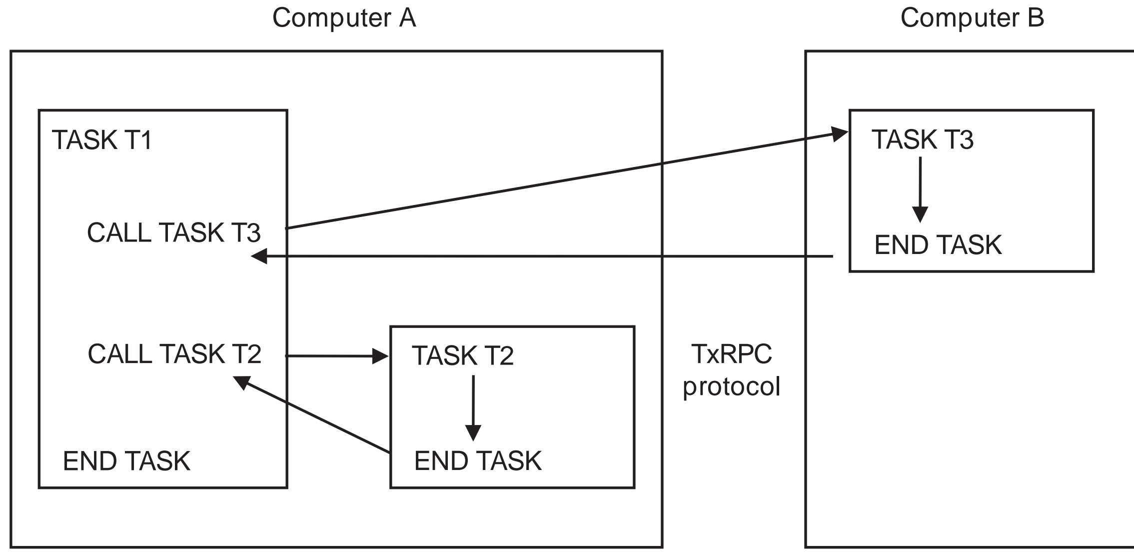


Figure 1-8 On-line Task Calls

Figure 1-8 also illustrates the difference between a local task call (client and server on the same computer) and a remote task call (client and server on different computers). When a client calls a server on another computer, the TxRPC or DCE RPC protocol is used. When a client calls a server on the same computer, a vendor-specific procedure call mechanism is used. A local task

call can be changed to a remote task call without reprogramming by specifying a remote server task instead of a local server task (using environmental information).

A transactional task call is identified as having dependent work; that is, the success of the calling task's transaction is dependent upon the successful completion of the server task. The server task is therefore identified as a COMPOSABLE TASK — one whose work is composed within the client task's transaction.

1.4.3 Submit Task

An STDL task can perform an indirect task call using a task queue. A task is placed on a task queue by a submitter task, which is a task that includes a SUBMIT TASK operation. A task can be submitted to a task queue for delayed invocation (see Figure 1-9).

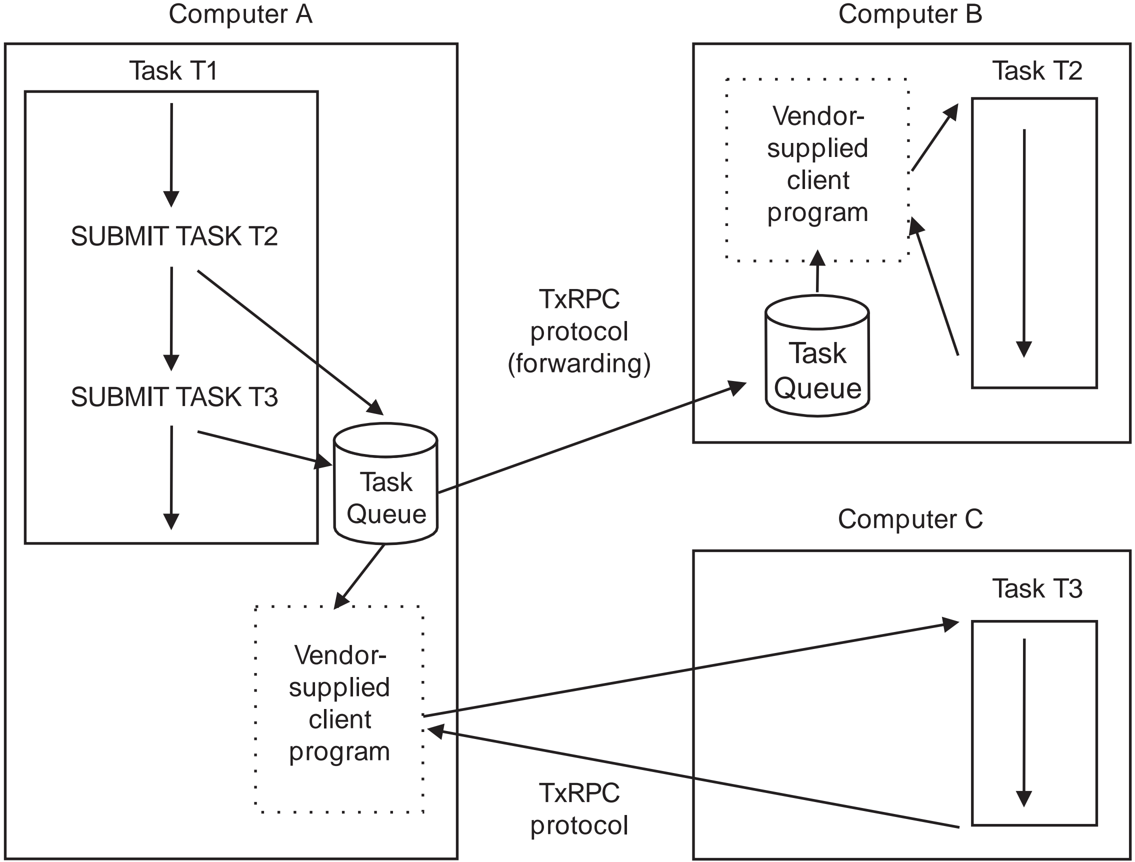


Figure 1-9 Submitted Task Invocation

A submitter task can schedule a local or remote task to be invoked at a time delay specified by the submitter task. The submitter task waits for the submit task operation to be completed, but does not wait for server task execution to be completed. No results or exceptions resulting from server task execution are returned to the submitter task. However, the submitter task receives an exception when there is a failure to submit the task to the task queue. Each vendor that implements STDL provides a special client program for dequeuing the submitted tasks asynchronously.

A submitted task can be forwarded from one computer to another using the TxRPC or DCE RPC protocol. Transactional and non-transactional forwarding are supported.

1.4.4 Transactional Task Call

At the programmer's option, a client task can call a remote composable task transactionally. The client and server tasks are in the same transaction (see Figure 1-10).

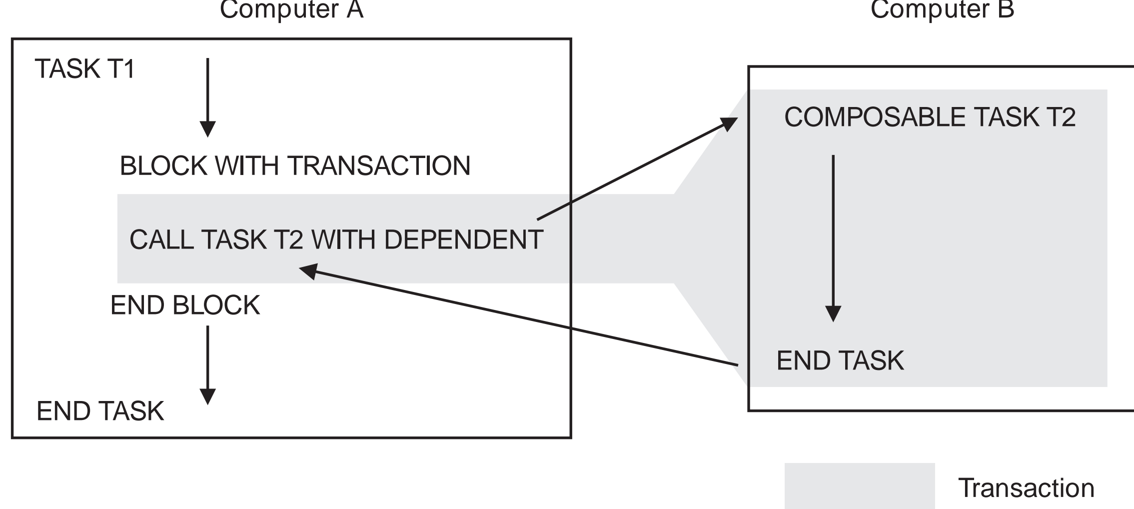


Figure 1-10 Composable Task Call

A remote composable task joins the transactional work of the client task. All updates to transactional resources by the client and server tasks are therefore atomic.

1.4.5 Non-transactional Task Call

A client task can call a remote task non-transactionally. The client and server tasks execute in separate transactions (see Figure 1-11 on page 14).

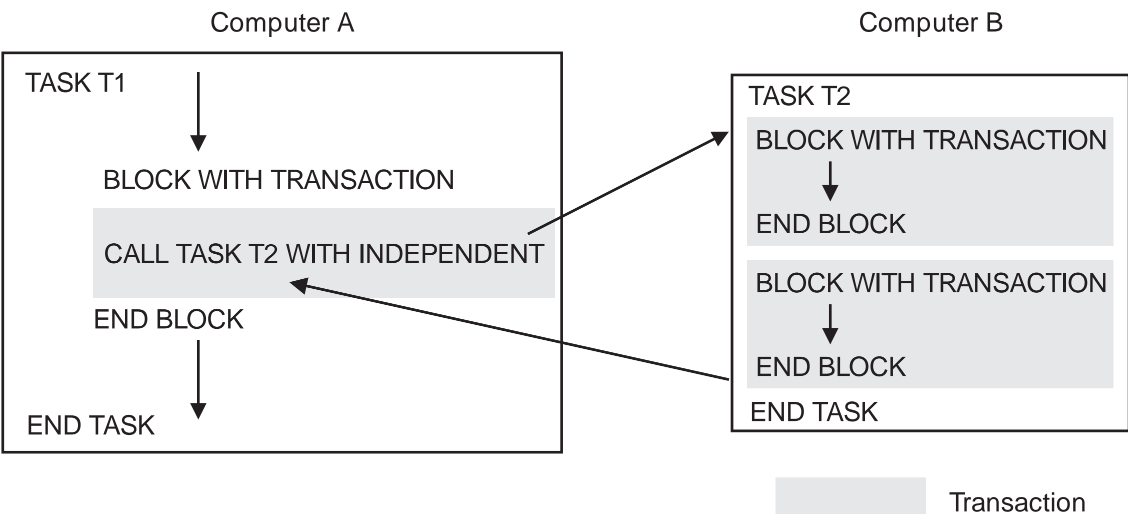


Figure 1-11 Non-composable Task Call

A remote non-composable task executes within its own transaction. The failure of a remote non-composable task does not automatically create a rollback in the client task, as it would in the case of a remote composable task call.

1.4.6 Transactional Submit Task

When the submitted task is dequeued and invoked, the submitted task is removed from the task queue either transactionally or non-transactionally, depending on whether the submitted task is composable or non-composable.

For a non-composable task, the dequeue operation occurs independently from the transaction in which the task executes. For a composable task, the dequeue operation occurs within the same transaction (started by the task dequeuer) in which the task executes.

Figure 1-12 on page 15 illustrates the two transactions involved with submitting a composable task with dependent work.

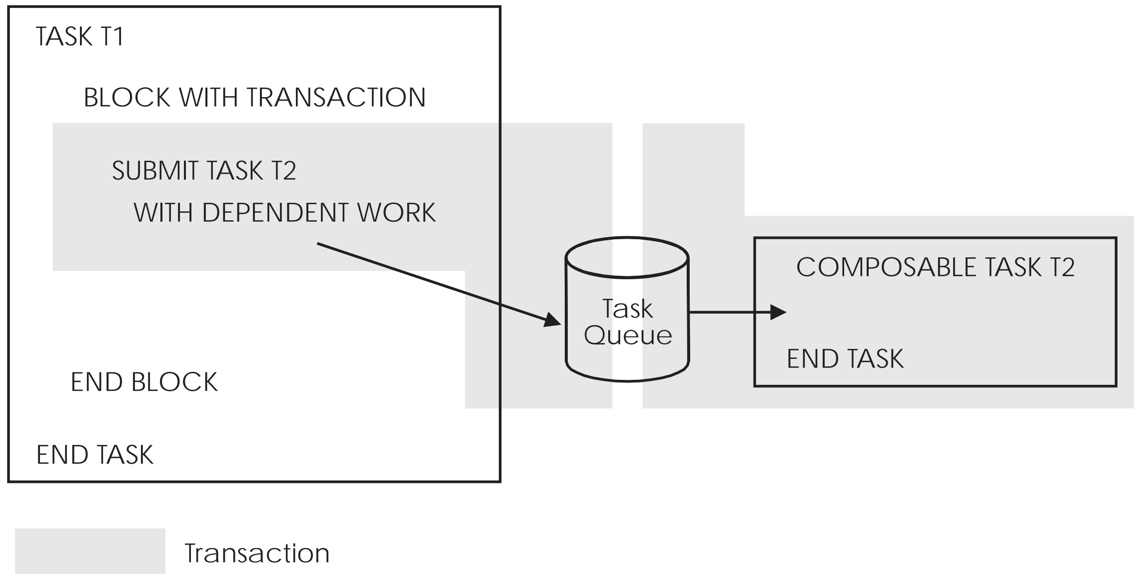


Figure 1-12 Submitting a Composable Task with Dependent Work

In Figure 1-12, the submitted task is enqueued within a transaction and dequeued within a transaction, which means both enqueue and dequeue operations are reliable. However, the programmer has the following additional options regarding transactions and submitted tasks:

- Task is enqueued transactionally and dequeued non-transactionally (DEPENDENT submit of a non-composable task).
- Task is enqueued non-transactionally and dequeued non-transactionally (INDEPENDENT submit of non-composable task).
- Task is enqueued non-transactionally and dequeued transactionally (INDEPENDENT submit of composable task).

When a submitted task that is dequeued and invoked does not complete successfully, it can be automatically retried.

1.4.7 Concurrent Execution

In STDL it is possible to call more than one task and processing procedure for parallel execution by enclosing the calls within a concurrent block (see Figure 1-13 on page 16).

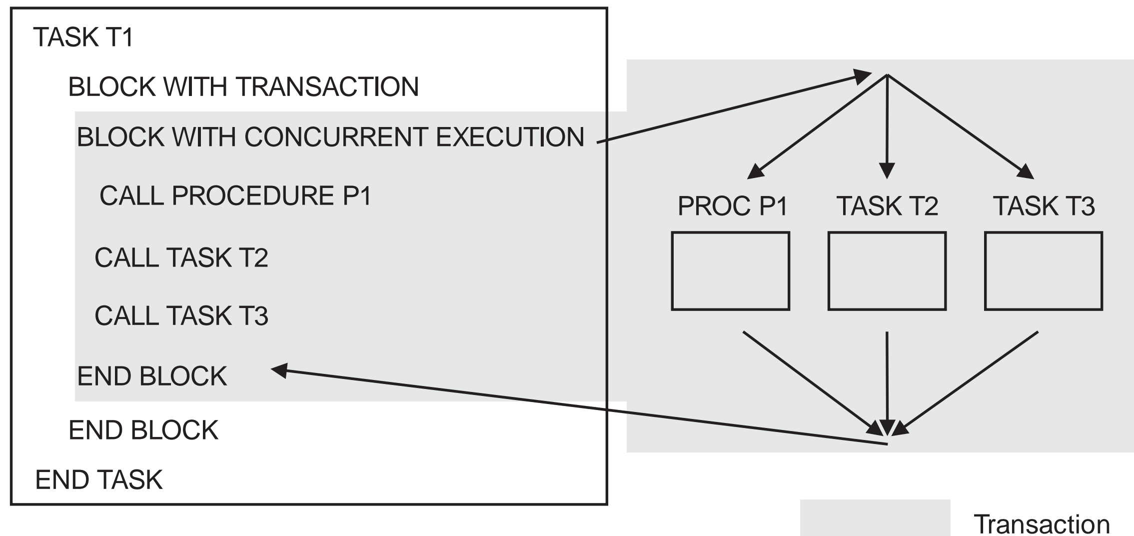


Figure 1-13 Concurrent Block Execution

This capability allows STDL to take advantage of the inherent parallelism in multi-processor systems. It also allows applications to perform work in parallel; that is, to execute multiple procedure calls concurrently instead of waiting for the results of one call before starting the next. The task calls can be remote or local.

Each of the calls within a concurrent block is executed using a separate thread of execution. The concurrent block allows multiple calls to execute concurrently with a common start and end point defined by the block begin and end.

1.4.8 Text Messages

Text messages also are created and handled using STDL. The STDL message group definition provides a way to create portable messages for applications and for use in exception handling. These messages can be retrieved by a task and displayed to the end-user or written to the audit log. The message group contains application-defined messages written in a single language (such as English or Japanese).

Application-defined exceptions are defined as messages with an associated exception class.

1.4.9 Exception Handling

Exceptions are raised when errors or problem conditions are encountered during the execution of tasks and procedures (see Figure 1-14 on page 17). When an exception is raised, an exception handler within a task is executed. Exceptions are raised by the vendor system that implements STDL and also by the application procedures.

STDL allows nested exception handling. An outer exception handler can include generic exception handling information, while an inner exception handler can include specific exception handling code. This nesting helps programmers write exception-handling code using a well-organised structure. STDL classifies exceptions into exception classes depending on what has to be done to recover from the exception rather than according to what caused the exception. This helps programmers write portable applications because they can handle exceptions without knowing the details of each product that raised exceptions.

Automatic rollback is required for any error that prevents a transaction from committing. However, both programmer-defined and other non-transactional exceptions also are supported for raising exceptions that do not require automatic rollback.

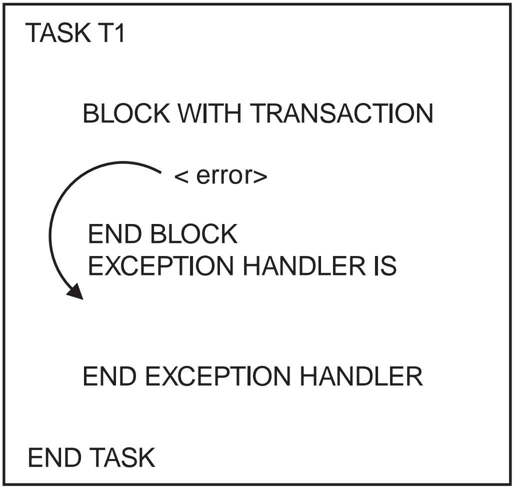


Figure 1-14 Exception Processing

When an exception is detected by the vendor’s system or raised by the application program, control is automatically transferred to the exception handler.

1.5 Database and File Access

A task can invoke one or more processing procedures. Processing procedures must be local to the task invoking them.

The task waits for the processing procedure execution to complete. Successful execution of the processing procedure returns data to the task, if any is to be returned. Unsuccessful execution returns an exception to the task (see Figure 1-15).

Processing procedures can be written in COBOL or C and typically access SQL databases. COBOL procedures can also access indexed files, sequential files and relative files. C procedures can also access stream and indexed files. Because most STDL resources are shared, processing procedures are restricted from performing operations that are inappropriate in a TP environment, such as Open/Close or Sort/Merge operations on shared files. Programmers are allowed to open and close non-transactional sequential and stream files. However, programmers are not actually prevented from accessing resources outside of those defined by the STDL environment, such as file transfer operations and use of legacy protocol.

Figure 1-15 illustrates the invocation of two processing procedures accessing two separate resource managers in a single transaction.

Figure 1-15 Processing Procedure Invocation

Upon the return from the call to the processing procedure, the run-time system determines whether or not to raise an exception.

1.6 Accessing a Display

STDL provides support for access to a display and other input/output devices, and to include the display access operations in the transaction. STDL includes the ability to optimise the data transfer and to tie the display access operations to the transaction.

STDL accesses displays and other devices using C or COBOL procedures contained in the presentation procedure group and provides send, receive and transceive calls to the presentation procedures. A presentation procedure is defined as a send procedure, receive procedure or transceive procedure. A send call passes input data to a presentation procedure; no arguments are returned. A receive call receives output data from a presentation procedure; no arguments are sent. A transceive call passes data in both directions.

A send procedure and a receive procedure can participate in a transaction (that is, these procedures can be transactional or non-transactional), but a transceive procedure cannot participate in a transaction. All calls to presentation procedures are local calls.

1.6.1 Non-transactional Presentation Procedures

Non-transactional presentation procedures can include non-transactional send and non-transactional receive as well as transceive operations. A task can invoke one or more presentation procedures (see Figure 1-16).

Figure 1-16 Non-transactional Receive and Send

The task waits for the presentation procedure execution to complete. Unsuccessful execution causes an exception to be raised to the task.

1.6.2 Transactional Presentation Procedures

At the programmer's option, STDL tasks provide some transaction control over presentation procedures. A send procedure can be automatically directed to execute only on successful completion of the transaction. A receive procedure can be directed to store input in a transactional buffer to support automatic restart of a transaction and protect against certain types of failures (see Figure 1-17).

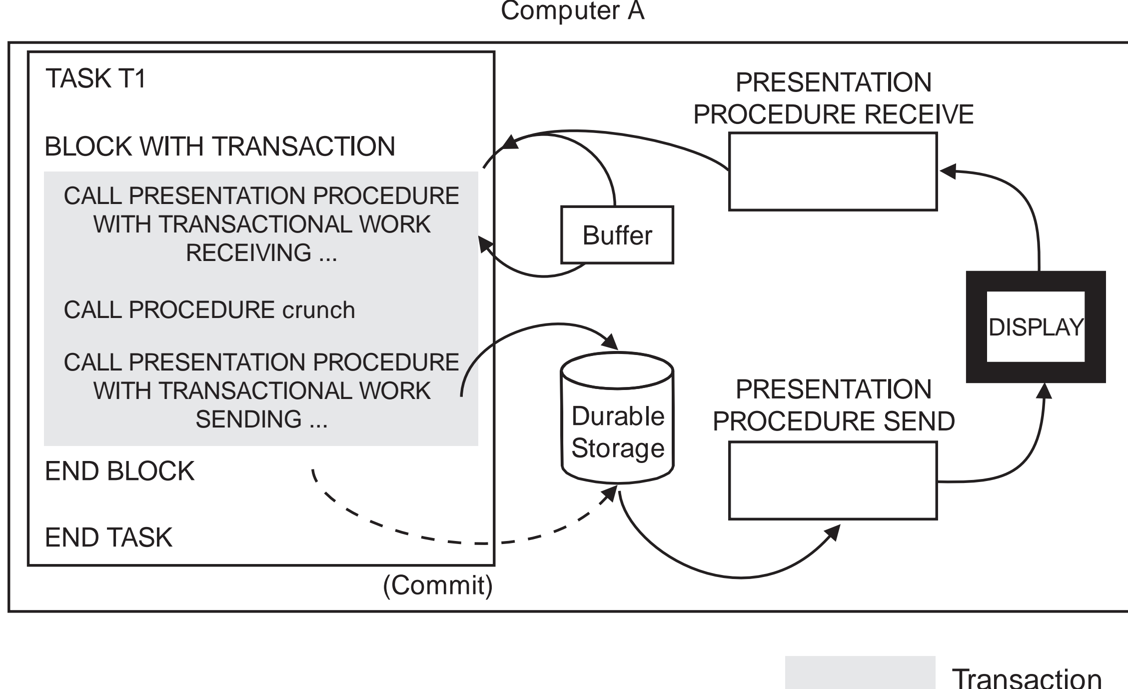


Figure 1-17 Transactional Presentation Procedures

A transactional send coordinates the execution of the send procedure with the transaction commit. The end-user sees a message, for example, only when the transaction commits.

1.7 Communications

An STDL task calls a remote task using the TxRPC protocol. The TxRPC protocol combines the DCE RPC protocol for data transfer and OSI TP protocol for two-phase commit coordination. STDL is mapped to the TxRPC protocol[1] to ensure interoperability in a multi-vendor environment.

Using the OSI TP protocol for two-phase commit coordination allows a transaction branch started by STDL to be joined with transaction branches started by other transactional communication protocols that use the OSI TP protocol.

A procedure call in the client results in an RPC request message being sent to the server where a procedure (task) is invoked. A similar sequence of events is required to return the response. To add transactional semantics to the RPC, two additional operations are required:

1. Transaction context must be propagated from the client to the server.
2. Termination must be controlled by an atomic commitment protocol.

Both of these requirements are addressed for STDL by the TxRPC protocol's use of OSI TP.

Communication between a task and a processing procedure or presentation procedure is always defined to be vendor-specific. Because STDL contains no restriction prohibiting it, a remote task call between two STDL tasks using the same vendor's TP monitor can optionally use a procedure call mechanism determined by the vendor. STDL requires standard communication protocols for external client calls (using the DCE RPC protocol) and multi-vendor remote task calls (using the TxRPC or DCE RPC protocol).

1. Although using the TxRPC API might be the simplest way to implement the TxRPC protocol, STDL does not actually require using the TxRPC API.

1.8 STDL Language Components

STDL features are implemented using separate language pieces, each of which address a separate application component (see Figure 1-18).

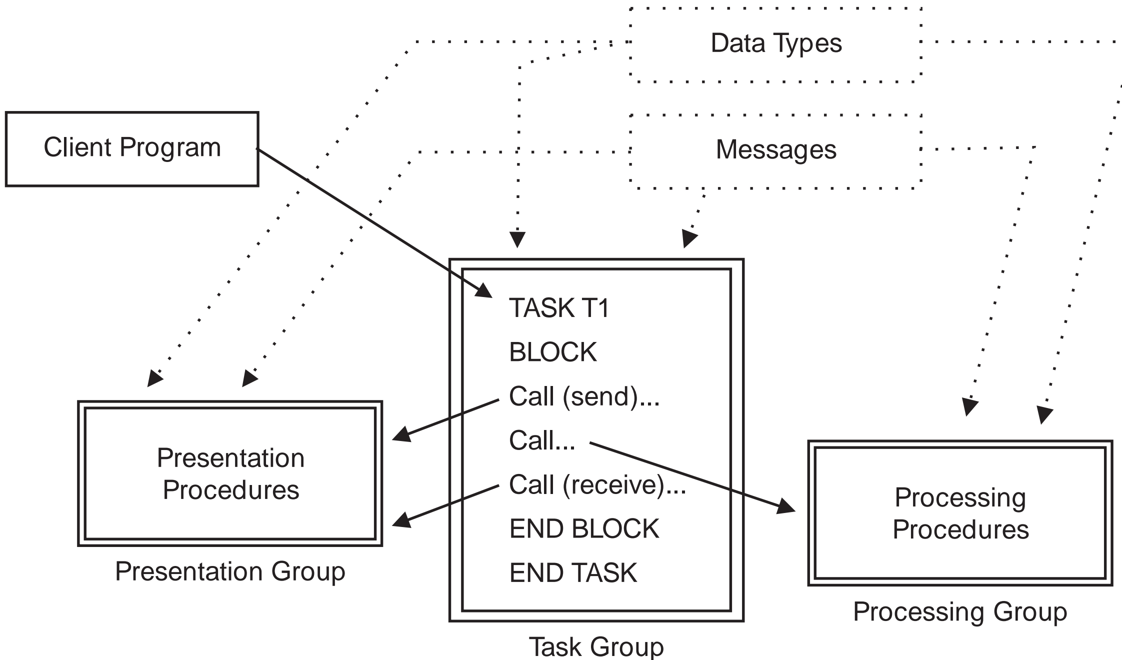


Figure 1-18 STDL Language Model

The client program shown is a customer-written client program written using C or COBOL. The data type definitions and message definitions support application requirements such as argument passing and end-user message definitions. The output from compiling data type and message files (shown with dotted lines in Figure 1-18) can be included in processing and presentation procedures.

1.8.1 STDL Source Files

STDL pieces are contained in separate source files. Source file names include the suffixes ‘‘definition’’, ‘‘specification’’ and ‘‘procedure’’, which help distinguish the different types of source files. A definition refers to a named collection of STDL syntax elements that are compiled to create an executable or manipulatable run-time application entity. A specification refers to a named collection of STDL syntax elements that are compiled to create a service (or call) interface to a group of related procedures. A procedure refers to a procedure written using STDL, C or COBOL.

When an STDL task is compiled, the interface definition, client stub and server stub for the RPC communication are produced automatically.

STDL includes the following source files:

- Data Type Definition

  Defines local variables and procedure call arguments using record structures.

- Message Definition

  Defines status and event messages in multiple languages for retrieval by an application.

- Task Group Specification

  Specifies the interface to a logical collection of STDL tasks.

- Processing Group Specification

  Specifies the interface to a logical collection of C or COBOL procedures used for database access, file access and computation.

- Presentation Group Specification

  Specifies the interface to a logical collection of C or COBOL procedures used for access to input/output devices such as displays, bar code readers, gas pumps, and so on. (A vendor-specific mechanism can be used to replace the presentation group specification with a vendor-supplied forms package.)

- Task Definition

  Defines the task procedure by specifying the execution flow of one or more transactions, sequences resource manager access and procedure calls, and handles exceptions.

In addition, the STDL language assumes the existence of the following types of procedures:

- Processing Procedures

  Standard C or COBOL procedures for database access (using embedded SQL), file access and computation.

- Presentation Procedures

  Standard C or COBOL procedures for display manipulation and device access.

The major part of STDL syntax is contained within the task definition. Table 1-1 on page 24 shows that all major STDL statements are task definition statements, except for those contained within the ''Declaration'' row.

Classification	Function	Major Statement
Data manipulation	Manipulates field contents.	• ASSIGNMENT
Declaration	Defines data types, messages and call interfaces.	• DATA TYPE • MESSAGE GROUP • TASK GROUP • PROCESSING GROUP • PRESENTATION GROUP
Task attribute definition	Defines task attributes, such as the task name and arguments.	• TASK • COMPOSABLE TASK
Transaction control	Defines a transaction demarcation. Restarts a transaction. Commits or rolls back a transaction.	• BLOCK WITH TRANSACTION • RESTART TRANSACTION • WITH COMMIT/WITH ROLLBACK
Database and file access	Calls a processing procedure.	• CALL PROCEDURE
Communications access	Calls a task. Requests a task invocation schedule. Cancels a scheduled task invocation.	• CALL TASK • SUBMIT TASK • CANCEL SUBMIT
Display access	Calls a presentation procedure.	• CALL PRESENTATION PROCEDURE
Other resource access	Stores or retrieves temporary data. Stores audit information.	• ENQUEUE/READ QUEUE/DEQUEUE RECORD • AUDIT
Exception handling	Declares an exception handler. Generates an exception. Passes along an exception. Converts an exception code to a message.	• EXCEPTION HANDLER • RAISE EXCEPTION • RERAISE EXCEPTION • GET MESSAGE
Execution flow control	Causes a conditional branch. Repeats an operation. Executes a concurrent operation. Causes a jump. Exits from a task. Exits from a block.	• IF THEN ELSE • WHILE • BLOCK WITH CONCURRENT EXECUTION • GOTO • EXIT TASK • EXIT BLOCK

Table 1-1 Major STDL Statements

Figure 1-19 on page 25 illustrates the required contents of an STDL source file when compiling all the components together: one or more data type definitions, followed by the task group specifications, processing group specifications, and presentation group specifications that

reference those data type definitions; and any message group definitions, followed by the task definitions that reference those preceding definitions and specifications.

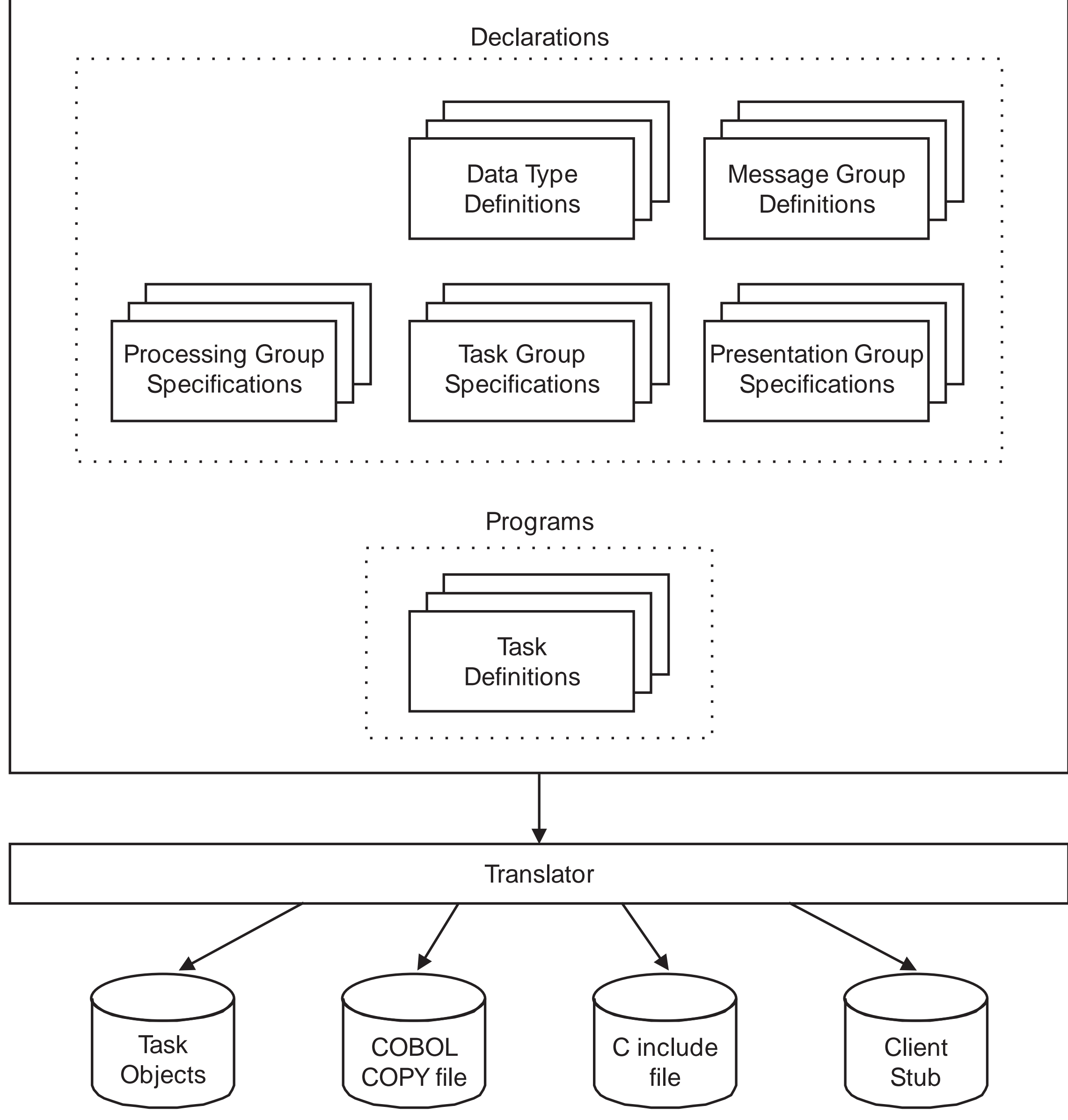


Figure 1-19 STDL Source Files

In order to produce the executable task objects, client stubs and header files, it is necessary to include within the task definition translation operation all related STDL source files.[2] In particular, group specifications must be included for all calls to procedures and tasks in other groups.

2. Client stubs are produced using the DCE RPC Interface Definition Language (IDL) or another mechanism that produces equivalent results.

1.8.2 Execution Environment

After an STDL application is written and compiled, the application is executed within an environment defined to provide TP applications with the support they need. Execution is regulated and controlled by environmental information, which specifies various limits, such as transaction timeout and fault limits, and defines run-time entity attributes that are set by a system manager.

1.8.3 The STDL Specification

The STDL specification describes language syntax and semantics, details the environment in which STDL applications execute, and maps STDL to the X/Open TxRPC or DCE RPC protocol.

STDL's mapping to the TxRPC protocol provides interoperability for distributed processing, supporting both transactional and non-transactional remote procedure calls (RPC). STDL allows programmers to develop portable, interoperable applications that can be implemented across multiple vendor platforms. STDL is compatible with the X/Open DTP model.

Chapter 2

Introduction to Part 1

2.1 Scope

Part 1, STDL Specification specifies the syntax and semantics of the STDL definitions and associated programs that can be used to create transaction processing (TP) applications. This includes:

- a data type definition for declaring the format of records used for workspaces and record queues in TP applications
- a message definition language for declaring message codes and associated message texts
- a presentation group specification for declaring the interface to a set of customer-written presentation procedures written using a standard programming language for interface to a display for tasks
- a processing group specification for declaring the interface to a set of processing procedures written in a standard programming language for computation and database I/O for tasks
- a task group specification for declaring the interface to a set of tasks
- a task definition language for declaring the execution flow within a TP application
- a set of rules for writing processing procedures for file and SQL access
- a set of rules for writing presentation procedures using vendor-supplied forms systems and customer-written procedures
- a set of rules for writing client programs using standard COBOL and C
- a set of rules for using standard COBOL and C in TP applications.

This part defines the basic concepts and facilities that must be implemented by a conforming implementation in order to support STDL applications. This includes:

- transaction manager
- file system that conforms to standard COBOL file semantics, with the addition of transaction semantics
- stream files that conform to standard C stream file access, without transaction semantics
- indexed files that conform to standard ISAM for access from standard C, with the addition of transaction semantics
- SQL environment
- record queues
- delayed task submission facilities.

This part prescribes a convention for porting TP applications from one conforming implementation to another.

2.2 Organisation of Part 1

This part is organised as follows:

- Chapter 1 on page 3 provides an informative introduction to STDL.
- Chapter 2 on page 27 defines the scope of this part, the notations and conventions used, and discusses general conformance criteria.
- Chapter 3 on page 37 defines and presents concepts used in STDL applications.
- Chapter 4 on page 75 defines language elements that occur in several parts of STDL specifications and definitions.
- Chapter 5 on page 121 defines the syntax and semantics of STDL data type definitions.
- Chapter 6 on page 133 defines the syntax and semantics of STDL message group definitions.
- Chapter 7 on page 141 defines the syntax and semantics of STDL presentation group specifications.
- Chapter 8 on page 151 defines the syntax and semantics of STDL processing group specifications.
- Chapter 9 on page 157 defines the syntax and semantics of STDL task group specifications and client stubs.
- Chapter 10 on page 167 defines the syntax and semantics of STDL task definitions.
- Chapter 11 on page 249 defines the semantics of processing procedures.
- Chapter 12 on page 257 defines the semantics of presentation procedures.
- Chapter 13 on page 263 defines the semantics of client programs.
- Chapter 14 on page 269 discusses the use of standard COBOL in TP applications.
- Chapter 15 on page 279 discusses the use of standard C in TP applications.
- Appendix A on page 481 specifies architectural constants.
- Appendix B on page 487 specifies the audit record definition.
- Appendix C on page 489 specifies exception classes.
- Appendix D on page 501 contains a complete list of STDL syntax productions.

Unless stated otherwise, all chapters and appendices are normative.

2.3 Syntax Notation

The STDL syntax is described using a modified BNF. All non-terminal syntax elements are enclosed in angle brackets (< >). All words reserved to STDL are written in upper-case letters. If a reserved word is enclosed in square brackets ([ ]), then the word can be omitted without changing the semantics. These words are called noise words.

The syntax is presented in a series of productions. Each production starts with a non-terminal syntax element followed by a double colon (::). This is followed by one or more definitions for the syntax element. If there are multiple definitions for a syntax element, they are separated by a vertical bar (|). If a definition is "<nil>", then the syntax element being defined can be omitted when used in other productions.

A definition may consist of a syntax element that can be repeated. This is indicated by the use of ellipses: ("..."). If the repetitions are to be separated by commas, then a comma ellipse is used: (", ..."). If the repetitions are to be separated by periods, then a period ellipse is used: (". ...").

Non-terminals ending in –1, –2 or –3 distinguish among literals, identifiers or fields of the same type within a single production. Syntax Rules and General Rules for each production describe the specific characteristics of each literal, identifier or field.

For example, consider the following syntax (this is not actual STDL syntax):

```
<transaction-block> ::
        BLOCK [WITH] TRANSACTION
                <statement-list>
        END [BLOCK] ;

<statement-list> ::
          <nil>
        | <statement> ...
```

These two productions define that a transaction block starts with the reserved words BLOCK WITH TRANSACTION and the WITH is a noise word. The statement list in the block consists of zero or more statements. The non-terminals defined are <transaction-block> and <statement-list>. <statement> is a non-terminal that is defined elsewhere. The reserved words are BLOCK, WITH, TRANSACTION and END.

2.4 Conventions

STDL syntactic and semantic elements are defined in terms of:

NAME
The name of an element and a brief statement of the purpose of an element.

SYNOPSIS
Shows the syntax of the element using the notation described in Section 2.3 on page 29.

SYNTAX RULES
Lists syntactic constraints the syntax element must satisfy that are not included in the format.

GENERAL RULES
Lists the run-time effects of a syntax element.

CONFORMANCE NOTES
Documents features that an implementation does not have to implement in order to be conformant at a less than full support level. These notes are made according to the capabilities listed in Section 2.5 on page 32.

EXCEPTIONS
Lists system and application exceptions and their exception classes.

IMPLEMENTOR NOTES
Lists any restrictions that an implementation can place on the STDL application.

NOTES
Provides informative clarification.

SEE ALSO
Provides pointers to sections of this part describing non-terminal syntax elements shown in the format.

In the Syntax Rules, the term ‘‘must’’ defines conditions that are required to be true of syntactically conforming STDL language. The term ‘‘cannot’’ defines conditions that are required to be false of syntactically conforming STDL language. An implementation flags syntax errors when STDL definitions do not satisfy ‘‘must’’ and ‘‘cannot’’ requirements of syntax rules.

In the General Rules, the terms ‘‘must’’ and ‘‘cannot’’ define conditions that are tested at run time during the execution of STDL definitions. If a ‘‘must’’ condition is not met, or if a ‘‘cannot’’ condition is true, the implementation returns an exception.

Implementor Note:

An implementation does not have to return an error for every violation of a syntax rule or a general rule.

2.4.1 Definitions

The following terms used in this document are defined in other specifications:

Standard COBOL or COBOL
Refers to ISO 1989: 1985 (including ISO 1989/Amendment 1).

Standard C or C
Refers to ISO C.

Standard programming languages
Refers to standard COBOL and standard C.

Standard SQL or SQL
Refers to the referenced X/Open SQL specification.

RTI
Refers to the Remote Task Invocation (RTI) protocol defined in the referenced X/Open TxRPC specification.

Standard ISAM
Refers to the referenced X/Open ISAM specification.

Standard XFTAM
Refers to the referenced X/Open XFTAM specification.

TxRPC Specification
Refers to the referenced X/Open TxRPC specification.

DCE RPC
Refers to the referenced X/Open DCE RPC specification.

Other terms used in this document are defined in Chapter 3 on page 37 and in the following list:

concurrent task
Two or more tasks whose execution in a TP system overlaps in time.

entity
An object within the TP system that can have attributes and characteristics managed within the environment.

forms system
A vendor-defined software system that manages the interaction between a TP system and its displays.

Profile A
An STDL Level 1, 2, 3 or 4 implementation on a computer system with a display, such as a PC or workstation.

Profile B
An STDL Level 3 or 4 implementation on a computer system without a display, such as a departmental server or host.

2.5 Conformance

This section applies to both Part 1, STDL Specification and Part 2, STDL Environment, Execution and Protocol Mapping Specification.

STDL defines four levels of support (1 to 4) according to the increasing complexity of implementation. Level 1 is the easiest to implement; Level 4 the most difficult.

Each level supports a number of capabilities called *functional capabilities*. The functional capabilities are organised into eight major categories:

1. Transactions
   - single resource transactions (transactional access to a single resource)
   - multi-resource transactions (concurrent transactional access to multiple resources on one TP system)
   - restartable transactions.
2. Communications
   - non-transactional communications between TP systems
   - transactional communications between TP systems.

   Note: Local communications (within the same TP system) are not subject to conformance.
3. Displays
   - single, logical display
   - broadcast
   - dynamic display.

   Notes:

   1. <call-presentation-procedure> is not possible without display support.
   2. <send-display> is not possible without dynamic display.
4. Task Submission
   - local task queue (includes task submit and cancel)
   - task forwarding, without transactions
   - task forwarding, with transactions
   - task dequeuer (includes task enqueuer).
5. Resource Manager
   - SQL
   - shared workspace
   - indexed files
   - relative files
   - sequential files
   - stream files

— audit writes

— record queues

— dequeue/read queue record with wait

— transactional send and receive

— private transactional workspaces.

6. Multi-tasking

— accept incoming task calls

— support multiple concurrent task execution

— support concurrent block execution (within a task).

7. Programming Languages

— STDL

— COBOL (Standard COBOL)

— C (Standard C).

8. Environment

— high/low water mark (task resource management)

— processing procedure fault count

— multiple entries in a distribution list

— customer-written presentation procedures.

2.5.1 Categories

Figure 2-1 shows the eight categories and the capabilities for each.

Transactions
Single Resource Transactions
Multi-Resource Transactions
Restartable Transactions

Communications
Non-Transactional Communication
Transactional Communication

Displays
Display
Broadcast
Dynamic Display

Task Submission
Local Task Queue
Forwarding, no Transaction
Forwarding, with Transaction
Task Dequeuer

Multi-Tasking
Accept Incoming Task Calls
Multiple Conc. Task Execution
Concurrent Block

Resource Manager
SQL
Shared Workspace
Indexed File
Relative File
Sequential File
Stream File
Audit
Record Queue
Dequeue/Read Queue with Wait
Transactional Send and Receive
Private Transactional Workspace

Environment
High/Low Water Mark
Processing Procedure Fault Count
Multi-Entry Distribution List
Customer-Written Presentation Procedure

Programming
STDL
COBOL
C

Figure 2-1 Conformance Categories and their Functional Capabilities

2.5.2 Levels

The support level and their associated functional capabilities are shown in Table 2-1. Table 2-1 lists the levels:

	Levels			
Capabilities	**1**	**2**	**3**	**4**
Single Resource Transaction	Y	Y	Y	Y
Multi-resource Transaction	N	N	Y	Y
Restartable Transaction	N	N	Y	Y
Non-transactional Communication	Y	Y	Y	Y
Transactional Communication	N	N	N	Y
Display	Y	Y	Y	Y
Broadcast	N	N	N	Y
Dynamic Display	Y	Y	Y	Y
Local Task Queue	N	Y*	Y	Y
Forwarding, no Transaction	N	Y	Y	Y
Forwarding, with Transaction	N	N	N	Y
Task Dequeuer	N	N	Y	Y
SQL	Y	Y	Y	Y
Shared Workspaces	N	N	Y	Y
Indexed Files	N	Y**	Y	Y
Relative Files	N	Y**	Y	Y
Sequential Files	Y**	Y**	Y	Y
Stream Files	Y	Y	Y	Y
Audit Write	N	Y	Y	Y
Record Queue	N	N	Y	Y
Dequeue/Read Queue with Wait	N	N	N	Y
Transactional Send and Receive	N	N	Y	Y
Private Transactional Workspace	N	N	Y	Y
Accept Task Calls	Y	Y	Y	Y
Multiple Concurrent Task Execution	Y	Y	Y	Y
Concurrent Block	N	N	N	Y
STDL	Y	Y	Y	Y
COBOL	Y	Y	Y	Y
C	Y	Y	Y	Y
High/Low Water Mark	N	N	N	Y
Processing Procedure Fault Count	N	N	N	Y
Multi-entry Distribution List	N	N	N	Y
Customer-written Presentation Procedure	Y	Y	Y	Y

Table 2-1 Level List by Difficulty of Implementation

* Indicates that a task submit must use WITH INDEPENDENT WORK.

** Indicates that these files are not transactional.

Y The capability is supported for a level.

N The capability is not supported for a level.

Notes:

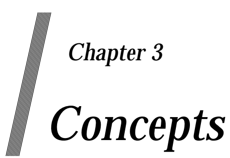
1. If Level 4 is implemented in a non-communicating environment, the following capabilities are not required:

   — communications (non-transactional and transactional)

- — task forwarding (with and without transactions)
- — accept task calls
- — distribution list.

2. The significant differences among the levels are:

- — multi-resource transaction manager and restartable transactions
(not supported by Level 1 or Level 2)
- — distributed transactions
(supported only by Level 4)
- — local task queue
(not supported by Level 1)
- — task submission request forwarding without transaction
(not supported by Level 1; supported by Level 2, Level 3 and Level 4)
- — task submission request forwarding with transaction
(supported only by Level 4)
- — task enqueuer
(supported by Level 3 and Level 4)
- — shared workspaces, indexed files, relative files, audit writes, record queue, transactional send and receive, and private transactional workspace
(not supported by Level 1)
- — record queue, transactional send and receive, and private transactional workspace
(not supported by Level 1 and Level 2)
- — dequeue/read queue with wait
(supported only by Level 4)
- — concurrent block
(supported only by Level 4)
- — environmental capabilities
(supported only by Level 4).



Chapter 3

Concepts

3.1 Transaction

A *transaction* is a sequence of operations that is atomic. The following sections define the semantics of transactions for this specification.

3.1.1 Transactional and Durable Resources

A *transactional operation* is the execution of a function whose effects are undone if the transaction in which the operation was executed does not complete successfully. The operation of undoing the effects of transactional operations in a transaction is called *rolling back* the transaction. If the transaction completes successfully, the effects of the transactional operations cannot be undone using a rollback. This is called *committing* the transaction. When a transaction commits, the effects of transactional operations are available to other transactions.

The results of a transactional operation are stored in a resource. The resource used to store the results of a transactional operation is known as a *transactional resource*. A component of a TP system that provides transactional operations to store results in a transactional resource is a *transactional resource manager*. All transactional operations are executed within an STDL transaction.

A *non-transactional operation* is the execution of a function whose effects are *not* undone if the transaction in which the operation was invoked does not complete successfully. The effects of a non-transactional operation are stored in a *non-transactional resource* by a *non-transactional resource manager*.

A *durable resource* is a resource whose data can survive system failure. For a transactional resource, only committed data can survive system failure. *System failure* is any event that causes the system to stop executing and the contents of volatile storage to be lost. A *non-durable resource* is a resource whose data is lost in the event of a system failure or other system shutdown. The data maintained within the resource is not available when the system is restarted.

Table 3-1 shows the types of resources provided by the TP system.

	Durable Resource	**Non-durable Resource**
Transactional Resource	Yes	Yes
Non-transactional Resource	Yes	Yes

Table 3-1 TP System Resource Types

In STDL applications, all transactional operations occur within a transaction. After a transaction within a task has completed (by either committing or rolling back), a new transaction is started before the task (or a processing procedure called by the task) accesses a transactional resource.

3.1.2 Isolation Level

An *isolation level* reflects the degree of interference a transaction tolerates from other concurrently executing transactions.

Concurrently executing transactions are transactions whose executions overlap in time. An execution E for a set of transactions is concurrent if:

- A transaction, T1, starts execution.
- A transaction, T2, starts execution before T1 completes.

A *serialisable execution* is defined to be an execution of the operations of concurrently executing transactions that produces the same effect as some *serial execution* of those same transactions. A serial execution is one in which each transaction completes execution before the next transaction is started. A serialisable execution isolates a transaction from inconsistencies that can be caused by concurrency. Different isolation levels can be defined by describing different degrees of interference between transactions that can be allowed. The different degrees of interference can be described using scenarios. The scenarios described below discuss transactions manipulating data items. The term data item refers to shared data manipulated by a transaction, such as a record within a file, a row within a table, or a shared workspace. The following are examples of the degrees of interference that can occur within concurrently executing transactions:

1. *Dirty Read*

   Occurs when one transaction reads the uncommitted results of another transaction. A dirty read problem arises in an execution E of a set of transactions if:

   a. A transaction, T1, inserts or updates a data item X.

   b. A transaction, T2, reads X before transaction T1 commits.

   c. Transaction T1 subsequently rolls back, or modifies the data item X.

   A dirty read can cause a non-serialisable execution because T2 can read data that will never be committed. In any serial execution of T1 and T2, transaction T2 would always read committed data. Therefore, E is not serialisable.

2. *Non-repeatable Read*

   Occurs when a transaction retrieves two different values from two read operations of the same data item. The non-repeatable read problem arises in an execution E of a set of transactions if:

   a. A transaction, T1, reads a data item X.

   b. A transaction, T2, updates or deletes X.

   c. Transaction T2 commits.

   d. Transaction T1 reads X and obtains a different value than the first read.

   A non-repeatable read problem can cause a non-serialisable execution as follows: in any serial execution of T1 and T2, the first and subsequent reads performed by transaction T1 would produce the same results. Therefore, E is not serialisable.

3. *Phantom*

   Occurs when a transaction notices the presence or absence of a data item at different points in its execution. A phantom arises in an execution E of a set of transactions if:

   a. A transaction, T1, inserts or updates a data item X.

b. A transaction, T2, reads a set of data items and the set that it reads does not include X.

   If T2 had read the set after transaction T1 inserted or updated X, then X would have been in the set that T2 read.

c. T1 and T2 have conflicting access to some data item such that T1's access precedes T2's access.

A phantom can cause a non-serialisable execution as follows: Due to c., T1 executes before T2 in any serial execution that is equivalent to E. In the serial execution, T1 inserts or updates X before T2 executes. This implies that T2 should read X, which did not happen in E. Therefore, E is not serialisable.

Four different isolation levels are defined. The isolation levels differ from each other with respect to the amount of interference they allow between transactions. Table 3-2 shows the isolation levels and the types of interference they do and do not allow.

Level	Dirty Read	Non-repeatable Read	Phantom
0	Allowed	Allowed	Allowed
1	Not allowed	Allowed	Allowed
2	Not allowed	Not allowed	Allowed
3	Not allowed	Not allowed	Not allowed

Table 3-2 Transaction Isolation Level

3.2 TP System Model

A *TP system* is an implementation-specific entity with a unique name that executes tasks. Figure 3-1 shows the entities within a TP system and their relationships.

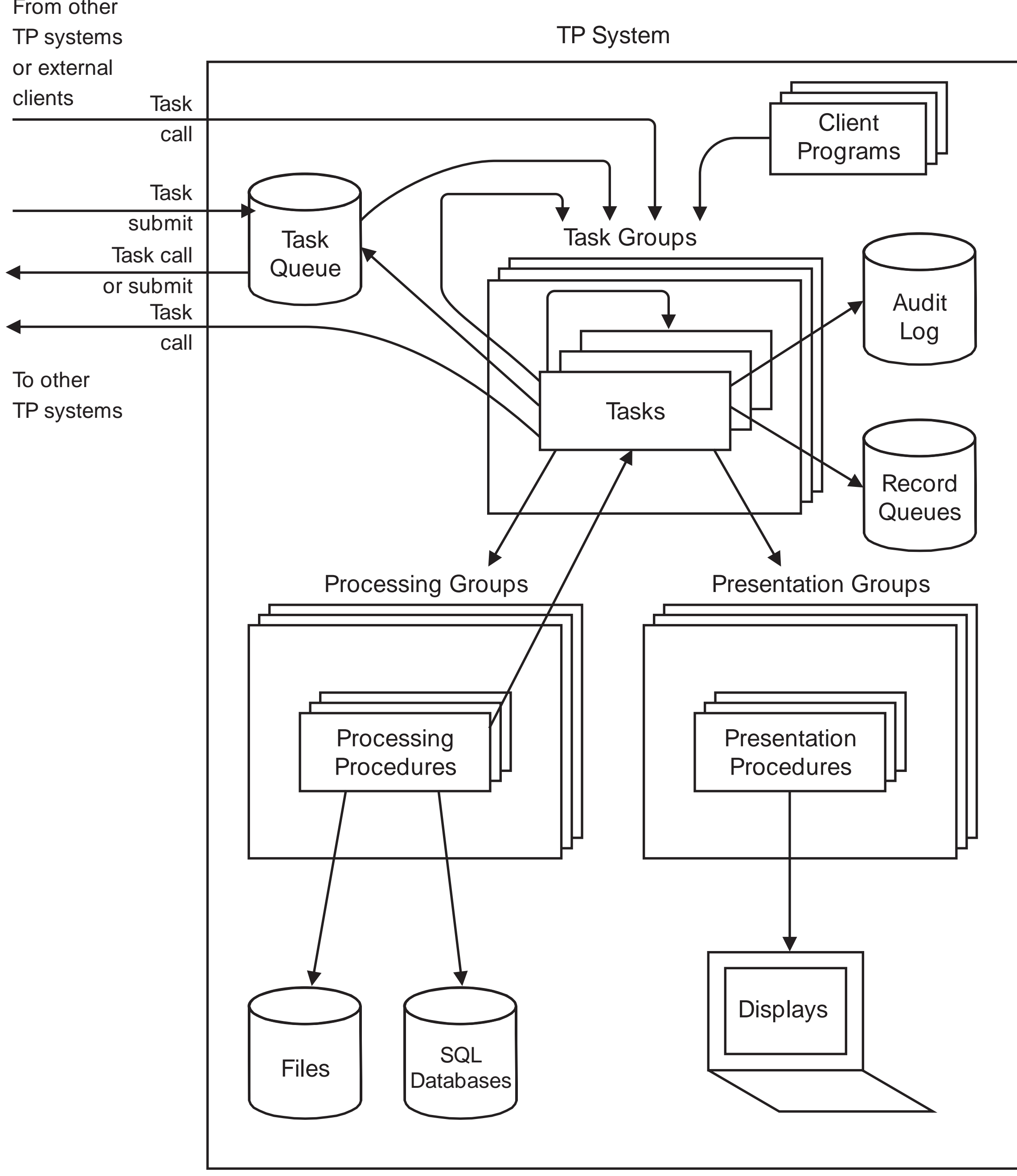


Figure 3-1 TP System Model

A TP system can send messages to and receive messages from other TP systems or receive messages from external clients (see Section 3.3.1 on page 47) to perform task invocations. The two types of task invocation messages are:

- Task call messages

  Request that a task start execution and return the results of the task execution.

- Task submit messages

  Request that a task be called later. No results of the task are returned.

A *distributed transaction* is a transaction that contains operations performed by more than one TP system. A *root task* is a task that starts a transaction. A *root TP system* is the TP system on which the root task starts a transaction. A *transaction tree* is the set of all tasks that participate in a transaction. The root task is the root of the transaction tree. There is a unique path from the root task to each task in the transaction tree.

3.2.1 Task

A *task* is a named sequence of operations defined using a <task-definition>. Tasks are invoked by entities called *client programs.* A task controls the flow of execution within a TP system.

A task consists of one or more transactions. All operations for a transaction are contained within a *transaction block.*

Two types of tasks can be defined: a *composable task* executes within a previously-started transaction, while a *non-composable task* starts a new transaction root.

A task starts as a single thread of execution but can execute parallel threads of execution. The operations for these parallel threads are contained within a *concurrent block* and execute within a single transaction.

A task can:

- call processing procedures that perform computations, access files and access databases
- call presentation procedures that output data to and get data from a display
- enqueue, dequeue and read queue data
- invoke other tasks (the results of a task calling itself, either directly or indirectly, are undefined)
- remove a task from a task queue
- raise and handle exceptions
- evaluate conditions and execute loops
- explicitly roll back or commit transactions
- write to an audit log
- translate message codes into message texts.

Implementor Note:

An implementation can prohibit a task from calling itself directly or indirectly. Refer to the implementor note in Section 10.3.5 on page 196 for more information.

3.2.2 Task Group

A *task group* is a named set of tasks defined using a <task-group-specification>. All the tasks in a task group are executable on each TP system that implements the task group unless specifically disabled using environmental information. A <task-group-specification> specifies the interface through which a task can be invoked.

Each task group is uniquely identified using a universal unique identifier (UUID). The UUID is used to ensure that the client calls a task in the correct task group (refer to the referenced X/Open DCE RPC specification for further information on UUIDs).

Each task group also has a version number, which consists of a major and minor number. The version number is used to ensure that the client calls a task in a compatible task group.

A task group defines the scope of context sharing between executions of tasks. The context that a task creates can be used by another task and is called the *task group context*. Two tasks share the same task group context if one of the following conditions is met:

- One task execution was caused by the other task execution through task calls to composable tasks in the same task group on the same TP system.
- Both task executions are caused by another task execution through task calls to composable tasks in the same task group on the same TP system.
- The task executions are caused by task calls made sequentially on the same transactional dialogue (refer to Section 3.3.3 on page 54 for more information).

The task group context consists of:

- transaction context (retained for the duration of the transaction)
  - — shared workspace context (see Section 10.2.3 on page 177)
  - — record queue context (see Section 10.3.19 on page 226 and Section 3.6.3 on page 60)
  - — processing group context (see Section 3.2.4 on page 43 and Section 11.6 on page 252)
  - — transactional dialogue context (see Section 3.3.3 on page 54)
- non-transaction context (minimally retained for duration of a task execution)
  - — presentation group context (see Section 3.2.7 on page 45).

Implementor Note:

> An implementation can execute two tasks in the same task group context if the two tasks are executing in the same transaction on the same TP system, even if the tasks are in different task groups, or are invoked by different task groups or task groups on a different TP system. However, this use of context may not be portable to another vendor's implementation.

3.2.3 Processing Procedure

A *processing procedure* is a named sequence of serial processing operations defined using standard COBOL or C called by tasks. A processing procedure comprises:

- *top-level processing procedures* (whose interfaces are defined using a processing procedure group specification)
- *nested processing procedures* (which are procedures called by top-level processing procedures or other nested processing procedures whose interfaces are defined using standard COBOL or C).

These two types of processing procedures are defined because STDL access and context rules differ depending on the type of processing procedure.

Processing procedures handle computation, database access and file access for tasks. A processing procedure executes within the context of a single transaction started by a task execution. A processing procedure can call a task.

A processing procedure can:

- perform computations
- perform data manipulation
- access files (see Section 14.3.1 on page 272 for allowed file I/O operations for COBOL, and Section 15.3.1 on page 282 for allowed file I/O operations for C)
- access SQL tables (see Section 11.8 on page 255 for allowed SQL operations)
- transfer files (see Section 11.9 on page 256 for a discussion of file transfer operations)
- raise exceptions (top-level processing procedures only)
- call a nested processing procedure (the results of a procedure calling itself, either directly or indirectly, are undefined)
- call a task (see Section 13.3 on page 265).

Implementor Note:

An implementation can prohibit a procedure from calling itself directly or indirectly.

3.2.4 Processing Group

A *processing group* is a named set of top-level processing procedures defined using a <processing-group-specification>. All the top-level processing procedures in a processing group are executable on each TP system that implements the processing group unless specifically disabled using environmental information. A <processing-group-specification> defines the interface through which a top-level processing procedure can be called.

Top-level processing procedures in a processing group are all written using a single programming language: COBOL or C. A top-level processing procedure can call a nested processing procedure written using the same or a different language.

A processing group defines the scope of context sharing for the execution of processing procedures. The context that a processing procedure creates can be used by another processing procedure and is called the *processing group context*.

Two processing procedure executions can share the same processing group context if all of the following conditions are met:

- Both executions are in the same transaction.
- Both executions are caused by calls from tasks sharing the same task group context. These are calls to top-level processing procedures and nested calls to nested processing procedures.
- The two top-level processing procedures involved in these executions are in the same processing group.
- Both executions are on the same TP system.
- Both procedures are written using the same programming language.

The processing group context consists of:

- file context
- SQL context.

The processing group context is maintained until the end of the transaction in which the context is created (see Section 11.6 on page 252).

Implementor Notes:

1. An implementation can execute two top-level processing procedures in the same processing group context if the two top-level processing procedures are executing in the same transaction called by tasks in the same task group on the same TP system, even if the top-level processing procedures are in different processing groups.
2. The results of re-entering a processing group within the same transaction are not defined; for example, a deadlock can occur.

3.2.5 Display

A *display* is a named logical device that provides input to the execution of a task, receives output from the execution of a task, or both.

How a display is mapped to a physical device or devices is implementation-specific. A display is defined using environmental information.

A display is accessed by a task in one of two modes:

- Normal Mode

  In normal mode, a task calls a presentation procedure to send data to and receive data from a single display in a sequence of one or more operations. This display is assigned to the task execution when the execution begins. This display can be assigned by the client program or using a <send-display> task attribute.

  A client can choose not to assign a display to a task, or a task can be executed in an environment in which no display is assignable to the task.

  When a display is assigned to a task using a <send-display> attribute, the display is used for output only.

- Broadcast Mode

A broadcast send causes a message to be given to the display without regard to any current operation on the display. After a message is sent to the display, any interrupted operation is resumed. This display is specified on the <call-presentation-procedure> statement performing the broadcast.

When sending broadcast data, a task uses a broadcast list. A *broadcast list* is a list of displays to which a broadcast message is sent. A broadcast list is defined using environmental information.

A display can be accessed by more than one task at a time.

Implementor Note:

An implementation can restrict normal mode access to a display so that all access to the display is serialised except for broadcast sends.

A display is assignable to a task when the display is not currently assigned to another task, or when the display is assigned to the task that called the current task.

3.2.6 Presentation Procedure

A *presentation procedure* is a named sequence of serial presentation operations defined using a vendor-supplied forms system, standard COBOL or standard C. A presentation procedure is called by a task to send data to or get data from a display. A presentation procedure handles the transfer of data between the task and a display. A presentation procedure executes independently of any transaction started by a task.

There are three types of presentation procedure:

- Send Procedure

  Takes data from the task and displays it on the display.

- Receive Procedure

  Takes data from the display and returns it to the task.

- Transceive Procedure

  Combines the semantics of a send and a receive procedure.

3.2.7 Presentation Group

A *presentation group* is a named set of presentation procedures defined using either a <presentation-group-specification> or a vendor-supplied forms system. The presentation procedures in a presentation group are all executable on each TP system that implements the presentation group. A <presentation-group-specification> defines the interface through which a customer-written presentation procedure can be called.

Customer-written procedures in a presentation group are all written using a single programming language: COBOL or C.

A presentation group defines the scope of context sharing for the execution of presentation procedures. The context that a presentation procedure creates can be used by another presentation procedure and is called the *presentation group context*.

Two presentation procedure executions share the same presentation group context if all of the following conditions are met:

- Both presentation procedures are called by the same task execution.
- Both presentation procedures are in the same presentation group.
- Both executions are on the same TP system.

The presentation group context consists of:

- context maintained by the display, if any
- context maintained by the presentation procedures, if any.

The presentation group context is maintained at least until the end of the task execution. See Section 12.4 on page 260 for further information on presentation context.

Implementor Note:

An implementation can execute two presentation procedures in the same presentation group context if the two presentation procedures are called by the same task execution on the same TP system, even if the presentation procedures are in different presentation groups.

3.3 Task Invocation

A *task invocation* is an operation that results in a task execution. Task invocation can be done in one of two ways:

- task call
- task submit.

An invoked task is called a *server task*. The task group in which the server task is defined is the *server task group*. The TP system on which the server task executes is called the *server TP system*.

3.3.1 Task Call

A program that performs a task call is called a *client program*. The TP system on which the client program executes is called the *client TP system*. The client TP system can be the same TP system as the server TP system or another TP system.

A *task call* is a task invocation in which the client program thread:

- supplies the initial input, if any, using arguments passed on the call
- specifies whether or not the server task executes within the client program's transaction:
  - If the server task executes within the client program's transaction, the server task is a *composable task*.
  - If the server task does not execute within the client program's transaction, the server task is a *non-composable task*.

  Each task definition specifies whether or not the task is composable or non-composable.
- waits for the server task to complete before continuing execution
- receives the final results, if any, of the task execution; this can be either:

Normal Execution	Any output arguments are returned to the client program.
Exception	An exception is returned to the client program and any output arguments are undefined.

For STDL applications, a client program can be:

- Internal to the TP System

  Using a vendor-specific procedure call mechanism.
- External to the TP System

  Using a procedure call mechanism defined by either the TxRPC or DCE RPC protocol.

An internal client program is one of the following:

- Customer-written Client (written using standard COBOL or C)

  A *customer-written client* is a client program running inside a TP system that calls tasks within the same TP system using client stubs generated during the translation of the server <task-group-specification>.
- Client Task (written using STDL)

  An internal *client task* is a task inside a TP system that performs a task call to another task within the same TP system using a vendor-specific procedure call mechanism.

- Menu Client Program (supplied by a vendor)

  A *menu client program* is a client program that runs inside a TP system and calls a task within the same TP system using a vendor-specific procedure call mechanism.

- Task Dequeuer (supplied by a vendor)

  An internal task dequeuer is a client program that runs inside a TP system, dequeues a task from a task queue and calls a task within the same TP system using a vendor-specific procedure call mechanism. (See Section 3.3.2 on page 50 for further information on the task dequeuer.)

- Processing Procedure (written using standard C or COBOL)

  An internal processing procedure is a client program running inside a TP system that performs a task call to another task within the same TP system using client stubs generated during the translation of the server <task-group-specification>.

An external client program is one of the following:

- External Client (written using standard C)

  An *external client* is a client program running outside a TP system that calls tasks within a TP system using the DCE RPC protocol. An external client uses client stubs generated during the translation of the server <task-group-specification>. The client stubs are made available to the external client using an implementation-specific mechanism.

- Client Task (written using STDL)

  An external client task is a task that performs a task call to a task outside the TP system using the TxRPC or DCE RPC protocol.

- Task Dequeuer (supplied by a vendor)

  An external task dequeuer is a client program that dequeues a task from a task queue and calls a task outside the TP system using the TxRPC or DCE RPC protocol.

- Processing Procedure (written using standard C or COBOL)

  An external processing procedure is a client program that calls a task outside the TP system using the TxRPC or DCE RPC protocol. A processing procedure uses client stubs generated during the translation of the server <task-group-specification>.

See Section 13.5 on page 267 for further information on client stub procedures.

A client program can call a task that is on the same TP system or on another TP system. When a client program calls a task, the client TP system must determine the server TP system to which to direct the task call.

When a client program calls a task, there are three cases to consider in determining the server TP system:

1. The client program specifies a server task in the same task group on the same TP system.

   This is possible only for client programs that are tasks. This is done by omitting the task group name and destination name on the call task or submit task statement.

2. The client program does not specify any explicit addressing information.

   For client programs that are tasks this is done by omitting the task group name and destination name on the call task or submit task statement.

For a customer-written client program, external client program or processing procedure, this is the only case. The client TP system uses information associated with the server task group specification to determine the server TP system. An implementation can choose to use either the server task group name or the server task group UUID.

3. The client program specifies explicit addressing information.

   This is possible only for client programs that are tasks. This is done by specifying the destination name and/or task group name on the call task or submit task statement.

When determining the server TP system for cases 2. and 3. above, the client TP system uses environmental information associated with a destination. (This environmental information may be supplied using a name resolution service; see Section 17.2.9 on page 306 for more information.) Each destination includes a list of task groups. Each entry in the task group list contains the following information:

1. task group identification (this can be either the task group name or task group UUID as specified in the <task-group-specification>)
2. version number (as specified in the <task-group-specification>)
3. RTI-AP-Title (the information required to send calls to the server TP system for this task group)
4. security mechanism type (specifies which security mechanism the client TP system should use in making calls to the server TP system.

When determining the server TP system for cases 2. and 3. above, the client TP system first determines which destination to use. When the client program does not explicitly specify addressing information, the implementation on the client TP system can choose to use either a destination with the same name as the server task group or a default destination. When the client program explicitly specifies the server TP system, the client TP system uses the destination with the name specified explicitly.

After the destination has been determined, a task group entry within the destination's task group list is chosen. The client TP system uses the following information in picking an entry from the task group list:

- The entry's task group identification matches the server task group identification — an implementation can choose to use either the task group name or the task group UUID as the task group identification.
- The entry's task group version must be compatible with the server task group version — the major version must match and the minor version in the task group entry must be greater than or equal to the minor version of the server task group specification used by the client program.
- In cases where more than one entry satisfies the above criteria, the client TP system uses an implementation-specific mechanism to select among such entries.

When the client TP system sends a task call to the server TP system, the client TP system sends the task group UUID and version number to the server TP system. The UUID ensures that the client and server TP system are using the same task group specification. The version number ensures that the versions of the task group specification used by the client and server TP systems are compatible.

The server TP system validates that the UUID and the version number on the task call message are valid. The UUID must match a UUID of a task group supported by the server TP system. The version number must be compatible with a task group version supported by the TP system. If

either of these two checks fail, the server TP system returns an ENV-INVOCATION-FAULT exception class to the client TP system. Otherwise, the server TP system executes the server task. See Chapter 9 on page 157 and the referenced X/Open DCE RPC specification for further information on UUIDs and version numbers.

3.3.2 Task Submit

A *task submit* is a task invocation in which the submitter task requests that this or another TP system performs the task call at some future time. A task submit can be performed only by a task. A task that performs a task submit is called a *submitter task.* The TP system on which the submitter task executes is called the *submitter TP system.*

A task submit is a task invocation in which the submitter task thread:

- supplies the initial input data, if any, for the server task
- specifies whether or not the task submit is done within the submitter task's transaction:
  — If done within the submitter task's transaction, then the task submission request is available for forwarding or calling when and if the submitter task's transaction commits.
  — If not done within the submitter task's transaction, then the task submission request is available for forwarding or calling when the task submit is completed.
- waits for the task submit to complete before continuing execution
- does not wait for the server task to complete
- does not receive any output data or exceptions from the server task
- optionally gets a run-time identifier that uniquely identifies the submitted task.

The abstract component of the TP system that performs the task call based on information supplied by the submitter task is called the *task dequeuer.* Because the task dequeuer is a client program, the TP system on which the task dequeuer executes is a client TP system.

Figure 3-2 on page 51 depicts the actions that occur during a task submission.

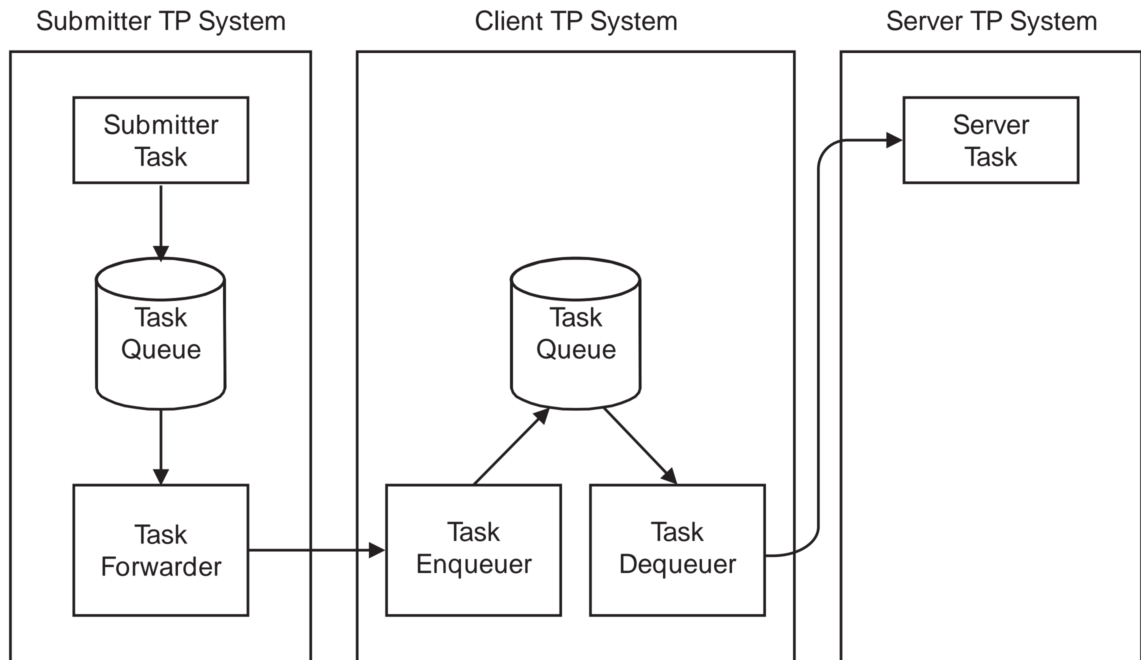


Figure 3-2 Task Submission, Forwarding and Calling

A task can submit a task to be executed on a single TP system or on multiple TP systems. If a task submits a task to be executed on a single TP system, the task submit can explicitly specify a destination name. If a destination name is not explicitly specified, the task group name is implicitly used as the destination name.

The submitter TP system uses the destination named on the task submit, either explicitly or implicitly, to map a task submission request to a client and a server TP system. If a destination name is not explicitly specified, the server task group name or the server task group UUID is implicitly used to determine the server TP system. The destination is also used to obtain the information each task submission request must contain in order for the client TP system to convert the task submission request into a task call. The submitter TP system stores the task submission request in a local task queue.

If a task submits a task to be executed on multiple TP systems, the task submit must explicitly specify a distribution list name. The submitter TP system uses the distribution list named on the task submit to map to a set of one or more destination names. The submitter TP system uses each destination in the distribution list to map a task submission request to a client and a server TP system, and to obtain the information each task submission request must contain in order for the client TP system to convert the task submission request into a task call. The submitter TP system stores a task submission request in a local task queue for each destination in the distribution list.

A *task queue* is a repository on a TP system in which task submission requests are durably stored. A submission request entry consists of:

- information needed to obtain the client TP system name and whether or not to use a transaction when forwarding the submission request entry
- server task name

- task group name for the server task
- UUID, version number and information required to direct the task call to the server TP system
- composability flag; whether or not the server task is composable
- task display
- language
- trigger information; activation information which is used by the client TP system to determine when to convert the task submission request into a task call
- repeat information; activation information which is used by the client TP system to determine the frequency at which to replicate the conversion of the task submission request into a task call
- task submission information:
  - — submission time in universal time (UT)
  - — submission TP system name
  - — submission task group
  - — submission task
- arguments
- each submission request entry in a task queue on a submitter TP system has a unique run-time identifier; the run-time identifier can be returned to the submitter task and can be used on the submitter TP system to remove the submission request entry from the task queue. This is called a cancel submit. A *cancel submit* can be performed within the submitter task's transaction or independently of the submitter task's transaction. If the submission request entry has been forwarded to another TP system or the server task has been started, the cancel submit returns an exception.

If the submitter TP system and client TP system are different TP systems, the task submission request is forwarded to the client TP system.

A TP system can be a client TP system, a server TP system, and a submitter TP system, depending on the method of task invocation and the destination of the submitted task.

A client TP system needs no information about the server task other than that supplied in the task submission request.

3.3.2.1 Actions on the Submitter TP System

For a submitter TP system to support task submission it must:

1. Store a task submission request in a task queue for each destination in the distribution list. The task queue used to store a task submission request must be on the local submitter TP system.
2. Determine which client TP system will convert the task submission request into a task call. Environmental information is used to map the TP system address of the submitted task to a client TP system.
3. Remove the task submission request from the local task queue and either:
   - — forward it to the appropriate client TP system

Whether the forwarding operation is done within a transaction depends on the transactional forwarding attribute of the destination, which is set using environmental information. The transaction within which a task is forwarded, if any, is outside of any transaction within which the initial submission is performed. When a task submission request is forwarded within a transaction, the submitter TP system deletes the submission request entry from the task queue in the same transaction in which the submission request is forwarded.

or:

— perform a task call if the submitter TP system is the client TP system.

4. Process forwarding exceptions. If the exception is ENV-INVOCATION-ERROR, TXN-FAILURE-ERROR, ENV-EXECUTION-ERROR or TXN-TIMEOUT-ERROR, the submitter TP system retries the forwarding operation based on environmental information. If the exception is anything else, or if the task queue's retry limit is exceeded for the entry, the submitter TP system audits the exception and places the task submission request on hold. If the task submission request is put on hold, an operator must either release the submission request or delete it.

See Chapter 18 on page 319 for more information.

3.3.2.2 *Actions on the Client TP System*

For a client TP system to support task submission it must:

1. Accept task submission requests forwarded from submitter TP systems and store them in a task queue on the local TP system.

   When a task submission request is forwarded within a transaction, the client TP system must store the entry in the task queue in the same transaction in which the submission request is forwarded.

2. Process the triggers associated with the task submission request and, when appropriate, call the task based on information in the submission request.

3. Call the task using information in the task submission request. This includes whether or not to call the task within a transaction started on the client TP system. This transaction is independent of any transaction used to originally submit the task or to forward the task from a submitter TP system.

4. If the task call completes successfully, delete the task submission request from the task queue:

   — within the same transaction in which a composable task executes

   — independently from the execution of a non-composable task, after the non-composable task is completed.

5. Process call exceptions. If the exception is ENV-INVOCATION-ERROR, ENV-EXECUTION-ERROR, TXN-FAILURE-ERROR, TXN-TIMEOUT-ERROR, TXN-INCOMPLETE-ERROR or REQUEST-TIMEOUT-ERROR, the TP system retries the task call based on environmental information. If the exception is anything else, or if the task queue's retry limit is exceeded for the entry, the client TP system audits the exception and places the task submission request on hold. If the task submission request is put on hold, an operator must either release the submission request or delete it.

See Chapter 18 on page 319 for more information.

3.3.3 Dialogues

A *dialogue* is a connection between two TP systems over which messages can be sent. A TP system sends a task invocation message to another TP system over a dialogue. Task invocation messages are task call messages and task submit messages. A dialogue is used for either task call messages or task submit messages, but not both. The system that originates the dialogue is the *dialogue client system*. The system that accepts the dialogue is the *dialogue server system*.

For a task call message, the dialogue client system can be either a client TP system or an external client. The server TP system is the dialogue server system. Each task call dialogue handles task calls for one task group in the server TP system.

For a task submit message, the submitter TP system is the dialogue client system and the client TP system is the dialogue server system. A task submit dialogue can be used for any task submission from the submitter TP system to the client TP system. In this section, task submissions are described as calls from the submitter TP system to a special task on the client TP system.

A *transactional dialogue* is a dialogue over which a transaction spanning two TP systems is coordinated. Calls to composable tasks or task forwarding within a transaction can be performed over a transactional dialogue. A dialogue server system can optionally accept multiple calls over a transactional dialogue. A transactional dialogue always terminates at the end of the transaction.

A *non-transactional dialogue* is a dialogue over which no transaction coordination is done. Calls to non-composable tasks and non-transactional task forwarding use a non-transactional dialogue.

If a dialogue server system for a non-transactional dialogue only accepts a single call over a dialogue, then the dialogue is terminated after each call. Otherwise, the dialogue is terminated by mutual agreement between the dialogue client system and the dialogue server system. The termination of a dialogue is summarised in Table 3-3.

	Transactional Dialogue	**Non-transactional Dialogue**
Dialogue server system accepts one call per dialogue.	After one transaction.	After one call.
Dialogue server system accepts multiple calls per dialogue.	After one transaction.	Determined by dialogue client system and dialogue server system.

Table 3-3 Dialogue Termination

Implementor Note:

An implementation is not required to accept multiple task calls on dialogues.

When the dialogue is used for task submissions, the following rules apply:

- For a non-transactional dialogue, the submitter TP system's removal of the forwarded submitted task request from the task queue must be performed after the completion of the call. The client TP system's storing of the forwarded submitted task request in a task queue must be performed before the completion of the call.

- For a transactional dialogue, the removal of forwarded submitted task requests from the submitted TP system's task queue and the storing of those task requests on the client TP system's task queue are performed within the transaction in which the forwarding operations take place.

Transactional dialogues used for task calls are part of the task group context at the client TP system and at the server TP system:

- Any two task calls made sequentially from tasks executing in the same task group context at the client TP system to the same server TP system for tasks in the same task group must use the same dialogue.
- If two task calls are made sequentially on the same transactional dialogue, then the server TP system must execute those two tasks in the same task group context.

Whether or not a non-transactional dialogue is part of the task group context is implementation-specific.

Implementor Notes:

1. If a dialogue server system supports only one call per dialogue, the client TP system receives an exception if it tries to call another task in the same task group in the same transaction on the same TP system.
2. An implementation can execute two tasks in the same task group in different task group contexts if the task calls arrive on different dialogues, even when the two tasks are executing in the same transaction on the same TP system.
3. If an implementation supports only dialogue servers that accepts one call per dialogue, then the implementation can omit transactional dialogue context as part of transactional task group context. (See Section 3.2.2 on page 42 for a description of task group context.)
4. The client TP system can allow a non-transactional dialogue to be used by any task as long as the dialogue setup is appropriate for the task. See Chapter 18 on page 319 for more information.

   The client TP system can terminate the dialogue at any time after the task call is completed.
5. The server TP system can create separate task group contexts for each incoming task call on a non-transactional dialogue or it can use one task group context for the life of the dialogue.

   The server TP system can request that the client TP system terminate the dialogue at any time after the task call is completed.

3.4 Security

There are three major concepts related to TP system security:

- Principal

  A *principal* is any active entity that can be held accountable for its actions and thus whose identity must be assured.

- Authentication

  — User Authentication

  User authentication is the local process of verifying the identity of users. Only after a user has been authenticated is a user allowed to perform any actions provided by the TP system.

  — Remote Authentication

  Remote authentication is the process in which a dialogue client system presents the authentication information for a principal to a dialogue server system so that the dialogue server system can verify the identity of the principal. The principal is authenticated at the dialogue client system.

- Authorization

  Authorization is the process by which a principal gains access to the execution of a task or is allowed to submit a task to a remote client TP system. Authorization uses access control defined for the task and for the TP system task queue.

3.4.1 Principal

A TP system must support at least the following principals:

- User

  The user is the principal for any task invocation from a menu client or from a customer-written client program executed by a user.

- TP System

  The TP system is the principal for a remote task invocation (task call and task submit) from one TP system to another.

  Implementor Note:

  > An implementation can use another entity logically contained within the TP system as a principal.

The principal for any other type of task invocation within the TP system is implementation-defined.

An external client is a principal when an external client calls a task on a TP system.

3.4.2 User Authentication

Before executing a menu client program or a customer-written client program, users must identify themselves to the TP system or the system on which the TP system is executing. A user is identified by the user name specified in the user profile. Implementation-specific information contained within the user profile (such as a password) verifies the user's identity. Upon verification of the user's identity, the TP system associates a user profile with that user.

3.4.3 Remote Authentication

Whenever a dialogue client system sends a task invocation message (either a task call or a task submit) to a dialogue server system, the authentication information of the principal at the dialogue client system must be sent to the dialogue server system. The dialogue server system uses the information to verify the principal's authentication.

The following authentication information is sent from the dialogue client system to the dialogue server system:

- Principal Name

  A text string that identifies the principal, as specified in the referenced X/Open TxRPC specification.

  For a TP system, the dialogue client system uses the TP system environmental information to determine which principal name to use.

- Authentication Mechanism Type

  Determines the type of authentication data sent for the principal.

  The dialogue client system uses the destination environmental information to determine which authentication mechanism to use. For a task call, the dialogue client system uses the call authentication mechanism type specified for the destination task group. For a task forward, the dialogue client system uses the forwarding authentication mechanism type.

  Two authentication mechanism types are specified in Section 8.3, DC-ASE Services, of the referenced X/Open TxRPC specification:

Default	The default authentication mechanism is password-based.
Customer-written	A customer can supply an authentication mechanism.

- Authentication Data

  Information that the dialogue server system uses to verify the authentication of the principal at the dialogue client system.

  The dialogue client system uses the authentication mechanism and the TP system environmental information to determine the authentication data.

  What the authentication data contains depends on the authentication type:

For default	A text string password for the principal, as specified in the referenced X/Open TxRPC specification.
For customer-written	An octet array of data for the principal used by the customer-supplied authentication mechanism, as specified in the referenced X/Open TxRPC specification.

When the dialogue server system receives a task invocation message, the dialogue server system uses the principal's name to locate the incoming authentication environmental information for that principal. If no incoming authentication environmental information is defined for the

principal, the dialogue server system returns an ENV-INVOCATION-FAULT class exception to the dialogue client system. If incoming authentication environmental information is defined for the principal, the authentication information passed from the dialogue client system is used by the dialogue server system to verify the authentication of the principal. The authentication mechanism used is determined by the authentication mechanism type passed by the dialogue client system. If the dialogue server system's verification of the authorization of the principal fails, the dialogue server system returns an ENV-INVOCATION-FAULT class exception to the dialogue client system.

If the dialogue server TP system can verify the authentication of the principal, the dialogue server system checks the authorization for the task (for a task call) or the TP system task queue (for a task forward). If the authentication for the task fails, the dialogue server system returns an ENV-INVOCATION-FAULT class exception to the dialogue client system.

An incoming authentication entity must be defined at the dialogue server system for each principal that can send a task invocation message to that dialogue server system.

3.4.4 Authorization

A TP system must support the definition of an access control that limits the access of a principal to a task execution and to the TP system task queue. Environmental information for a task and for the TP system task queue defines an implementation-specific access control attribute. The access control attribute is used by an implementation-specific enforcement mechanism that grants or denies access to the task or the TP system task queue.

3.5 Environment

An implementation translates source files within the translation environment and executes TP system entities within the execution environment.

3.5.1 Translation Environment

The *translation environment* is the environment within which STDL definitions and specifications and associated standard programming language sources are processed into executable entities.

The translation environment contains the information necessary to accomplish the translation of STDL definitions and specifications and associated programming language source files into executable entities.

3.5.2 Execution Environment

The *execution environment* is the environment within which the executable entities produced within the translation environment are executed.

The execution environment contains the information necessary to control and regulate the execution entities at run time.

3.5.3 Environmental Information

Environmental information is information used within the translation and execution environments that is not specified directly using STDL definitions and specifications or associated standard programming language source files. Environmental information controls and regulates the behaviour of TP system entity functionality.

Environmental information is specified by assigning attributes to TP system entities and by performing operations on the entities using a set of abstract services. The mechanism used to specify environmental information and to implement the defined functionality is implementation-specific.

3.6 Record Queuing

Record queuing allows records to be enqueued, read and dequeued. This section describes the record queuing model.

3.6.1 Ordering

A *record queue* is a sequential list of records. The queue is ordered according to the FIFO ordering policy. Records are enqueued at the end of the list, called the *tail* of the queue. Records can be read (without removal) or removed from any position within the queue. Records are typically removed from the beginning of the list, called the *head* of the queue.

3.6.2 Records and Positions

Conceptually a queue entry consists of a queue record and an associated position. A logical ''beginning-of-queue'' marker, called the head of the queue, and a logical ''end-of-queue'' marker, called the tail, each with an associated position (see Figure 3-3) bracket the queue. A queue *cursor* either points to a position in the queue or is undefined. The cursor is used only for reading records from the queue. There is only one cursor for each transaction in a task group.

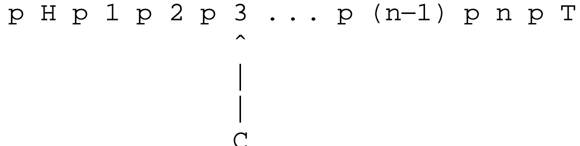


Figure 3-3 Record Queue

p A position.

n A record.

H The head of the queue.

T The tail of the queue.

C The queue cursor.

Each record on the queue is tagged with an *identifier*. The identifier is unique with respect to the queue. It is allocated when the record is first inserted onto the queue.

3.6.3 Context

The scope of the cursor is limited to a task group and a transaction. The queue context consists of:

- Cursor.

3.6.4 Queue Operations

There are three basic operations that can be performed on a record queue: enqueue, read queue and dequeue.

3.6.4.1 Enqueue Operation

This operation inserts a record on the queue. The record is always inserted at the tail of queue, independent of the queue cursor. The operation does not affect the queue cursor. The queue record is tagged with an identifier which is unique with respect to the queue.

3.6.4.2 Dequeue Operation

The dequeue operation either:

- removes the first available record (as determined by the isolation level of record queue access) from the head of the queue
- removes a specific record from the queue, identified using the unique record identifier
- removes the first available record (as determined by the isolation level of record queue access) from the head of the queue whose record key and value match the qualifier used on the dequeue operation.

If a dequeue operation removes the record pointed to by the queue cursor, then the cursor is set to point to the position immediately preceding the deleted record for the queue. Otherwise, the dequeue operation does not affect the queue cursor.

3.6.4.3 Read Queue Operation

The read queue operation allows a queued record to be read without removing the record from the queue. Through a series of read queue operations, a queue can be traversed. The queue cursor is used for this purpose.

All read queue operations follow the same general rules. Prior to each operation, the queue cursor directive, which must always be explicitly provided, is evaluated and the queue cursor adjusted accordingly. After adjustment of the cursor, the record associated with the position pointed to by the cursor is read.

In particular, for a queue it is required that:

- For the first read queue operation on a queue, the queue cursor must first be established, or anchored.

  This is done using an explicit queue cursor directive: <read-queue-key-first>. A read queue operation using a specific identifier (<read-queue-id>) cannot be used to anchor the cursor.
- Prior to each read queue operation, the directives are evaluated, and the queue cursor is adjusted. If the directive <read-queue-key-first> is used, the cursor is set to point to the first available record on the queue. If the directive <read-queue-key-next> is used, the cursor is set to point to the next available position for the queue. The cursor is undefined after a record queue operation using a specific identifier (<read-queue-id>).
- The <read-queue-key-next> directive must always follow a <read-queue-key-first> or a <read-queue-key-next> directive on the same queue. Any read queue or dequeue operation for which no record is found on the queue, queue is empty, or no record is found with a matching ID or key (if specified), results in a non-transaction exception. The queue cursor is undefined.

3.7 Exceptions

An *exception* is an event detected by the TP system that changes the normal flow of execution within a task. Exceptions are caused by errors or other conditions encountered during the execution of an operation for a task or procedure. When an exception occurs, the output of the operation that caused the exception is undefined.

3.7.1 Types of Exceptions

Exception type is a categorisation of exception conditions based on whether or not the exception prevents the current transaction from committing. There are two types of exception:

- Non-transaction Exception

  A *non-transaction exception* is an exception that does not prevent the current transaction from committing. A non-transaction exception causes the execution of an exception handler, if there is an applicable exception handler.

- Transaction Exception

  A *transaction exception* is an exception that prevents the current transaction from committing. A transaction exception causes the current transaction to be rolled back. There are three subtypes of transaction exceptions based on the actions of the TP system:

  — Transient Transaction Exception

    A *transient transaction exception* causes the transaction to be rolled back and then causes the TP system to retry the transaction.

  — Permanent Transaction Exception

    A *permanent transaction exception* causes the transaction to be rolled back and then causes the TP system to execute an exception handler, if there is an applicable exception handler.

  — Fatal Transaction Exception

    A *fatal transaction exception* causes the transaction to be rolled back and then causes the TP system to terminate the execution of the task. The exception is reported to the client program.

3.7.2 Exception Classes, Sources, Codes and Level

The following information is provided for each exception:

- Exception Class

  An *exception class* is a classification of exception conditions based on the type of recovery action that can be taken. Exception classes are defined in Appendix C on page 489. Each exception class is initially raised with a specific exception type. As explained later, this type can change depending on how the exception is reported.

  The suffix on the exception class name is used to further classify exception classes by the ability to retry the operation that generated the exception class. There are two exception class suffixes:

  ERROR — Indicates that the exception is a soft error and the operation may be successful if retried, at some future time, without modifying the STDL source, standard program language source, or environmental information. In some cases, data has to be modified.

FAULT Indicates that the exception is a hard error and modifying the STDL source, standard programming language source, or environmental information is required before the operation can be successfully retried.

The exception class suffix provides additional information about the type of recovery action that can be taken by the application designer.

Exception classes allow for portable exception handling.

- Exception Source

  An *exception source* is a classification of exception conditions based on whether the exception was generated by the application or the TP system. The application can be the task, the processing procedure, or a customer-written presentation procedure.

- Exception Code and Exception Code Group

  An *exception code* is a detailed classification of an exception condition. For application exceptions, the exception code is defined in a <message-group-definition> and can be made portable. For TP system exceptions, the exception code is implementation-specific and is not portable.

  An *exception code group* is the identification of the group in which the exception code is defined. Exception code groups are used for exceptions raised by the system and can be used for exceptions raised by the application if the <message-group-definition> specifies a UUID.

  Message groups with UUIDs are the exception code groups for exceptions raised by the application. If no UUID is specified in the <message-group-definition>, the programmer is responsible for uniquely defining the exception codes.

  An implementation must define one or more system exception UUIDs. These UUIDs are system-generated in a manner that guarantees uniqueness.

  See the referenced X/Open DCE RPC specification for more information on generating UUIDs.

- Exception Level

  An *exception level* is a classification of exception conditions based on whether the exception was generated by the execution of the current task or the execution of a called task, processing procedure, or presentation procedure, which propagated the exception to the current task, processing procedure, customer-written client program, or external client..

  An exception level is defined as *current* when the exception is generated by the current task execution and as *propagated* when the exception is generated by the execution of a task, processing procedure, or presentation procedure called by the current task, processing procedure, customer-written client program, or external client. An exception level is defined as current also when an exception is encountered during the invocation of a task, processing procedure, or presentation procedure.

  A processing procedure, customer-written client program, or external client cannot distinguish between a current exception and a propagated exception.

- Exception Location

  The *exception location* is a pair of text fields that describe where the exception occurred. The exception location text fields capture the names of the exception procedure and exception procedure group in which the exception occurred. The exception procedure name can be a task name, processing procedure name, presentation procedure name, or arbitrary string. The exception procedure group name can be a task group name, processing group name,

presentation group name, or an arbitrary string.

When an exception is raised in a task, the TP system sets the exception procedure name to the task name and the exception procedure group name to the task group name.

When an exception is raised in a processing or presentation procedure, the procedure can choose to set the exception procedure name and exception procedure group name. When an exception is raised by the TP system, the TP system sets the exception procedure name and the exception procedure group name to the name of the procedure executing at the time the exception was raised and the name of the processing or presentation group through which the procedure was called.

3.7.3 How Exceptions Are Generated

Exceptions are generated by:

- The TP system:
  - — if any error is detected while performing an operation within a task
  - — if an error occurs that prevents the continued execution of a processing or presentation procedure
  - — optionally, if an error occurs during the execution of file I/O or SQL statements in processing procedures that prevents the transaction from committing
  - — if an error occurs during the execution of a presentation procedure provided by the TP system (within a vendor-supplied forms system) that prevents data from being returned to the task.
- The application:
  - — tasks; by executing <raise-exception>, <reraise-exception> or <restart-transaction>
  - — processing and presentation procedures; by setting values in STDL-defined external variables and returning them to the TP system (see Section 14.2 on page 270 for how COBOL procedures raise exceptions, and Section 15.2 on page 280 for how C procedures raise exceptions).

3.7.4 Exception Handlers

An *exception handler* is the part of the task that is executed only when an exception occurs. Each exception handler has a scope — that part of the task for which the exception handler is executed in response to an exception. The scope of an exception handler can include parts of a task for which no exception handler is specified.

The scope of an exception handler differs for non-transaction and transaction exceptions. A *non-transaction exception handler* handles only non-transaction exceptions. A non-transaction exception handler can be associated with a <statement-block> that is within the <transaction-block>, or within a <concurrent-block>, or a <composable-task>, which are always within a <transaction-block>.

A *transaction exception handler* handles only transaction exceptions. A transaction exception handler is associated with a <transaction-block> or with a <statement-block> that is outside of a <transaction-block>.

Before executing an exception handler, the TP system sets the exception class, source, code, group, level and location in the EXCEPTION-INFO-WORKSPACE system workspace. This provides the exception handler with access to this information. See Section 3.8.2 on page 69 for further information on system workspaces.

The exception handler itself can raise an exception, if necessary. The exception handler can reraise an exception. An exception handler is defined using an <exception-handler>.

Implementor Note:

If the TP system encounters a system-generated exception during the execution of a transaction exception handler, an implementation can treat the exception as a fatal transaction exception.

3.7.5 TP System Actions when an Exception is Generated

The TP system performs the following actions when an exception is generated:

1. maps the implementation-specific exception code to the appropriate exception class, for an exception raised by the TP system
2. determines the type of the exception using the default exception type (see Appendix C on page 489), except for exceptions raised as follows:
   — during the execution of a transactional <call-presentation-procedure> with a <call-presentation-receive-list>; always permanent transaction

   — upon the execution of a <raise-exception> with rollback; always permanent transaction

   — upon execution of a <restart-transaction>; always transient transaction
3. examines the application-defined exception code when a top-level processing or presentation procedure exits or a task executes a <raise-exception> or <restart-transaction>; if the exception code is set to a non-zero value, the TP system maps the exception code to the appropriate exception class, as defined in the message group
4. generates an AP-EXECUTION-FAULT exception, if it is unable to map the exception code to class due to either of the following conditions:
   — The exception code is not defined in the message group for an application-generated exception.

   — The exception class is not one of the standard, predefined exception classes.
5. increments the fault count attribute for each execution of a task or top-level processing procedure that results in an exception contained within the exception list
6. disables the task or procedure, if the fault count attribute exceeds the maximum fault limit attribute.

The remaining actions of the TP system depend on where the exception was generated, to what the exception was passed, and the type of exception.

3.7.5.1 Exception Handling in Customer-written and External Clients

If an exception is passed to a customer-written client or an external client, the exception is passed to the customer-written client or external client by setting the STDL-defined external exception variables as described in Section 13.1 on page 263 and Section 13.4 on page 266.

3.7.5.2 *Exception Handling in Processing Procedures*

If an exception is generated in a processing procedure or passed to the processing procedure, the TP system:

- Performs the following if the top-level processing procedure returns to the task with the STDL-defined external exception variables set:
  - Returns the exception with the same type to the calling task if the exception has a type of non-transaction, transient transaction or permanent transaction exception.
  - Returns the exception with a permanent transaction exception type to the calling task if the exception has a type of fatal transaction.

  Implementor Note:

  > An implementation can handle a fatal transaction exception raised by a top-level processing procedure as if it were a fatal exception raised by the task that called the top-level processing procedure.

- Performs the following if the exception was raised during a call to a task or a called task returned an exception to the processing procedure:
  - Returns the exception to the processing procedure with the exception information set in STDL-defined external exception variables as described in Section 13.3 on page 265 if the exception type was non-transaction.
  - Terminates the top-level processing procedure called by the task and returns the exception to the calling task if the exception type was transient transaction or permanent transaction.
- Performs the following if the TP system detects an exception during the execution of the processing procedure other than during a call to a task:
  - Terminates the top-level processing procedure and returns the exception to the calling task.

3.7.5.3 *Exception Handling in Presentation Procedures*

If an exception is generated in a presentation procedure, the TP system returns a non-transaction exception to the calling task, if any exception was generated in a presentation procedure for a <call-presentation-procedure> with a non-transactional attribute and a <call-presentation-send-list>, with a non-transactional attribute and a <call-presentation-receive-list>, or with a <call-presentation-transceive-list>. Returns a permanent transaction exception to the calling task, if any exception was generated in a presentation procedure for a transactional <call-presentation-procedure> with a <call-presentation-receive-list>.

3.7.5.4 *Non-transaction Exception Handling in Tasks*

When a non-transaction exception is generated in a task or passed to a task, the TP system:

- Performs the following, if the task that generates the non-transaction exception is a composable task:
  - Executes the non-transaction exception handler, if there is an applicable non-transaction exception handler.
  - Terminates the task and returns the non-transaction exception to the client program, if there is no applicable non-transaction exception handler.

- Performs the following, if the task that generates the non-transaction exception is a non-composable task:
  - — Executes the non-transaction exception handler, if there is an applicable non-transaction exception handler.
  - — Changes the type of exception from non-transaction exception to permanent transaction exception, if there is no applicable non-transaction exception handler.

3.7.5.5 Transient Transaction Exception Handling in Tasks

Transient transaction exception handling depends on whether or not the task is restartable. A *restartable task* is a task that allows transient transaction exceptions. A *non-restartable task* is a task that treats a transient transaction exception as a permanent transaction exception. The restartable attribute is defined in the <task-definition>. When a transient transaction exception is generated in a non-restartable task or a transient transaction exception is passed to a non-restartable task, the TP system changes the type of exception from transient transaction exception to permanent transaction exception, and then performs the actions associated with a permanent transaction exception in a task. The exception class, code and source are not changed. When a transient transaction exception is generated in a restartable task or passed to a restartable task, the TP system:

- Terminates the task and returns the transient transaction exception to the client program, if the task that raises the exception is a composable task.
- Performs the following, if the task that generates the transient transaction exception is a non-composable task:
  - — Rolls back the current transaction.
  - — Changes the type of exception from transient transaction exception to permanent transaction exception, if a <call-presentation-procedure> statement with a non-transactional attribute and a <call-presentation-send-list> or a <call-presentation-receive-list> was performed during the execution of the transaction, and then performs the actions associated with a permanent transaction exception. The exception class, code and source are not changed.
  - — Changes the type of exception from transient transaction exception to permanent transaction exception, if the number of retries exceeded the restart limit attribute for this transaction, and then performs the actions associated with a permanent transaction exception. The exception class, code and source are not changed.
  - — Increments the retry count and re-executes the transaction that generated the transient transaction exception, unless a <call-presentation-procedure> statement with a non-transactional attribute and a <call-presentation-send-list> or a <call-presentation-receive-list> was performed or the restart limit attribute was exceeded.

3.7.5.6 Permanent Transaction Exception Handling in Tasks

When a permanent transaction exception is generated in a task or passed to a task, the TP system:

- Terminates the task and returns the permanent transaction exception to the client program, if the task that generates the exception is a composable task.
- Performs the following if the task that generated the permanent transaction exception was a non-composable task:

1. Rolls back the current transaction.
2. Executes the transaction exception handler, if there is an applicable transaction exception handler.
3. Terminates the task and returns the non-transaction exception to the client program, if there is no applicable transaction exception handler.

Implementor Note:

If an implementation cannot process an exception as a permanent transaction exception, the TP system can treat the exception as a fatal transaction exception.

3.7.5.7 Fatal Transaction Exception Handling in Tasks

When a fatal transaction exception is generated in the task, the TP system:

- Terminates the task and returns a permanent transaction exception to the client program, if the task that generates the exception is a composable task.
- Performs the following, if the task that generates the fatal transaction exception is a non-composable task:
  1. Rolls back the current transaction.
  2. Terminates the task and returns the non-transaction exception to the client program.

3.8 Resources

A *resource* is an entity in which data is stored or retrieved by a task or processing procedure execution. The following resources are provided by the TP system.

A transaction can include both read and write operations to a resource in any order when a resource can be accessed for both read and write.

3.8.1 Private Workspace

A *private workspace* is a named unit of storage containing a set of values that:

- is a non-durable resource
- can be either transactional or non-transactional, as specified in the <workspace-definition>
- can be accessed for read and write from a single task execution only; each task execution accesses its own private workspace.

A *system workspace* is a private workspace used by the TP system for storing TP system status information.

3.8.2 Shared Workspace

A *shared workspace* is a named unit of storage containing a set of values that:

- is a non-durable resource
- is a transactional resource
- can be accessed for read and write by concurrently executing transactions within the same task group
- supports isolation level 2.

Implementor Note:

> An implementation can support isolation level 2 with serial access but must also then support isolation level 1 for better performance. An implementation that supports isolation level 2 and high concurrency is not required to support isolation level 1. See also the Implementor Note in Section 10.2.3 on page 177.

3.8.3 SQL Database

An *SQL database* is a collection of data items that are accessed using Structured Query Language (SQL). SQL is a data definition and manipulation language used to access relational databases. The properties of the SQL database are defined as part of standard SQL.

An SQL database accessed within the TP system:

- is accessed for read and write by a processing procedure
- is a durable resource
- is a transactional resource
- must support isolation level 3.

Implementor Note:

An implementation can support isolation level 3 with serial access but also must then support isolation level 2 for better performance. An implementation that supports isolation level 3 and high concurrency is not required to support isolation level 2.

3.8.4 Indexed File

An *indexed file* is a collection of records that can be accessed either dynamically or sequentially. Each record within the indexed file is uniquely identified by a key. The semantics of indexed files are defined as part of standard COBOL and standard ISAM.

An indexed file accessed within the TP system:

- is accessed by a processing procedure written using COBOL or C
- is a durable resource
- can be either a transactional or non-transactional resource
- can be accessed within a transaction by only a single file descriptor
- if transactional, can be accessed for read and write by concurrently executing transactions within a TP system
- if non-transactional, can be accessed for read by concurrently executing transactions within a TP system
- if non-transactional, can be accessed serially for write; only one executing transaction at a time can access the non-transactional indexed file when that transaction accesses the file for write
- if transactional, must support isolation level 3.

Implementor Note:

An implementation can support isolation level 3 with serial access but also must then support isolation level 2 for better performance. An implementation that supports isolation level 3 and high concurrency is not required to support isolation level 2.

3.8.5 Relative File

A *relative file* is a collection of records that can be accessed in either random or sequential order. Each record within the relative file is uniquely identified by an integer value greater than 0. Relative files are defined as part of standard COBOL.

A relative file accessed within the TP system:

- is accessed by a processing procedure written using COBOL
- is a durable resource
- can be either a transactional or non-transactional resource
- can be accessed within a transaction by only a single file descriptor
- if transactional, can be accessed for read and write by concurrently executing transactions within a TP system

- if non-transactional, can be accessed for read by concurrently executing transactions within a TP system
- if non-transactional, can be accessed serially for write; only one executing transaction at a time can access the non-transactional indexed file when that transaction accesses the file for write
- if transactional, must support isolation level 3.

Implementor Note:

An implementation can support isolation level 3 with serial access but also must then support isolation level 2 for better performance. An implementation that supports isolation level 3 and high concurrency is not required to support isolation level 2.

3.8.6 Sequential File

A *sequential file* is a collection of records that can be accessed in sequence. Sequential files are defined as part of standard COBOL.

A sequential file accessed within the TP system:

- is accessed by processing procedures written using COBOL
- is a durable resource
- can be either a transactional or non-transactional resource
- can be accessed for read access by concurrently executing transactions within a TP system
- can be accessed serially for write access; only one executing transaction at a time can access the sequential file when that transaction accesses the file for write.

Sequential file support is only required for disks.

These rules do not apply to a non-transactional sequential file opened by a COBOL processing procedure.

3.8.7 Stream File

A *stream file* is a stream of addressable bytes. A stream file can be accessed using C programs only.

A stream file accessed within the TP system:

- is accessed by processing procedures written using C
- is a durable resource
- is a non-transactional resource
- can be accessed for read access by concurrently executing transactions within a TP system
- can be accessed serially for write access; only one executing transaction at a time can access the sequential file when that transaction accesses the file for write.

Stream file support is only required for disks.

These rules do not apply to a stream file opened by a C processing procedure.

3.8.8 Transactional Send and Receive

Transactional send data accessed within a TP system:

- is accessed by a task that provides data for the transactional send
- is a durable resource
- is a transactional resource
- is accessed by a single task execution.

Transactional receive data accessed within a TP system:

- is accessed by a task that gets data from the transactional receive
- is a non-durable resource
- is a transactional resource (see Chapter 18 on page 319 for more information)
- is accessed by a single task execution.

3.8.9 Task Queue

A task queue accessed within the TP system:

- is accessed by a task that provides data for submitted task entries
- is a durable resource
- can be either a transactional or non-transactional resource
- can be accessed by concurrently executing transactions within a TP system
- must support at least isolation level 1.

3.8.10 Record Queue

A *record queue* is a named repository in which data items can be inserted and removed.

A record queue accessed within the TP system:

- is accessed for read and write by a task
- can be either a durable or non-durable resource
- is a transactional resource
- can be accessed by concurrently executing transactions within a TP system
- must support at least isolation level 1.

3.8.11 Audit Log

An *audit log* is a collection of recorded events that occurred during task executions.

The audit log accessed within the TP system:

- is accessed by a task that has write-only access to the audit log
- is a durable resource
- is a non-transactional resource
- can be accessed by concurrently executing transactions within a TP system.

The implementation must provide a method of transforming the audit log records written by tasks into a standard sequential file whose format is documented in Appendix B on page 487.

3.8.12 Transactional and Durable Attributes

Table 3-4 provides a table of the transactional and durable attributes of the resources used in STDL.

Resource	Durable	Non-durable	Transactional	Non-transactional
Private Workspace	N	Y	Y	Y
Shared Workspace	N	Y	Y	N
SQL Database	Y	N	Y	N
Indexed File	Y	N	Y	Y
Relative File	Y	N	Y	Y
Sequential File	Y	N	Y	Y
Stream File	Y	N	N	Y
Transactional Send	Y	N	Y	N
Transactional Receive	N	Y	Y	N
Task Queue	Y	N	Y	Y
Record Queue	Y	Y	Y	N
Audit Log	Y	N	N	Y

Table 3-4 Transactional and Durable Attributes of Resources

3.9 Message Group

A *message group* is a named set of messages that is defined using one or more <message-group-definition>s. Multiple <message-group-definition>s are used to support multiple human languages.

Each message in a message group defines a message code and a message text. *Application-defined* exceptions are defined as messages with an associated exception class. Application-defined exceptions can be raised by the STDL application and are available on each TP system that implements that message group.

For each message the following is defined:

- A message code as both:
  - — <message-identifier> that is a symbolic name for a message code
  - — the integer value defined for <message-identifier> is the numeric value for a message code. The integer value can be exchanged between programs.
- A message text that can be exchanged between programs.
- An exception class that refers to one of the standard predefined exception classes (see Appendix C on page 489).

Chapter 4

Common Elements

4.1 Character Sets

A *character set* is a named collection of characters. A character set can include characters from another character set.

Two types of character sets are defined for STDL: source character sets and execution character sets.

4.1.1 Source Character Sets

An STDL definition or specification is written using characters from the STDL source character sets. There are two source character sets defined for STDL:

1. STDL Alphanumeric

   This character set includes the following characters:

   — upper-case letters of the Latin alphabet (A-Z)

   — lower-case letters of the Latin alphabet (a-z)

   — numbers (0-9 decimal digits)

   — special characters:

!	Exclamation mark
"	Double quote
(	Open parenthesis
)	Close parenthesis
,	Comma
.	Period
-	Hyphen or minus sign
:	Colon
;	Semicolon
<	Less than
=	Equal
>	Greater than
_	Underscore
&	Ampersand
	Space
+	Plus sign

- `*` Multiplication sign
- `/` Division sign
- → Horizontal tab
- <FF> Form feed
- # Translation directive flag
- % Percent sign
- <eol> End of line character.

2. STDL Kanji

   This character set includes the following characters:

   — upper-case letters of the Latin alphabet (A-Z)

   — lower-case letters of the Latin alphabet (a-z)

   — numbers (0-9 decimal digits)

   — Hiragana (see JIS Kanji)

   — Katakana (see JIS Kanji)

   — Kanji (see JIS Kanji)

   — special characters:

   - `-` Hyphen
   - `_` Underscore
   - Prolonged sound symbol.

4.1.2 Execution Character Sets

The following character sets are defined for interpretation in the execution environment; an implementation can provide additional character sets:

Simple Latin The Simple Latin character set contains at least the characters defined in the alphanumeric character set in ISO Alphanumeric.

ISO Latin 1 The ISO Latin 1 character set contains the characters defined in ISO Latin 1.

ISO Latin 2 The ISO Latin 2 character set contains at least the characters defined in ISO Latin 2

Katakana The Katakana character set contains at least the characters defined in the Katakana and alphanumeric character sets in JIS Katakana.

Kanji The Kanji character set contains at least the characters defined in the Kanji character set in JIS Kanji.

ISO UCS-2 The ISO UCS-2 character set contains at least the characters defined in the ISO UCS-2 character set defined in the ISO UCS-2 standard.

See Appendix G on page 529 for information on the way that the information in the supported character set is transmitted between TP systems. The internal representation (or encoding) is implementation-specific.

In addition, the execution character sets can be used in string literals and comments.

Implementor Note:

An implementation can restrict the concurrent use of Katakana characters from the Katakana execution character set and lower-case Latin characters to characters from either the Simple Latin execution character set or the Katakana execution character set within a <task-definition> or any <data-type-definition> referenced by that <task-definition>.

4.2 Tokens

A *token* is the minimal lexical element of STDL. Tokens are:

- reserved words
- predefined words
- customer-defined words
- workspace field references
- literals
- operators
- punctuators
- message parameters.

Tokens are separated using white space, operators and punctuators. *White space* consists of any combination of one or more of the following:

- comment text (see Section 4.12 on page 95)
- space
- horizontal tab
- form feed
- end of line.

4.3 Reserved Words

Reserved words are those words used in STDL definition and specification syntax. Only the STDL alphanumeric character set can be used for reserved words. Lower-case, upper-case and mixed-case reserved words are equivalent. An implementation cannot extend the STDL reserved word list.

Table 4-1 lists the STDL reserved words.

AND	EXECUTION	NATIONAL	SET
APPLICATION	EXIT	NEXT	SHARED
ARE		NO	SIZE
ARRAY	FIELD	NOMATCH	SOURCE
AS	FIRST	NOT	SPECIFICATION
AT	FOR	NUMBER	STRING
AUDIT	FROM		SUBMIT
		OCTET	SUBMITTER
BLOCK	GET	OF	SYSTEM
BROADCAST	GO	ON	
BY	GOTO	OPERATOR	TASK
	GROUP	OR	TERMINATION
CALL		OUTPUT	TEXT
CANCEL	HANDLER		THEN
CASE	HOLD	PASSED	TO
CHARACTER		PRESENTATION	TRANSACTION
CLASS	ID	PRIVATE	TRANSACTIONAL
CLIENT	IDENTIFIER	PROCEDURE	TRUE
CODE	IF	PROCESSING	TYPE
COMMIT	IN		
COMPOSABLE	INDEPENDENT	QUEUE	UNTIL
CONCURRENT	INITIALIZATION		UPDATE
CONTROL	INOUT	RAISE	USING
	INPUT	READ	UUID
DECIMAL	INTEGER	RECEIVING	
DEPENDENT	INTO	RECORD	VALUE
DEPENDING	IS	REPEATING	VERSION
DEQUEUE		RERAISE	
DISPLAY	KEY	RESTART	WAIT
DO		RESTARTABLE	WHILE
	LANGUAGE	ROLLBACK	WITH
ELSE	LENGTH		WORK
END	LIST	SCALE	WORKSPACE
ENQUEUE		SELECT	WORKSPACES
EVERY	MESSAGE	SEND	
EXCEPTION		SENDING	

Table 4-1 Reserved Words

4.4 Predefined Words

Only the STDL alphanumeric character set can be used for predefined words. Lower-case, upper-case and mixed-case predefined words are equivalent. Predefined words are used in this specification for:

- character set identifiers
- class identifiers
- system workspace identifiers.

4.4.1 <character-set-identifier>

<character-set-identifier> identifies the character set to be used in the text data type. The following character set names are defined:

SIMPLE-LATIN	Specifies that the Simple Latin character set is used.
ISO-LATIN-1	Specifies that the ISO Latin 1 character set is used.
ISO-LATIN-2	Specifies that the ISO Latin 2 character set is used.
KATAKANA	Specifies that the Katakana character set is used.
KANJI	Specifies that the Kanji character set is used.
ISO-UCS-2	Specifies that the ISO UCS-2 character set is used.

4.4.2 <class-identifier>

<class-identifier> is the name of one of the standard, predefined exception classes. These are listed in Appendix C on page 489.

4.4.3 System Workspace Identifiers

The following system <workspace-identifier>s are defined:

- EXCEPTION-INFO-WORKSPACE
- SYSTEM-INFO-WORKSPACE.

4.5 OS Names

Operating system names (OS names) are defined outside the scope of STDL definitions and COBOL and C programs that are used in STDL definitions.

OS names must:

- contain between 1 and 255 characters
- contain only the following characters from the Simple Latin character set:
  - — letters (upper-case and lower-case characters are equivalent)
  - — numbers
  - — underscore (_) and hyphen (-)
- begin with a letter.

Implementor Notes:

1. OS names can be restricted to contain only 8 characters.
2. Underscore (_) and hyphen (-) can be disallowed when only 8 characters are allowed in an OS name.

The TP system must convert lower-case letters to upper-case when sending an OS name to another TP system in an RTI message.

4.5.1 Broadcast List

The TP system uses environmental information to map the name of the broadcast list to a broadcast list entity, as described in Section 17.2.5 on page 303.

The name of the broadcast list is unique, at least, on a TP system.

The name of the broadcast list is an OS name

4.5.2 Computer Language

The name of a computer language used to code processing and presentation procedures.

The computer language is an OS name.

The following computer languages must be supported:

- C
- COBOL.

4.5.3 Destination

The TP system uses environmental information to map the name of the destination to a destination entity, as described in Section 17.2.9 on page 306.

The name of the destination is unique, at least, on a TP system.

The name of the destination is an OS name.

4.5.4 Display

The TP system uses environmental information to map the name of the display to a display entity, as described in Section 17.2.7 on page 305.

The name of the display is unique on a TP system and is meaningful only on the TP system in which it is defined. Environmental information defines which displays are available to executing tasks within a TP system.

The name of the display is an OS name.

4.5.5 Distribution List

The TP system uses environmental information to map the name of the distribution list to a distribution list entity, as described in Section 17.2.8 on page 306.

The name of the distribution list is unique, at least, on a TP system.

The name of the distribution list is an OS name.

4.5.6 Human Language

The name of a human language is used by the TP system when executing a <get-message> to map to the correct <message-group-definition>.

The name of a human language is an OS name.

Human languages are defined using the format:

```
<language>_<territory>
```

<language> is a language symbol as defined in ISO 639: 1988.

<territory> is a country code (ALPHA-2) as defined in ISO 3166: 1993.

The following human languages are defined for compatibility with previous STDL versions:

- JAPANESE
- ENGLISH.

See Appendix F on page 527 for the list of human languages and execution character set mappings.

Implementor Notes:

1. An implementation is not required to support all of the language names as defined in ISO 639: 1988 or all of the country codes as defined in ISO 3166: 1993.
2. An implementation is allowed to map JAPANESE and ENGLISH to the appropriate <language>_<territory>.

A character set is associated with every human language. Message texts in a human language must contain only characters in the character sets associated with the human language. The association of human language names other than ENGLISH and JAPANESE is implementation-defined.

4.5.7 Record Queue

The name of the record queue specifies the record queue to which an <enqueue-record>, <dequeue-record> or <read-queue-record> refers.

The TP system uses environmental information to map the record queue name to an implementation-specific physical queue.

The name of the record queue is unique on a TP system.

The name of the record queue is an OS name.

4.6 Customer-defined Identifiers

Identifiers are defined using STDL definitions and COBOL and C programs. Identifiers are used to reference entities within a definition or entities in another, external definition.

Each identifier is classified in two ways:

- The scope in which the identifier exists.

  Determines the part of the definitions in which an identifier is available for use. STDL defines the following identifier scopes:

  External — The identifier is defined within a source file but outside of a definition. Once an external identifier is defined, it is available for use throughout the remainder of the source file.

  Internal — The identifier is defined within a definition and is available for use only within that definition.

- The name space in which the identifier exists.

  Determines the set of identifiers in which an identifier is grouped for the purpose of checking uniqueness. STDL defines the following name spaces:

  Unqualified — Either internal or external scope. An unqualified identifier is unique within its scope. An unqualified internal identifier hides an unqualified external identifier of the same name, except that an unqualified internal identifier does not hide an external unqualified identifier used for a <data-type-identifier>.

  Qualified — Either internal or external scope. A qualified identifier is qualified by either another qualified identifier (field identifiers only) or an unqualified identifier. Qualified identifiers are unique within the definition of the identifier by which they are qualified.

Implementor Notes:

1. An implementation can require that a qualified external identifier is treated as an unqualified external identifier.
2. An implementation can require that an unqualified external identifier is unique within a TP system.
3. An implementation can restrict the use of COBOL or C reserved words as external and internal identifiers. Vendor extensions to either the COBOL or C reserved word sets cannot be included in this restriction.

4.6.1 <external-identifier>

External identifiers must:

- contain between 1 and 31 characters
- contain only the following characters from the STDL alphanumeric character set:
  - — letters (upper-case and lower-case characters are equivalent)
  - — numbers
  - — underscore (_) and hyphen (-)

Underscore and hyphen are treated as equivalent characters.

- begin with a letter
- not duplicate STDL reserved or predefined words
- not end with a hyphen (-) or an underscore (_).

Implementor Notes:

1. External identifiers can be restricted to contain only 8 characters.
2. Underscore (_) and hyphen (-) can be disallowed when only 8 characters are allowed in an external identifier.

The TP system must convert lower-case letters to upper-case when sending an external identifier to another TP system in an RTI message.

4.6.1.1 <data-type-identifier>

<data-type-identifier> identifies a data type definition.

<data-type-identifier> is an unqualified external identifier.

<data-type-identifier>s are used by tasks, procedures in presentation groups, and procedures in processing groups to define arguments for passing data back and forth.

A record queue uses <data-type-identifier> to define the record format of the queue.

4.6.1.2 <message-group-identifier>

<message-group-identifier> identifies a message group.

A <message-group-identifier> is an unqualified external identifier.

4.6.1.3 <message-identifier>

<message-identifier> identifies a message within a message group.

A <message-identifier> is an unqualified external identifier.

When a message group name or message group UUID is available, <message-identifier> can be used as a qualified external identifier.

4.6.1.4 <presentation-group-identifier>

<presentation-group-identifier> identifies a presentation group.

A <presentation-group-identifier> is an unqualified external identifier.

4.6.1.5 <presentation-procedure-identifier>

<presentation-procedure-identifier> identifies a presentation procedure in a presentation group.

A <presentation-procedure-identifier> is a qualified external identifier. <presentation-procedure-identifier> is qualified by the <presentation-group-identifier> of the presentation group specification in which the presentation procedure is defined.

4.6.1.6 <processing-group-identifier>

<processing-group-identifier> identifies a processing group.

A <processing-group-identifier> is an unqualified external identifier.

4.6.1.7 <processing-procedure-identifier>

<processing-procedure-identifier> identifies a top-level processing procedure in a processing group.

A <processing-procedure-identifier> is a qualified external identifier. <processing-procedure-identifier> is qualified by the <processing-group-identifier> of the processing group specification in which the top-level processing procedure is defined.

4.6.1.8 <task-group-identifier>

<task-group-identifier> identifies a task group.

A <task-group-identifier> is an unqualified external identifier.

4.6.1.9 <task-identifier>

<task-identifier> identifies a task within a task group.

A <task-identifier> is a qualified external identifier. <task-identifier> is qualified by the <task-group-identifier> of the task group specification in which the task is defined.

4.6.2 <internal-identifier>

Internal identifiers must:

- contain between 1 and 31 characters
- contain characters from either the STDL alphanumeric or STDL Kanji character set. An internal identifier cannot contain characters from both the STDL alphanumeric and STDL Kanji character sets.
  - If from the STDL alphanumeric character set, the following characters can be used:
    - letters (upper-case and lower-case characters are equivalent)
    - numbers
    - underscore (_) and hyphen (-).

      Underscore and hyphen are treated as equivalent characters.
  - If from STDL Kanji character set, the following characters can be used:
    - underscore (_) and hyphen (-)

      Underscore and hyphen are treated as equivalent characters.
    - numeric
    - alphabetic
    - Hiragana
    - Katakana
    - Kanji

— prolonged sound symbol.

- Begin with either a letter from the STDL alphabetic character set or an alphabetic, Hiragana, Katakana or Kanji character from the STDL Kanji character set.
- Not duplicate STDL reserved or predefined words.
- Not end with a hyphen (-) or an underscore (_).

Implementor Notes:

1. Internal identifiers that contain STDL alphanumeric characters can be restricted to 30 characters.
2. Internal identifiers that contain STDL Kanji characters can be restricted to 13 characters.
3. An implementation is not required to support Kanji identifiers.

4.6.2.1 <field-identifier>

<field-identifier> identifies a <field-definition> within a <record-type>.

A <field-identifier> is a qualified internal identifier. <field-identifier> is qualified by the record <field-identifier> or <workspace-identifier> in which the <field-identifier> appears. The scope of a <field-identifier> is the <record-type> in which it is defined. A <field-identifier> can be accessed using less than its fully-qualified name as described in Section 4.7.1 on page 89.

4.6.2.2 <label-identifier>

<label-identifier> identifies an executable statement within a <task-definition>.

A <label-identifier> is an unqualified internal identifier. The scope of a <label-identifier> is the <task-definition> in which it is defined.

4.6.2.3 <workspace-identifier>

<workspace-identifier> identifies a private or shared structured area of memory used for data storage in a task. A <workspace-identifier> is defined by a <workspace-definition> in a <task-definition>.

<workspace-identifier> is an unqualified identifier. The scope of <workspace-identifier> depends on whether the <workspace-identifier> has been declared as a private or shared workspace:

- For a private workspace, the scope of a <workspace-identifier> is internal to the <task-definition> in which it is defined.
- For a shared workspace, the scope of a <workspace-identifier> is external but the shared workspace name follows the rules for <internal-identifier>.

4.6.3 Syntactical References to Identifiers

Identifiers are defined and then used within an STDL definition. When an internal identifier is defined, the syntax production represents the identifier as <internal-identifier>. When an external identifier is defined, the syntax production represents the identifier as <external-identifier>.

When an identifier is used after its definition, the syntax production represents the identifier using one of the syntax references defined in this section; for example, <task-identifier>. For an identifier to be used, the identifier must be defined and be in scope. For all identifiers other than <label-identifier>, this means that the definition of the identifier must appear before its use. <label-identifiers> can be used before they are defined.

4.7 Workspace Fields

This section defines the syntax and semantics for workspace field references.

4.7.1 <workspace-field>

NAME

<workspace-field> — Identifies a field within a workspace or the entire workspace.

SYNOPSIS

```
<workspace-field> ::
    <workspace-field-element> . ...

<workspace-field-element> ::
      <workspace-identifier>
    | <field-identifier-1>
    | <field-identifier-2> ( <workspace-field-subscript-list> )

<workspace-field-subscript-list> ::
    <workspace-field-subscript> , ...

<workspace-field-subscript> ::
      <integer-literal>
    | <integer-workspace-field>
```

SYNTAX RULES

1. <field-identifier-2> must have a type of array.
2. A <field-identifier-1> in a <workspace-field-element> that is not the last in a <workspace-field> must have a data type of record.
3. A <field-identifier-2> in a <workspace-field-element> that is not the last in a <workspace-field> must have a data type of array (to any level of nesting) of record.
4. <field-identifier-2> can have a data type of array only if it is used in the last <workspace-field-element> in a <workspace-field>.
5. The number of <workspace-field-subscript>s in a <workspace-field-subscript-list> must match the array nesting of the corresponding <field-definition> for <field-identifier-2> in the <data-type-definition> associated with the workspace.
6. <workspace-identifier> can be used as the first or only <workspace-field-element> in a <workspace-field>.
7. The first or only <field-identifier-1> in a <workspace-field> must be unique within the scope of the task.
8. The first or only <field-identifier-2> in a <workspace-field> must be unique within the scope of the task.
9. Any <field-identifier-1> or <field-identifier-2> other than the first in a <workspace-field> must be unique within the scope of the previous <workspace-identifier>, <field-identifier-1> or <field-identifier-2> in the <workspace-field>.
10. If a <field-identifier-1> or <field-identifier-2> is contained within a field that is an array, the <field-identifier> for that array must be present in the list of <workspace-field-element>s that precede the <field-identifier-1> or <field-identifier-2>.

GENERAL RULES

1. The data type of the last or only <field-identifier-1> or <field-identifier-2> determines the data type of <workspace-field>. <field-identifier-1> or <field-identifier-2> can be a field identifier of:

   — an octet field

   — an integer field

   — a text field

   — a decimal string field

   — an array field (either specifying a specific array element using <workspace-field-subscript-list> or the entire array by omitting the <workspace-field-subscript-list>)

   — a record (specified using either a <record-type> or a <data-type-identifier>)

   — a UUID field.

2. <workspace-field-subscript> references a specific element of the <field-identifier> within <workspace-field-element>.

3. Array elements are stored as rows. The rightmost subscript varies fastest as elements are accessed in storage order.

4.7.2 Syntax References

STDL syntax references can restrict the data type permitted for a <workspace-field>. When the data type is not restricted by the syntax, a generic <workspace-field> reference is used. When the data type is restricted by the syntax, the <workspace-field> is referenced using one of the following data type-specific workspace field names:

- <octet-workspace-field> is a <workspace-field> of data type OCTET.
- <integer-workspace-field> is a <workspace-field> of data type INTEGER.
- <text-workspace-field> is a <workspace-field> of data type TEXT or NATIONAL TEXT.
- <decimal-workspace-field> is a <workspace-field> of data type DECIMAL STRING.
- <array-workspace-field> is a <workspace-field> of data type ARRAY.
- <record-workspace-field> is a <workspace-field> of data type RECORD or a <workspace-identifier>.
- <s-text-workspace-field> is a <workspace-field> of data type TEXT CHARACTER SET SIMPLE-LATIN.
- <id-workspace-field> is a <workspace-field> of data type ARRAY SIZE 16 OF OCTET.

  The <id-workspace-field> can contain a task submission request entry identifier or a record queue element identifier.
- <uuid-workspace-field> is a <workspace-field> of data type UUID.

4.8 Literals

4.8.1 <integer-literal>

<integer-literal> is a sequence of characters selected from the numeric characters 0 through 9, the plus sign (+), and the minus sign (-) of the STDL alphanumeric character set.

The rules for an <integer-literal> are:

- Must contain at least one numeric character.
- Must contain at most one sign. If a sign is used, it must be the first character. If a sign is not used, then the <integer-literal> is non-negative.

4.8.2 <decimal-literal>

<decimal-literal> is a sequence of characters selected from the numeric characters 0 through 9, the plus sign (+), the minus sign (-), and the decimal point (.) of the STDL alphanumeric character set.

The rules for <decimal-literal>s are:

- Must contain at least one numeric character.
- Must contain at most one sign. If a sign is used, it must be the first character. If a sign is not used, then the <decimal-literal> is non-negative.
- Must contain at most one decimal point. The decimal point can appear anywhere except before a sign. If the decimal point is omitted, then the <decimal-literal> is treated as if there is a decimal point after the last character.

4.8.3 <string-literal>

<string-literal> is a sequence of characters.

The rules for a <string-literal> are:

- Can contain characters from a single execution character set.
- Can contain characters from any one of the execution character sets as defined in Section 4.1.2 on page 76.
- Must start and end with double quotes (") from the STDL alphanumeric character set.
- If a double quote is to be included in a <string-literal>, it must be represented by two double quote characters ("").
- Can contain from 0 to an implementation-specific maximum number of characters. Appendix A on page 481 defines the minimum number of characters an implementation must support in a <string-literal>.
- If two sequences of characters enclosed in double quotes have an ampersand (&) between them (ignoring any white space), they are treated as a single <string-literal>. The ampersand is from the STDL alphanumeric character set; the two sequences of characters must be from the same character set.

4.8.4 <string-literal> Syntax References

STDL syntax references can restrict the contents of a <string-literal> in both the character set allowed and the characters within that character set allowed. When the contents of a <string-literal> are restricted by the syntax, the <string-literal> is referenced using one of the following restricted string literal names:

- <os-name-literal> is a <string-literal> where the contents are restricted to being an OS name.
- <n-string-literal> is a <string-literal> where the contents are restricted to be the numeric characters 0 through 9 from the Simple Latin character set.
- <uuid-string-literal> is a <string-literal> where the contents are restricted to being the numeric characters 0 through 9, a through f, A through F, and the hyphen from the Simple Latin character set. A <uuid-string-literal> contains a UUID of the format eight hexadecimal digits followed by a hyphen, followed by three groups of four hexadecimal digits with each group separated by hyphens, followed by a hyphen, followed by twelve hexadecimal digits. The format of a <uuid-string-literal> is:

  ```
  xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
  ```

- <computer-language-literal> is a <string-literal> where the contents are restricted to one of the computer language names listed in Section 4.5.2 on page 81, which must be specified using the Simple Latin character set.
- <human-language-literal> is a <string-literal> where the contents are restricted to one of the human language names listed in Section 4.5.6 on page 82, which must be specified using the Simple Latin character set.

4.9 <operator>

There are three sets of operators: relational, arithmetic and assignment.

4.9.1 Relational Operators

Only the STDL alphanumeric character set can be used for a relational operator. The following relational operators are used in <boolean-expression>s:

=	Equal
<>	Not equal
<	Less than
<=, =<	Less than or equal
>	Greater than
>=, =>	Greater than or equal.

4.9.2 Arithmetic Operators

Only the STDL alphanumeric character set can be used for an arithmetic operator. The following arithmetic operators are used in <integer-expression>s:

+	Plus
-	Minus
/	Divide
*	Multiply.

4.9.3 Assignment Operator

Only the STDL alphanumeric character set can be used for the assignment operator. The following assignment operator is used in <assignment>s:

=	Assignment.

4.10 <punctuator>

Only the STDL alphanumeric character set can be used for punctuators. The following punctuators are used in STDL definitions and specifications:

: Colons

- — Separate a label from the syntax element or elements referenced by the label.
- — Separate a Boolean expression from the statement that is executed when the expression evaluates true, and the keyword NOMATCH from the statement that is executed when no match is found for a Boolean expression.

; Semicolons; terminate statements, field definitions and attributes.

, Commas; separate arguments when more than one argument is being passed.

. Periods; separate fields in a workspace name.

( ) Parentheses; can be used to enclose Boolean expressions and to delimit array subscripts.

4.11 Message Parameter

4.11.1 <message-parameter>

<message-parameter> allows the user to specify variable parameters in a <message-definition>. A <message-parameter> is used in a <message-text> to specify the order that identified the corresponding real argument to be substituted when getting the message text, and the position of the substitution in the message text.

The rules for a <message-argument> are:

- A <message-parameter> begins with a percent sign (%) and is followed by an <integer-literal> n without a sign. There is no white space between the percent sign and the <integer-literal>.
- The <integer-literal> n denotes that the parameter is to be substituted by the nth real argument when getting the message text at run time.

4.12 Comment Text and Comment Character

Comment text is included in STDL definitions and specifications using the exclamation mark (!) as a comment character.

Comment text can contain characters from the alphanumeric, ISO Latin 1, ISO Latin 2, Katakana or Kanji character sets as defined in ISO Alphanumeric, ISO Latin 1, ISO Latin 2, JIS Katakana and JIS Kanji, respectively. Comment text can also contain horizontal tabs and form feeds. Comment text is ignored when definitions and specifications are processed into executable entities.

A *comment character* marks the beginning of comment text. All characters following the comment character in a source line are treated as comment text. Comment text is terminated by the end of a source line.

Within a <string-literal>, a comment character is treated as part of the <string-literal>.

Comment text can be included by:

- beginning a source line with an exclamation point
- inserting an exclamation point within a source line.

Comment text cannot be continued from one source line to the next.

4.13 Source File and Source Line

A *source file* is an implementation-specific unit within which is stored the text of an STDL definition or specification. A source file is subject to preprocessing before translation. A source line is a line in a source file that contains STDL tokens.

One source line is separated from the next source line using an implementation-specific mechanism.

When exchanging source files from one vendor's system to another vendor's system, the end-of-line character must separate one source line from the next source line. Source code transfer is performed on source files before they are preprocessed.

A token cannot be continued from one line to the next.

After preprocessing, an STDL source file can contain one or more STDL definitions. An STDL definition is a <data-type-definition>, <task-group-specification>, <task-definition>, <processing-group-specification>, <presentation-group-specification> or <message-group-definition>. STDL source files can be defined, preprocessed and translated in any order.

If a data type reference includes a <data-type-identifier>, the definition of that identifier must precede its use.

An implementation must at least support the following types of source files:

- Data Type Source File

  A data type source file consists of zero or more <data-type-definition>s.

- Task Group Source File

  A task group source file consists of zero or one <task-group-specification> and any referenced <data-type-definition>s, which must precede their references within the <task-group-specification>.

- Task Source File

  A task source file consists of zero or one <task-definition> and the following referenced entities, which must precede the <task-definition> in the source file:

  — <data-type-definition>s referenced by other <data-type-definition>s, the <task-definition>, <task-group-specification>s, <processing-group-specification>s or <presentation-group-specification>s in the source file, and must precede those references in that source file

  — <task-group-specification>s, including the task group specification which contains the task's <task-definition> and any <task-group-specification>s for tasks called or submitted by the task

  — <processing-group-specification>s

  — <message-group-definition>s

  — <presentation-group-specification>s.

- Processing Group Source File

  A processing group source file consists of zero or one <processing-group-specification> and any referenced <data-type-definition>s, which must precede their references within the <procesing-group-specification> or other <data-type-definition>s.

- Presentation Group Source File

A presentation group source file consists of zero or one <presentation-group-specification> and any referenced <data-type-definition>s, which must precede their references within the <presentation-group-specification> or other <data-type-definition>s.

- Message Group Source File

  A message group source file consists of zero or one <message-group-definition>.

4.14 Preprocessing

An STDL source file is subject to preprocessing before it is further processed during translation.

NAME

Preprocessing

SYNOPSIS

```
<preprocessing-file> ::
    <pre-group> ...

<pre-group> ::
      <nil>
    | <pre-group-part> ...

<pre-group-part> ::
      <pre-source-line>
    | <pre-processor-directive>

<pre-source-line> ::
    <pre-token> ... <eol>

<pre-processor-directive> ::
      <pre-define>
    | <pre-if>
    | <pre-include>
    | <pre-undef>

<pre-token> ::
      <pre-identifier>
    | <string-literal>
    | <integer-literal>
    | <decimal-literal>
    | <operator>
    | <punctuators>
```

SYNTAX RULES

1. The preprocessing syntax is separate from the rest of the STDL syntax and is processed before any other STDL syntax.
2. The preprocessing syntax defines <pre-identifier> as an <internal-identifier>. The scope of the definition is the preprocessing syntax.
3. A <pre-identifier> must:
   - Consist of 1 to 31 characters.
   - Contain characters from either the STDL alphanumeric or STDL Kanji character set. A <pre-identifier> cannot contain characters from both the STDL alphanumeric and STDL Kanji character sets.

     If from the STDL alphanumeric character set, the following characters can be used:
     - letters (upper-case and lower-case characters are equivalent)
     - numbers

  - — underscore (_) and hyphen (-).

    Underscore and hyphen are treated as equivalent characters.

  If from the STDL Kanji character set, the following characters can be used:

  - — underscore (_) and hyphen (-)

    Underscore and hyphen are treated as equivalent characters.
  - — numeric
  - — alphabetic
  - — Hiragana
  - — Katakana
  - — Kanji
  - — prolonged sound symbol.

- — Begin with either a letter from the alphabetic character set or an alphabetic, Hiragana, Katakana or Kanji character from the Kanji character set.

4. <eol> is an implementation-specific line separator.
5. The preprocessor directives are written using the STDL alphanumeric source character set. Lower-case Latin letters are treated the same as upper-case Latin letters in the directives.

GENERAL RULES

1. Preprocessing semantically consists of the following steps:
   - — All comments are removed and replaced by white space.
   - — The preprocessing syntax is applied to the source file. During preprocessing, line demarcation is significant as indicated in the syntax by <eol>.
   - — Any <pre-processor-directive> is interpreted according to the rules for each preprocessor directive.
   - — Any <pre-source-line> that is included in the translation (see Section 4.14.2 on page 103 for how source lines can be omitted) is scanned for <pre-identifier>s. If a preprocessor macro is defined for a <pre-identifier>, the <pre-define-replacement-list> defined for that <pre-identifier> is substituted for the <pre-identifier>. The <pre-define-replacement-list> is also subject to macro replacement except for any <pre-identifier> identical to a previously-replaced <pre-identifier>. Macro replacement is recursive, except in the case of identical <pre-identifier>s.
2. If source files are included via the <pre-include> directive, General Rule 1. is applied to the source included.
3. Unless otherwise stated, preprocessor lines are not subject to macro substitution.

IMPLEMENTOR NOTES

1. An implementation is not required to support # IF <pre-expression>.
2. An implementation can restrict <pre-identifier> to not contain Kanji characters when <pre-identifier> is used as the first argument of # DEFINE, # UNDEF, # IFDEF and # IFNDEF.

3. An implementation can restrict <pre-identifier> to not contain hyphens when <pre-identifier> is used as the first argument of # DEFINE, # UNDEF, # IFDEF and # IFNDEF.

NOTES

1. During preprocessing, the STDL source file is examined on a line-by-line basis. This is in contrast to the translation of the STDL syntax which is done without regard to lines.
2. The rules for <pre-identifier> are a superset of reserved words, predefined words and customer-defined words. During preprocessing, reserved words, pre-defined words and customer-defined words are all treated as <pre-identifier>s. For example, it is possible, but not recommended, to replace all occurrences of an STDL reserved word with another word during pre-processing.
3. A source line whose first non-white space character is the translation directive flag character (#) is a preprocessing directive.
4. Preprocessing directive lines may contain comments. Other than # DEFINE, no STDL syntax can appear on preprocessing directive lines.
5. The preprocessor directives are not reserved words since they are lexically identified by the preprocessor directive flag (#).
6. An implementation does not have to preprocess an STDL source file separately from translation. The semantic description is done in separate steps to simplify the description.

4.14.1 <pre-define>

NAME

<pre-define> — Defines a preprocessor replacement macro.

SYNOPSIS

```
<pre-define> ::
    # DEFINE <pre-identifier> <pre-define-replacement-list> <eol>

<pre-define-replacement-list> ::
      <nil>
    | <pre-token> ...
```

SYNTAX RULES

1. <pre-define> defines <pre-identifier> as a preprocessor macro. This definition is valid for the rest of the source file unless undefined by a <pre-undef>.
2. If <pre-identifier> has been previously defined as a preprocessor macro, then the <pre-define-replacement-list> must consist of identical tokens to the previous definition.

GENERAL RULES

1. The <pre-define-replacement-list> is saved as the definition of <pre-identifier>. In any source after <pre-define> and before any <pre-undef> for the <pre-identifier>, any occurrence of <pre-identifier> in the source is replaced by <pre-define-replacement-list>.
2. The <pre-define> line is removed before translation.

NOTES

1. The scope of preprocessor macros is independent of the non-preprocessor syntax of STDL.
2. Since <pre-define-replacement-list> can be <nil>, it is possible to remove occurrences of <pre-identifier> in the source by using a <pre-define> with a nil <pre-define-replacement-list>.
3. The <pre-identifier> and <pre-define-replacement-list> is not subject to macro replacement in <pre-define>.

4.14.2 <pre-if>

NAME

<pre-if> — Allows conditional inclusion of source.

SYNOPSIS

```
<pre-if> ::
    <pre-if-type> <pre-elif-list> <pre-else> <pre-end-if>

<pre-if-type> ::
      # IF <pre-expression> <eol> <pre-group>
    | # IFDEF <pre-identifier> <eol> <pre-group>
    | # IFNDEF <pre-identifier> <eol> <pre-group>

<pre-expression> ::
    <boolean-expression>

<pre-elif-list> ::
    <nil>
    | <pre-elif> ...

<pre-elif> ::
    # ELIF <pre-expression> <eol> <pre-group>

<pre-else> ::
    <nil>
    | # ELSE <eol> <pre-group>

<pre-end-if> ::
    # ENDIF <eol>
```

SYNTAX RULES

None.

GENERAL RULES

1. Within a <pre-if>, at most one <pre-group> is included in the source to be translated.
2. For # IF, the <pre-group> is included only if <pre-expression> evaluates to true.
3. For # IFDEF, the <pre-group> is included only if <pre-identifier> has been defined as a preprocessor macro.
4. For # IFNDEF, the <pre-group> is included only if <pre-identifier> has not been defined as a preprocessor macro.
5. For # ELIF, the <pre-group> is included only if no previous <pre-group> within the <pre-if> has been included and <pre-expression> evaluates to true.
6. For # ELSE, the <pre-group> is included only if no previous <pre-group> within the <pre-if> has been included.
7. A <boolean-expression> in a <pre-if> refers to preprocessor identifiers and integer literals.
8. A <boolean-expression> in a <pre-if> cannot contain a <string-literal>.
9. Before evaluating <boolean-expression> for <pre-expression>, the following is done in this order:

— Any occurrence of DEFINED (<identifier>) is replaced by:

  — If <identifier> is defined as a preprocessor macro, then by a <boolean-term> which evaluates to true.

  — Otherwise, by a <boolean-term> which evaluates to false

— All <pre-identifiers> that are defined as preprocessor macros are replaced by the corresponding <pre-define-replacement-list> as discussed above.

— All remaining <pre-identifier>s other than AND, OR and NOT are replaced by <boolean-term>s that evaluate to false.

10. In <pre-group>s not included in the source, no other preprocessing of the source text in those <pre-group>s is performed.

11. The <pre-if-type>, <pre-elif>, <pre-else> and <pre-end-if> lines are removed before translation.

NOTES

1. When skipping text, the translator must be tracking preprocessor directives only to find a matching # ENDIF or # ELSE.

2. The <pre-identifier> is not subject to macro substitution in <pre-ifdef> and <pre-ifndef>.

4.14.3 <pre-include>

NAME <pre-include> — Includes in the source file for translation all source lines from an external source file.

SYNOPSIS

```
<pre-include> ::
    # INCLUDE <os-name-literal> <eol>
```

SYNTAX RULES

1. # INCLUDE is the directive operator for the <include> translation directive.
2. <os-name-literal> is the directive operand for the <include> translation directive.

GENERAL RULES

1. Any <pre-identifiers> in the <pre-include> are subject to macro substitution.
2. <os-name-literal> is resolved by an implementation-specific mechanism into the name of an external source file.
3. Translation cannot continue if the external source file specified by <os-name-literal> does not exist.
4. Source lines from the external source file lexically unconditionally replace the source line that contains the <include> directive.
5. Any translation directives encountered in the included source lines from the external source file are processed by the same rules. Recursive references to the same external source file are prohibited.
6. The <pre-include> line is removed before translation.

Implementor Note:

An implementation can restrict the use of <pre-include> to the inclusion of external source files for <message-group-definition>s, <presentation-group-specification>s, <processing-group-specification>s, <task-group-specification>s and <data-type-definition>s.

NOTES

1. Since macro substitution is allowed in <pre-include>, a macro can be used to represent the external file. For example:

```
#if defined (system1)
#define file "sys1"
#else
#define file "default"
#endif

#include file
```

4.14.4 <pre-undef>

NAME

<pre-undef> — Removes a preprocessor replacement macro definition.

SYNOPSIS

```
<pre-undef> ::
    # UNDEF <pre-identifier> <eol>
```

SYNTAX RULES

1. <pre-undefine> removes the definition of <pre-identifier> as a preprocessor macro for the rest of the source file unless another <pre-define> is used.
2. If <pre-identifier> has not been previously defined as a preprocessor macro, then the <pre-undef> is ignored.

GENERAL RULES

1. The <pre-undef> line is removed before translation.

NOTES

1. <pre-undef> can be used before a <pre-define> to redefine a preprocessor macro to have a different <pre-define-replacement-list>.
2. The <pre-identifier> is not subject to macro replacement in <pre-undef>.

4.15 <boolean-expression>

Boolean expressions are conditions that produce a truth value when evaluated.

NAME

<boolean-expression>

SYNOPSIS

```
<boolean-expression> ::
      <boolean-and-expression>
    | <boolean-expression> OR <boolean-and-expression>
```

SYNTAX RULES

None.

GENERAL RULES

1. Within a <boolean-expression>, the order of evaluation is indicated by the syntax. Each syntax production defines the semantics for evaluating part of a Boolean expression which results in a value. That value is then used when evaluating the next part of the Boolean expression defined by another syntax production.
2. Each part of a <boolean-expression> has the value true or false.
3. The value of <boolean-expression> is:

— If no OR is used, the value of <boolean-and-expression>.

— If OR is used, the logical inclusive OR of the value of <boolean-expression> and the value of <boolean-and-expression>. If one or both are true, then the value is true. If both are false, the value is false.

4.15.1 <boolean-and-expression>

NAME

<boolean-and-expression>

SYNOPSIS

```
<boolean-and-expression> ::
      <boolean-term>
    | <boolean-and-expression> AND <boolean-term>
```

SYNTAX RULES

None.

GENERAL RULES

1. The value of <boolean-and-expression> is:
   - — If no AND is used, the value of <boolean-term>.
   - — If AND is used, the logical conjunction AND of the value of <boolean-expression> and the value of <boolean-and-expression>. If both are true, then the value is true. If one or both are false, the value is false.

4.15.2 <boolean-term>

NAME

<boolean-term>

SYNOPSIS

```
<boolean-term> ::
        <boolean-comparison>
      | <boolean-negated-condition>
      | ( <boolean-expression> )
```

SYNTAX RULES

None.

GENERAL RULES

1. The value of <boolean-term> is the value of <boolean-comparison>, <boolean-negated-condition> or <boolean-expression>.

4.15.3 <boolean-comparison>

NAME

<boolean-comparison> — Consists of a single relational expression.

SYNOPSIS

```
<boolean-comparison> ::
    <boolean-operand-1> <relational-operator> <boolean-operand-2>

<boolean-operand> ::
        <workspace-field>
      | <message-identifier>
      | <message-group-identifier>
      | <class-identifier>
      | <decimal-literal>
      | <integer-literal>
      | <string-literal>

<relational-operator> ::
        =
      | <>
      | <
      | <=
      | =<
      | >
      | >=
      | =>
```

SYNTAX RULES

None.

GENERAL RULES

1. The value of <boolean-comparison> depends on the <relational-operator> used:

   — If "=" is used, the value is true when <boolean-operand-1> is equal to <boolean-operand-2>. Otherwise the value is false.

— If "<>" is used, the value is true when <boolean-operand-1> is not equal to <boolean-operand-2>. Otherwise, the value is false.

— If "<" is used, the value is true when <boolean-operand-1> is less than <boolean-operand-2>. Otherwise, the value is false.

— If "<=" or "=<" is used, the value is true when <boolean-operand-1> is less than or equal to <boolean-operand-2>. Otherwise, the value is false.

— If ">" is used, the value is true when <boolean-operand-1> is greater than <boolean-operand-2>. Otherwise, the value is false.

— If ">=" or "=>" is used, the value is true when <boolean-operand-1> is greater than or equal to <boolean-operand-2>. Otherwise, the value is false.

2. The data type of <boolean-operand-1> and <boolean-operand-2> must be the same, according to the following rules:

   — An <octet-workspace-field> can be compared only with another <octet-workspace-field>.

   — An <integer-workspace-field> or <integer-literal> can be compared with either another <integer-workspace-field>, <message-identifier>, <class-identifier> or <integer-literal>.

   — A <text-workspace-field> or <string-literal> can be compared with either another <text-workspace-field> or <string-literal>. Comparisons of text strings must be performed on text strings of the same character set. Comparisons of alphabetic characters are case-sensitive.

   — A <decimal-workspace-field> or <decimal-literal> can be compared with either another <decimal-workspace-field> or <decimal-literal>.

   — An <array-workspace-field> can be compared only with another <array-workspace-field> with the same array element data type, the same level of array nesting, and the same size of each level of array nesting as defined in Section 5.1 on page 121. When comparing varying-length arrays, the values of the DEPENDING ON fields must be equal.

   — A <record-workspace-field> can be compared only with another <record-workspace-field> whose fields or subfields match in order, data type and size.

   — A <uuid-workspace-field> can be compared with another <uuid-workspace-field>, a <message-group-identifier> or a <uuid-string-literal>.

3. Comparisons of <octet-workspace-field>s are performed for equality only. No greater-than or less-than comparisons are supported.

4. Comparisons of <array-workspace-field>s and <record-workspace-field>s are performed for equality only. No greater-than or less-than comparisons are supported. These comparisons are performed on a field-by-field basis.

5. In comparisons of <text-workspace-field>s or <string-literal>s of unequal length, the shorter field or literal is padded with spaces to the right for the comparison.

6. In comparisons of <decimal-workspace-field>s or <decimal-literal>s, the decimal points of the two fields or literals are aligned. The field or literal with fewer digits to the left of the decimal point is padded with zeros to the left, and the field or literal with fewer digits to the right of the decimal point is padded with zeros to the right.

7. Comparisons of <uuid-workspace-field>s, <message-group-identifier>s and <uuid-string-literal>s are performed for equality only. No greater-than or less-than comparisons are supported.

4.15.4 <boolean-negated-condition>

NAME

<boolean-negated-condition> — Negates the <boolean-term> with logical operator NOT.

SYNOPSIS

```
<boolean-negated-condition> ::
    NOT <boolean-term>
```

SYNTAX RULES

None.

GENERAL RULES

1. The value of <boolean-negated-condition> is the logical converse of <boolean-term>. If the value of <boolean-term> is true, the value of <boolean-negated-condition> is false. If the value of <boolean-term> is false, the value of <boolean-negated-condition> is true.

4.16 <integer-expression>

Integer expressions result in an integer value.

NAME

<integer-expression>

SYNOPSIS

```
<integer-expression> ::
      <multiplicative-expression>
    | <integer-expression> + <multiplicative-expression>
    | <integer-expression> - <multiplicative-expression>
```

SYNTAX RULES

1. The + and - must be preceded by white space and followed by white space.

GENERAL RULES

1. Within an <integer-expression> the order of evaluation is indicated by the syntax. Each syntax production defines the semantics for evaluating part of an integer expression which results in a value. That value is then used when evaluating the next part of the integer expression defined by another syntax production.
2. The value of <integer-expression> is:
   — If no + or - is used, the value of <multiplicative-expression>.
   — If + is used, the result of the integer addition of the value of <integer-expression> and the value of <multiplicative-expression>.
   — If - is used, the result of the integer subtraction of the value of <multiplicative-expression> from the value of <integer-expression>.

4.16.1 <multiplicative-expression>

NAME

<multiplicative-expression>

SYNOPSIS

```
<multiplicative-expression> ::
      <unary-expression>
    | <multiplicative-expression> * <unary-expression>
    | <multiplicative-expression> / <unary-expression>
```

SYNTAX RULES

1. The * and / must be preceded by white space and followed by white space.

GENERAL RULES

1. The value of <multiplicative-expression> is:

   — If neither * or / is used, the value of <unary-expression>.

   — If * is used, the result of the integer multiplication of the value of <multiplicative-expression> and the value of <unary-expression>.

   — If / is used, the result of the integer division of the value of <multiplicative-expression> by the value of <unary-expression>. The result, including precision, of any necessary intermediate calculation is implementation-specific.

4.16.2 <unary-expression>

NAME

<unary-expression>

SYNOPSIS

```
<unary-expression> ::
      <primary-expression>
    | + <primary-expression>
    | - <primary-expression>
```

SYNTAX RULES

1. The + and - must be preceded by white space and followed by white space.

GENERAL RULES

1. The value of <unary-expression> is:

   — If - is not used, the value of <primary-expression>.

   — If - is used, the value resulting by changing the sign of the value of <primary-expression>.

4.16.3 <primary-expression>

NAME

<primary-expression>

SYNOPSIS

```
<primary-expression> ::
      <integer-literal>
    | <integer-workspace-field>
    | ( <integer-expression> )
```

SYNTAX RULES

None.

GENERAL RULES

1. The value of <primary-expression> is the value of <integer-literal>, the contents of <integer-workspace-field>, or the value of <integer-expression>.

4.17 Values

Values specify OS name, time and date.

4.17.1 <absolute-time-value>

NAME

<absolute-time-value> — Specifies a specific time and date.

SYNOPSIS

```
<absolute-time-value> ::
      <n-string-literal>
    | <s-text-workspace-field>
```

SYNTAX RULES

1. <n-string-literal> and <s-text-workspace-field> must be 14 characters in length.
2. The format of the data in <n-string-literal> or <s-text-workspace-field> must be YYYYMMDDHHMMSS, where:
   - — YYYY is the year from 0000 to 9999
   - — MM is the month from 01 to 12
   - — DD is the day from 01 to 31
   - — HH is the hour from 00 to 23
   - — MM is the minute from 00 to 59
   - — SS is the second from 00 to 59.

GENERAL RULES

None.

4.17.2 <delta-time-value>

NAME

<delta-time-value> — Specifies a time duration.

SYNOPSIS

```
<delta-time-value> ::
      <n-string-literal>
    | <s-text-workspace-field>
```

SYNTAX RULES

1. <n-string-literal> and <s-text-workspace-field> must be 10 characters in length.
2. The format of the data in <n-string-literal> or <s-text-workspace-field> must be DDDDHHMMSS, where:
   - — DDDD is the number of days from 0000 to 9999
   - — HH is the number of hours from 00 to 23
   - — MM is the number of minutes from 00 to 59
   - — SS is the number of seconds from 00 to 59.

GENERAL RULES

None.

4.17.3 <os-name-value>

NAME

<os-name-value> — Specifies an OS name.

SYNOPSIS

```
<os-name-value> ::
      <os-name-literal>
    | <s-text-workspace-field>
```

SYNTAX RULES

None.

GENERAL RULES

1. The value of <os-name-literal> or <s-text-workspace-field> is used as the OS name.

Chapter 5

Data Type Definitions

5.1 <array-type>

NAME

<array-type> — Specifies that the data type is a fixed or varying length array of any data type.

SYNOPSIS

```
<array-type> ::
      <fixed-length-array-type>
    | <varying-length-array-type>

<fixed-length-array-type> ::
    ARRAY
        SIZE <integer-literal-1>
        OF <data-type>

<varying-length-array-type> ::
    ARRAY
        SIZE <integer-literal-2>
        TO <integer-literal-3>
        DEPENDING ON <field-identifier>
        OF <data-type>
```

SYNTAX RULES

1. ARRAY is a sequence of identical data elements, with an index of positive numbers starting at 1.
2. For fixed length arrays:
   - <integer-literal-1> specifies the size of the array. The size must be greater than zero.
   - Each element in an array is assigned the same initial value, if any is defined for the <data-type>.
3. For varying length arrays:
   - <integer-literal-2> specifies the minimum size of the array. The minimum size must be greater than or equal to zero.
   - <integer-literal-3> specifies the maximum size of the array. The maximum size must be greater than the minimum size.
   - <field-identifier> specifies the depending on field for the array. The value of the depending on field specifies the current size of the array.
   - The depending on field must be a field in the same <field-list> as the field being defined. The depending on field must be of <data-type> INTEGER.
   - The <varying-length-array-type> can only be used as the last field in a <field-list>.

— The <record-type> containing a <field-list> with a <varying-length-array-type> can only be used in a <data-type-definition>. The <data-type-identifier> defined by that <data-type-definition> cannot be used in other <data-type-definition>s.

— A <varying-length-array-type> cannot be initialized.

GENERAL RULES

1. An array is a sequence of M identical data elements, where each element is uniquely identified by the <field-identifier> and an integer subscript in the range of (1...M).

   — For fixed length arrays, M is the size of the array.

   — For varying length arrays, M is the current size of the array which is the value of the depending on field.

2. The value of the depending on field must be greater than or equal to the minimum size and less than or equal to maximum size.

3. The value of the depending on field which determines the current size of a varying array is bound at the time of reference to a field that is the varying array, to a record field or workspace containing the varying array, or a workspace field that is an element of the varying array. This reference occurs at the beginning of the execution of a clause, at the end of the execution of a clause, or both at the beginning and at the end. The following determines when STDL defines that a workspace or workspace field is referenced in an executable clause:

   — If the workspace or workspace field is a source of data in the executable clause, then the workspace or workspace field is referenced at the beginning of the execution of the clause.

   — If the workspace or workspace field is a destination of data in the executable clause, then the workspace or workspace field is referenced at the end of the execution of the clause.

   The value of the depending on field can be changed between references to the entire array or array element.

4. The <data-type> of an ARRAY can be ARRAY. This can be repeated for an implementation-specific number of times. The number of levels is the level of array nesting for the <field-definition>.

5. For a field of <data-type> ARRAY, the array element data type is the last <data-type> in the field definition.

SEE ALSO

None.

5.2 <data-type>

NAME

<data-type> — Specifies a data type.

SYNOPSIS

```
<data-type> ::
       <array-type>
     | <data-type-identifier>
     | <decimal-string-type>
     | <integer-type>
     | <octet-type>
     | <record-type>
     | <text-type>
     | <uuid-type>
```

SYNTAX RULES

None.

GENERAL RULES

1. When <data-type-identifier> is used as a <data-type>, the <data-type-definition> of the <data-type-identifier> is used for the <data-type>.

SEE ALSO

Section 5.6, Section 5.5, Section 5.8, Section 5.4, Section 5.1, Section 5.7, Section 5.9.

5.3 <data-type-definition>

NAME

<data-type-definition> — Defines a data type.

SYNOPSIS

```
<data-type-definition> ::
    TYPE <data-type-identifier> [IS] <record-type> ;

<data-type-identifier> ::
    <external-identifier>
```

SYNTAX RULES

1. <data-type-definition> defines <external-identifier> as a <data-type-identifier>. The scope of the definition is the rest of the source file.

GENERAL RULES

1. A <data-type-definition> can be directly referenced within a <task-definition>, <task-group-specification>, <presentation-group-specification>, <processing-group-specification> or another <data-type-definition>.
2. The implementation must provide a way to generate a COBOL COPY file (see Section 14.7 on page 277) and a C include file (see Section 15.7 on page 287) for a <data-type-definition>.

SEE ALSO

Section 5.7.

5.4 <decimal-string-type>

NAME

<decimal-string-type> — Specifies that the data type is a decimal string and defines the initial value when the data type is used in a workspace.

SYNOPSIS

```
<decimal-string-type> ::
    DECIMAL STRING
        SIZE <integer-literal-1>
        <decimal-scale>
        <decimal-initializer>

<decimal-scale> ::
      <nil>
    | SCALE <integer-literal-2>

<decimal-initializer> ::
      <nil>
    | = <decimal-literal>
```

SYNTAX RULES

1. DECIMAL STRING represents a numeric value.
2. The default initial value is a plus sign (+) as the leftmost character followed by all zeros.
3. An initial value specification can be set using a <decimal-literal>.
4. <integer-literal-1> specifies the size of the decimal string. The size indicates the number of characters to be allocated, not including the space for a sign character nor the decimal point. The size must be positive.
5. <integer-literal-2> specifies the scale of the decimal string. The scale indicates the number of places to the right of the decimal point. The scale must be non-negative from 0 to less than or equal to the size.
6. The default scale is zero.
7. The characters in a decimal string are from the STDL alphanumeric character set.

GENERAL RULES

1. The leftmost character contains a sign (+ or -).
2. The sign is followed by the number of characters specified by the size, which are numerals from 0 to 9.
3. The position of the implied decimal point is determined using the scale number of characters from the right.
4. The data is aligned by decimal point and is moved to the receiving digit positions with zero fill or truncation on either end, as required.
5. When the scale is zero, the data item is treated as if it has an assumed decimal point immediately following its rightmost digit and is aligned as stated in General Rule 4.
6. Truncation or padding does not affect the leftmost character, which contains the sign.

SEE ALSO

None.

5.5 <integer-type>

NAME

<integer-type> — Specifies that the data type is an integer and defines the initial value when the data type is used in a workspace.

SYNOPSIS

```
<integer-type> ::
    INTEGER <integer-initializer>

<integer-initializer> ::
      <nil>
    | = <integer-literal>
```

SYNTAX RULES

1. INTEGER represents an integer value in the range of -2^{31} to $2^{31}-1$.
2. The default initial value is zero.

GENERAL RULES

1. The order of the bits for an INTEGER is implementation-specific.

SEE ALSO

None.

5.6 <octet-type>

NAME

<octet-type> — Specifies that the data type is an octet.

SYNOPSIS

```
<octet-type> ::
    OCTET
```

SYNTAX RULES

1. OCTET represents an 8-bit value.

GENERAL RULES

1. OCTET stores binary data not translated by the TP system.
2. The initial value of an OCTET field is undefined.

SEE ALSO

None.

5.7 <record-type>

NAME

<record-type> — Specifies that the data type is a record and specifies the fields in the record.

SYNOPSIS

```
<record-type> ::
    RECORD
        <field-list>
    END [RECORD]

<field-list> ::
    <field-definition> ...

<field-definition> ::
    <field-identifier> [IS] <data-type> ;

<field-identifier>  ::
    <internal-identifier>
```

SYNTAX RULES

1. RECORD defines an aggregate type consisting of a sequence of named <field-definition>s.
2. <field-definition> defines <internal-identifier> as a <field-identifier> within the <data-type-definition> or <record-type> being defined. The scope of this definition is the same as the scope of the data type being defined.

GENERAL RULES

1. <internal-identifier> must be unique within a <record-type>.

SEE ALSO

None.

5.8 <text-type>

NAME

<text-type> — Specifies that the data type is a text field and defines the initial value when the data type is used in a workspace.

SYNOPSIS

```
<text-type> ::
      TEXT <text-character-set> SIZE <integer-literal>
          <text-initializer>
    | NATIONAL TEXT SIZE <integer-literal>
          <text-initializer>

<text-character-set> ::
      <nil>
    | CHARACTER SET <character-set-identifier>

<text-initializer> ::
      <nil>
    | = <string-literal>
```

SYNTAX RULES

1. TEXT is a fixed array of characters from a specified character set.
2. The default initial value is spaces.
3. An initial value specification can be set using a <string-literal>.
4. <character-set-identifier> specifies a character set. The default character set is SIMPLE-LATIN.
5. <integer-literal> indicates the number of characters to be allocated. <integer-literal> must be positive.
6. NATIONAL TEXT can be defined either as TEXT CHARACTER SET KANJI, TEXT CHARACTER SET ISO-LATIN-1 or TEXT CHARACTER SET ISO-LATIN-2, depending on an implementation-specific mechanism.

GENERAL RULES

1. If the receiving data item is a TEXT field, the sending data is moved to the receiving character positions and aligned at the leftmost character position in the data item with space fill or truncation to the right, as required.

SEE ALSO

None.

5.9 <uuid-type>

NAME

<uuid-type> — Specifies that the data type is a Universal Unique Identifier (UUID).

SYNOPSIS

```
<uuid-type> ::
    UUID
```

SYNTAX RULES

1. UUID represents a UUID value.

GENERAL RULES

1. The initial value of a UUID field is undefined.

NOTES

1. For application programmers, the STDL UUID data type can be used to handle message group UUIDs. Within STDL, the <message-group-identifier> is treated as having the data type UUID. This allows a programmer to move <message-group-identifier> into a <uuid-workspace-field> and to compare <uuid-workspace-field>s (such as the EXCEPTION-CODE-GROUP field of the EXCEPTION-INFO-WORKSPACE) with a <message-group-identifier>.

SEE ALSO

None.

Chapter 6

Message Definition Language

6.1 <message-group-definition>

NAME

<message-group-definition> — Defines a set of exception codes and associated message texts.

SYNOPSIS

```
<message-group-definition> ::
    MESSAGE [GROUP] <message-group-identifier>
        <message-group-attribute-list>
        <message-list>
    END [MESSAGE] [GROUP] ;

<message-group-identifier> ::
    <external-identifier>
```

SYNTAX RULES

1. <message-group-definition> defines <external-identifier> as a <message-group-identifier>. The scope of the definition is the rest of the source file.

GENERAL RULES

1. The implementation must provide a way to generate a COBOL COPY file and C include file for a <message-group-definition>. Each application-defined exception code must be defined in a <message-group-definition> associated with the <task-definition> in which it is used. (See Section 14.7 on page 277 and Section 15.7 on page 287 for further information.)

2. Different human languages can be used within a task by associating multiple <message-group-definition>s with the <task-definition>; one for each human language. These rules must be followed:

   — The <message-group-identifier> must be the same for each <message-group-definition>.

   — Each <message-group-definition> must either specify no <message-group-uuid> or must specify a <message-group-uuid> with the same <uuid-string-literal>. See Section 6.2.2 on page 136.

   — Each <message-group-definition> must specify a different <os-name-literal> for its <message-group-language>. See Section 6.2.1 on page 135.

   — Each <message-group-definition> must define the same <message-definition>s.

   — Each <message-group-definition> can contain different <message-text>s for a <message-definition>.

SEE ALSO

Section 6.2, Section 6.3.

6.2 <message-group-attribute-list>

NAME

<message-group-attribute-list> — Specifies the attributes of a message group.

SYNOPSIS

```
<message-group-attribute-list> ::
    <message-group-attribute> ...

<message-group-attribute> ::
    <message-group-language>
    | <message-group-uuid>
```

SYNTAX RULES

1. Each <message-group-definition> must have one and only one <message-group-language>.
2. Each <message-group-definition> can have at most one <message-group-uuid>.

GENERAL RULES

None.

SEE ALSO

Section 6.2.1, Section 6.2.2.

6.2.1 <message-group-language>

NAME

<message-group-language> — Specifies the human language for message group texts.

SYNOPSIS

```
<message-group-language> ::
    LANGUAGE [IS] <human-language-literal> ;
```

SYNTAX RULES

1. <human-language-literal> specifies the human language used for the message texts.

GENERAL RULES

None.

SEE ALSO

None.

6.2.2 <message-group-uuid>

NAME

<message-group-uuid> — Specifies the UUID for the message group.

SYNOPSIS

```
<message-group-uuid> ::
    UUID [IS] <uuid-string-literal> ;
```

SYNTAX RULES

1. If <uuid-value> is not specified, the default <uuid-value> is zero.
2. <message-group-uuid> defines the UUID value for <message-group-identifier>.

GENERAL RULES

1. A UUID uniquely identifies a <message-group-definition> within the <message-group-definition>s specified using the same <message-group-language>. The same UUID can be used for multiple <message-group-definition>s if the <message-group-language>s for the <message-group-definition>s are different.
2. A UUID is system-generated in a manner that guarantees uniqueness. See the referenced X/Open DCE RPC specification for more information on the generation of UUIDs.

SEE ALSO

None.

6.3 <message-list>

NAME

<message-list> — Specifies the list of messages in a message group.

SYNOPSIS

```
<message-list> ::
    <message-definition> ...
```

SYNTAX RULES

None.

GENERAL RULES

None.

SEE ALSO

Section 6.3.1.

6.3.1 <message-definition>

NAME

<message-definition> — Defines a message and its associated message text.

SYNOPSIS

```
<message-definition> ::
    <message-identifier>
        VALUE [IS] <integer-literal>
        CLASS [IS] <class-identifier>
        TEXT [IS] <message-text> ;

<message-identifier> ::
    <external-identifier>
```

SYNTAX RULES

1. <message-definition> defines <external-identifier> as a <message-identifier> within the message group being defined. The scope of this definition is the rest of the source file.
2. <integer-literal> defines the integer value of <message-identifier> being defined. <integer-literal> must be a non-zero, positive number.

GENERAL RULES

1. The <integer-literal> for <message-identifier> must be unique within the message groups used by tasks in a task group, unless the message group has a UUID defined. If the message group has a UUID defined, then the <integer-literal> must be unique within the message group.
2. The <message-identifier> must be unique within the message groups used by tasks in a task group, unless the message group has a UUID defined. If the message group has a UUID defined, then the <message-identifier> must be unique within the message group.
3. When a <task-definition> or a top-level processing procedure raises an exception using the <message-number> or <message-identifier>:

   — The <class-identifier> is retrieved by the TP system for the <message-identifier>.

   — The initial type of the exception raised is based on the <class-identifier>.
4. Each <class-identifier> must refer to any one of the standard, predefined exception classes (see Appendix C on page 489). The standard, predefined exceptions classes are defined by the TP system and cannot be changed by the customer.

IMPLEMENTOR NOTES

1. An implementation is not required to check <integer-literal>s for uniqueness within a task group.
2. An implementation can check whether a message identifier has been defined at run time. Errors occurring at that time are mapped to AP-EXECUTION-FAULT.

SEE ALSO

Section 6.3.2.

6.3.2 <message-text>

NAME

<message-text> — Defines a message text associated with a message.

SYNOPSIS

```
<message-text> ::
    <message-text-element> ...

<message-text-element> ::
    <string-literal>
    | <message-parameter>
```

SYNTAX RULES

1. The maximum number of <message-parameter>s in a <message-text> that an implementation must support is defined in Appendix A on page 481.
2. The value of the <integer-literal> of a <message-parameter> ranges from 1 through the number of the <message-parameter>s in the <message-text>, and must be unique within the <message-text>.
3. Two sequences of <string-literal> without a <message-parameter> between them cannot be specified.

GENERAL RULES

1. The <string-literal>s are expressed using an execution character set suitable for the human language specified using <message-group-language>:
   - — For ENGLISH, the <string-literal>s must contain only characters from the Simple Latin or ISO Latin-1 execution character sets.
   - — For JAPANESE, the <string-literal>s must contain only characters from the Kanji execution character set.
   - — For any other language, the <string-literal>s must contain only characters from the character set associated with the human language specified by <message-group-language>. If the language is not listed in Appendix F on page 527, the association is implementation-defined.
2. The <string-literal>s make up the message text for a <message-definition>.
3. The <message-parameter>s are replaced with <text-workspace-field-2>s of a <get-message> when getting the message text.

SEE ALSO

Section 4.11.1.

Chapter 7

Presentation Group Specification

7.1 <presentation-group-specification>

NAME

<presentation-group-specification> — Specifies a set of interfaces to customer-written presentation procedures.

SYNOPSIS

```
<presentation-group-specification> ::
    PRESENTATION GROUP [SPECIFICATION] <presentation-group-identifier>
        <presentation-group-attribute-list>
        <presentation-procedure-list>
    END [PRESENTATION] [GROUP] [SPECIFICATION] ;

<presentation-group-identifier> ::
    <external-identifier>
```

SYNTAX RULES

1. <presentation-group-specification> defines <external-identifier> as a <presentation-group-identifier>. The scope of this definition is the rest of the source file.

GENERAL RULES

1. <presentation-group-identifier> is used in the <call-presentation-procedure> statement to identify the presentation group in which the presentation procedure is to be found.

SEE ALSO

Section 7.2, Section 7.3.

7.2 <presentation-group-attribute-list>

NAME

<presentation-group-attribute-list> — Defines the attributes for the presentation procedure group.

SYNOPSIS

```
<presentation-group-attribute-list> ::
    <presentation-group-attribute> ...

<presentation-group-attribute> ::
      <presentation-initialization-procedure>
    | <presentation-source-language>
    | <presentation-termination-procedure>
```

SYNTAX RULES

None.

GENERAL RULES

None.

SEE ALSO

Section 7.2.2, Section 7.2.1, Section 7.2.3.

7.2.1 <presentation-initialization-procedure>

NAME

<presentation-initialization-procedure> — Defines the presentation initialization procedure for the presentation group.

SYNOPSIS

```
<presentation-initialization-procedure> ::
    INITIALIZATION PROCEDURE [IS] <presentation-procedure-identifier> ;

<presentation-procedure-identifier> ::
    <external-identifier>
```

SYNTAX RULES

1. <presentation-initialization-procedure> defines <external-identifier> as a <presentation-procedure-identifier> within the presentation procedure group being defined. The scope of this definition is the rest of the source file.
2. There can be at most one <presentation-initialization-procedure> in a <presentation-group-specification>.

GENERAL RULES

1. The presentation initialization procedure establishes context required by a presentation procedure. This context is associated with a presentation group handle. (See Section 12.5.1 on page 261 and Section 12.7 on page 262 for further information.) The presentation initialization procedure is executed before any presentation procedure is executed.

SEE ALSO

None.

7.2.2 <presentation-source-language>

NAME

<presentation-source-language> — Defines the programming language used in the presentation procedures.

SYNOPSIS

```
<presentation-source-language> ::
    SOURCE [LANGUAGE] [IS] <computer-language-literal> ;
```

SYNTAX RULES

1. <computer-language-literal> specifies the computer language used for the presentation procedures.
2. There must be one and only one <presentation-source-language> in a <presentation-group-specification>.

GENERAL RULES

1. The computer language literal, C or COBOL, must match the programming language used to code the set of presentation procedures within the presentation group.

SEE ALSO

None.

7.2.3 <presentation-termination-procedure>

NAME

<presentation-termination-procedure> — Defines the presentation termination procedure for the presentation group.

SYNOPSIS

```
<presentation-termination-procedure> ::
    TERMINATION PROCEDURE [IS] <presentation-procedure-identifier> ;

<presentation-procedure-identifier> ::
    <external-identifier>
```

SYNTAX RULES

1. <presentation-termination-procedure> defines <external-identifier> as a <presentation-procedure-identifier> within the presentation procedure group being defined. The scope of this definition is the rest of the source file.
2. There can be at most one <presentation-termination-procedure> in a <presentation-group-specification>.

GENERAL RULES

1. The presentation termination procedure invalidates the presentation group context established by the initialization procedure. The presentation termination procedure is invoked when the presentation group context is no longer required. (See Section 12.5.1 on page 261 and Section 12.7 on page 262 for further information.)

SEE ALSO

None.

7.3 <presentation-procedure-list>

NAME

<presentation-procedure-list> — Defines the list of presentation procedures in the presentation group.

SYNOPSIS

```
<presentation-procedure-list> ::
    <presentation-procedure-interface> ...
```

SYNTAX RULES

None.

GENERAL RULES

None.

SEE ALSO

Section 7.3.1.

7.3.1 <presentation-procedure-interface>

NAME

<presentation-procedure-interface> — Specifies the interface for a presentation procedure.

SYNOPSIS

```
<presentation-procedure-interface> ::
    PROCEDURE <presentation-procedure-identifier>
    <presentation-procedure-argument-type> ;

<presentation-procedure-argument-type> ::
      <presentation-procedure-send-argument-list>
    | <presentation-procedure-receive-argument-list>
    | <presentation-procedure-transceive-argument-list>

<presentation-procedure-send-argument-list> ::
    SENDING <presentation-procedure-argument-list>

<presentation-procedure-receive-argument-list> ::
    RECEIVING <presentation-procedure-argument-list>

<presentation-procedure-transceive-argument-list> ::
    SENDING <presentation-procedure-argument-list-1>
    RECEIVING <presentation-procedure-argument-list-2>

<presentation-procedure-argument-list> ::
      <nil>
    | <data-type-identifier> , ...

<presentation-procedure-identifier> ::
      <external-identifier>
```

SYNTAX RULES

1. <presentation-procedure-interface> defines <external-identifier> as a <presentation-procedure-identifier> within the presentation procedure group being defined. The scope of this definition is the rest of the source file.
2. The <presentation-procedure-identifier> names the entry point in the presentation group for the customer-written procedure.

GENERAL RULES

1. The <presentation-procedure-identifier> defined must match the entry point for the presentation procedure.
2. A presentation procedure can be classified as one of the following:
   - — A send procedure, which is a presentation procedure whose <presentation-procedure-interface> specifies a <presentation-procedure-send-argument-list>.
   - — A receive procedure, which is a presentation procedure whose <presentation-procedure-interface> specifies a <presentation-procedure-receive-argument-list>.
   - — A transceive procedure, which is a presentation procedure whose <presentation-procedure-interface> specifies a <presentation-procedure-transceive-argument-list>.

3. Send procedure semantics are:

   — The arguments are passed to the presentation procedure from the <task-definition>.

   — If a presentation procedure modifies an argument passed using the SENDING phrase, it is undefined whether its modification is propagated to the calling task.

   — The arguments in the SENDING phrase must match the arguments specified on the corresponding SENDING phrase of the <call-presentation-send-list> of the <call-presentation-procedure> statement in the <task-definition> in:

     — Number

     — Order.

     — The presentation procedure is executed.

4. Receive procedure semantics are:

   — The presentation procedure is executed.

   — The arguments are passed from the presentation procedure to the <task-definition>.

   — The arguments in the RECEIVING phrase must match the arguments specified on the corresponding RECEIVING phrase of the <call-presentation-receive-list> of the <call-presentation-procedure> statement in the <task-definition> in:

     — Number

     — Order.

5. Transceive procedure semantics are:

   — The SENDING arguments are passed to the presentation procedure from the <task-definition>.

   — If a presentation procedure modifies an argument passed using the SENDING phrase, it is undefined whether its modification is propagated to the calling task.

   — The arguments in the SENDING phrase must match the arguments specified on the corresponding SENDING phrase of the <call-presentation-transceive-list> of the <call-presentation-procedure> statement in the <task-definition> in:

     — Number

     — Order.

     — The presentation procedure is executed.

   — The presentation procedure is executed.

   — The RECEIVING arguments are passed from the presentation procedure to the <task-definition>.

   — The arguments in the RECEIVING phrase must match the arguments specified on the corresponding RECEIVING phrase of the <call-presentation-transceive-list> of the <call-presentation-procedure> statement in the <task-definition> in:

     — Number

     — Order.

6. <data-type-identifier> refers to a <data-type-definition>. The <field-definition>s within the <record-type> must match in order, type, and size with the <workspace-field>s within the corresponding <workspace-identifier> in the SENDING phrase of the <call-presentation-send-list> of the <call-presentation-procedure> statement and the RECEIVING phrase of the <call-presentation-receive-list> of the <call-presentation-procedure> statement.

IMPLEMENTOR NOTES

1. An implementation can require that <presentation-procedure-identifier> be unique within a TP system.

SEE ALSO

None.

Chapter 8

Processing Group Specification

8.1 <processing-group-specification>

NAME

<processing-group-specification> — Specifies a set of interfaces to top-level processing procedures.

SYNOPSIS

```
<processing-group-specification> ::
    PROCESSING GROUP [SPECIFICATION] <processing-group-identifier>
        <processing-source-language>
        <processing-procedure-list>
    END [PROCESSING] [GROUP] [SPECIFICATION] ;

<processing-group-identifier> ::
    <external-identifier>
```

SYNTAX RULES

1. <processing-group-specification> defines <external-identifier> as a <processing-group-identifier>. The scope of this definition is the rest of the source file.

GENERAL RULES"

1. <processing-group-identifier> is used in the <call-procedure> in a <task-definition> to identify the processing group in which the top-level processing procedure is to be found.
2. A <processing-procedure> must be included in the <processing-group-specification> for each top-level processing procedure called from a <task-definition> that specified the <processing-group-identifier> of the <processing-group-specification> on a <call-procedure>.

SEE ALSO

Section 8.1.1, Section 8.2.

8.1.1 <processing-source-language>

NAME

<processing-source-language> — Specifies the programming language used in the top-level processing procedures.

SYNOPSIS

```
<processing-source-language> ::
    SOURCE [LANGUAGE] [IS] <computer-language-literal> ;
```

SYNTAX RULES

1. <computer-language-literal> specifies the computer language used for the top-level processing procedures.

GENERAL RULES"

1. The source language, C or COBOL, must match the programming language used to code the set of top-level processing procedures within the processing group.

SEE ALSO

None.

8.2 <processing-procedure-list>

NAME

<processing-procedure-list> — Specifies the list of top-level processing procedures in a processing group.

SYNOPSIS

```
<processing-procedure-list> ::
    <processing-procedure-interface> ...
```

SYNTAX RULES

None.

GENERAL RULES"

None.

SEE ALSO

Section 8.2.1.

8.2.1 <processing-procedure-interface>

NAME

<processing-procedure-interface> — Specifies the interface for a top-level processing procedure.

SYNOPSIS

```
<processing-procedure-interface> ::
    PROCEDURE <processing-procedure-identifier>
        <processing-argument-specification> ;

<processing-argument-specification> ::
      <nil>
    | USING <processing-interface-argument-list>

<processing-interface-argument-list> ::
    <processing-interface-argument> , ...

<processing-interface-argument> ::
    <data-type-identifier> <processing-interface-argument-passing>

<processing-interface-argument-passing> ::
      <nil>
    | PASSED [AS] <processing-interface-argument-passing-direction>

<processing-interface-argument-passing-direction> ::
      INPUT
    | OUTPUT
    | INOUT

<processing-procedure-identifier> ::
    <external-identifier>
```

SYNTAX RULES

1. <processing-procedure-interface> defines <external-identifier> as a <processing-procedure-identifier> within the processing procedure group being defined. The scope of this definition is the rest of the source file.
2. <processing-procedure-identifier> names the entry point in the processing group for the top-level processing procedure.
3. <data-type-identifier> identifies the data type of an argument for the top-level processing procedure.

GENERAL RULES"

1. If <processing-argument-specification> is <nil>, the default is to pass no arguments.
2. The default <processing-interface-argument-passing-direction> is PASSED AS INOUT.
3. <processing-interface-argument-passing-direction> ensures that:
   — The TP system can properly restrict access to a workspace passed as an argument.
   — Concurrency control is applied to shared workspaces.

4. The arguments in the <processing-procedure-interface> must match the arguments specified on the corresponding <call-procedure> in a <task-definition> in:
   — Number
   — Order
   — Passing characteristics.
5. PASSED AS semantics are:
   — If INPUT is specified the argument is passed to the top-level processing procedure from the <task-definition>.
   — If OUTPUT is specified the argument is passed from the top-level processing procedure to the <task-definition>.
   — If INOUT is specified the argument obeys the semantics of both INPUT and OUTPUT.
6. <processing-procedure-identifier> must match the entry point name in the processing procedure.
7. If a processing procedure modifies an argument passed as INPUT, it is undefined whether the modification is propagated to the caller task or client program.
8. <data-type-identifier> refers to a customer-defined data type. The <field-definition>s within the data type must match in order, type, and size with the <workspace-field>s within the corresponding workspace identifier in the <call-procedure>.

IMPLEMENTOR NOTES

1. An implementation can require that <processing-procedure-identifier> be unique within a TP system.

SEE ALSO

None.

Chapter 9

Task Group Specification

9.1 <task-group-specification>

NAME

<task-group-specification> — Specifies a set of interfaces to tasks.

SYNOPSIS

```
<task-group-specification> ::
    TASK GROUP [SPECIFICATION] <task-group-identifier>
        <task-group-attribute-list>
        <task-interface-list>
    END [TASK] [GROUP] [SPECIFICATION] ;

<task-group-identifier> ::
    <external-identifier>
```

SYNTAX RULES

1. <task-group-specification> defines <external-identifier> as a <task-group-identifier>. The scope of this definition is the rest of this source file.

GENERAL RULES

1. A <task-interface> must be included in the <task-group-specification> for each task that can be called on a TP system.
2. The <task-group-identifier> is used in the <call-task> and <submit-task> and in environmental information to specify attributes for the task group.

SEE ALSO

Section 9.2, Section 9.3.

9.2 <task-group-attribute-list>

NAME

<task-group-attribute-list> — Defines attributes of the task group.

SYNOPSIS

```
<task-group-attribute-list> :: <task-group-attribute> ...

<task-group-attribute> ::
      <task-group-uuid-attribute>
    | <task-group-version-attribute>
```

SYNTAX RULES

1. <task-group-uuid-attribute> must be specified and can be specified only once in a <task-group-specification>.
2. <task-group-version-attribute> is optional and can be specified at most once in a <task-group-specification>.

GENERAL RULES

None.

SEE ALSO

Section 9.2.1, Section 9.2.2.

9.2.1 <task-group-uuid-attribute>

NAME

<task-group-uuid-attribute> — Specifies the UUID for this task group.

SYNOPSIS

```
<task-group-uuid-attribute> ::
    UUID [IS] <uuid-string-literal> ;
```

SYNTAX RULES

None.

GENERAL RULES

1. A UUID uniquely identifies each <task-group-specification>. The UUID makes the task group unique among all callable server TP systems. Callable refers to the ability of the client TP system to send a task call message to the server TP system.
2. A UUID is system-generated in a manner that guarantees uniqueness. See the referenced X/Open DCE RPC specification for more information on the generation of UUIDs.
3. The client TP system sends the UUID as part of the task call message to the server TP system.
4. The UUID is associated with the client and server tasks when translating the tasks into executable entities.
5. The client program receives a specification mismatch exception (ENV-INVOCATION-FAULT) if the UUID in the <task-group-specification> used by the client program for the server task is not the same as the UUID used by the server task.

SEE ALSO

None.

9.2.2 <task-group-version-attribute>

NAME

<task-group-version-attribute> — Specifies a version number for a task group.

SYNOPSIS

```
<task-group-version-attribute> ::
    VERSION [IS] <decimal-literal> ;
```

SYNTAX RULES

1. <decimal-literal> must be specified without a sign.

GENERAL RULES

1. Any numeric characters before the decimal point represent the major version number and any numeric characters following the decimal point represent the minor version number.

2. The major and minor version numbers must each have a value between 0 and 65,535.

3. The defaulting rules for <task-group-version-attribute> are:

   — If only the major version number is specified, the default minor version number is 0.

   — If only the minor version number is specified, the default major version number is 0.

   — If <task-group-version-attribute> is not specified, the default version number is 0.0.

4. The version number is used to identify the version of a <task-group-specification>, in the case where multiple versions of the <task-group-specification> exist.

5. The rules for changing version numbers are:

   — The minor version number must be increased any time an upwardly-compatible change or set of upwardly-compatible changes is made to the <task-group-specification>.

   — The major version number must be increased any time any other change or set of changes is made to the <task-group-specification>.

   — The major version number can never be decreased.

   — The minor version number cannot be decreased without simultaneously increasing the major version number.

6. Definition of an upwardly-compatible change:

   — Adding a <task-interface> to the <task-group-specification>, if and only if the <task-interface> is placed lexically after all existing <task-interface>s in the <task-group-specification>.

   — Adding a <data-type-definition>, provided the new <data-type-definition> is used only by <task-interface>s added at the same time, or later.

7. The client system sends the version number as part of the task call message to the server TP system.

8. The version number is associated with the client program and server task when translating the tasks into executable entities.

9. A client program receives a specification mismatch exception (ENV-INVOCATION-FAULT) if the major version number used by the client program is not the same as the major version number used by the server task.
10. A client program receives a specification mismatch exception (ENV-INVOCATION-FAULT) if the minor version number used by the client program is greater than the minor version number used by the server task.

SEE ALSO

None.

9.3 <task-interface-list>

NAME

<task-interface-list> — Specifies the list of tasks in a task group.

SYNOPSIS

```
<task-interface-list> ::
      <task-interface> ...

<task-interface> ::
      <composable-task-interface>
    | <noncomposable-task-interface>
```

SYNTAX RULES

None.

GENERAL RULES

None.

SEE ALSO

Section 9.3.2, Section 9.3.1.

9.3.1 <composable-task-interface>

NAME

<composable-task-interface> — Specifies the interface to a <composable-task>.

SYNOPSIS

```
<composable-task-interface> ::
    COMPOSABLE TASK <task-identifier>
        <task-interface-argument-list-definition> ;

<task-identifier> ::
    <external-identifier>
```

SYNTAX RULES

1. <composable-task-interface> defines <external-identifier> as a <task-identifier> within the task group being defined. The scope of this definition is the rest of the source file.

GENERAL RULES

1. <composable-task-interface> specifies that the <task-definition> with the specified task identifier must be called within a client program's transaction.

SEE ALSO

Section 9.3.3.

9.3.2 <noncomposable-task-interface>

NAME

<noncomposable-task-interface> — Specifies the interface to a <noncomposable-task>.

SYNOPSIS

```
<noncomposable-task-interface> ::
    TASK <task-identifier>
        <task-interface-argument-list-definition> ;

<task-identifier> ::
    <external-identifier>
```

SYNTAX RULES

1. <noncomposable-task-interface> defines <external-identifier> as a <task-identifier> within the task group being defined. The scope of this definition is the rest of the source file.

GENERAL RULES

1. <noncomposable-task-interface> specifies that the <task-definition> with the specified <task-identifier> must be called outside of the client program's transaction.

SEE ALSO

Section 9.3.3.

9.3.3 <task-interface-argument-list-definition>

NAME

<task-interface-argument-list-definition> — Lists the workspaces passed between a client program and task.

SYNOPSIS

```
<task-interface-argument-list-definition> ::
      <nil>
    | USING <task-interface-argument-list>

<task-interface-argument-list> ::
      <task-interface-argument-definition> , ...

<task-interface-argument-definition> ::
      <data-type-identifier>
          <task-interface-argument-passing>

<task-interface-argument-passing> ::
      <nil>
    | PASSED [AS]
          <task-interface-argument-passing-direction>

<task-interface-argument-passing-direction> ::
      INPUT
    | OUTPUT
    | INOUT
```

SYNTAX RULES

None.

GENERAL RULES

1. The default for passing an argument is PASSED AS INOUT.
2. PASSED AS semantics are:
   — If INPUT is specified, the argument is passed to the called <task-definition> from the client program.
   — If OUTPUT is specified, the argument is passed from the called <task-definition> to the client program.
   — If INOUT is specified, the argument obeys the semantics of both INPUT and OUTPUT.
3. <task-interface-argument-list-definition> supplies the client system with the information required to:
   — Format the argument data being sent to a server task.
   — Interpret the argument data being returned by a server task.
4. The <task-interface-argument-definition>s in a <task-interface-argument-list-definition> for a task in the <task-group-specification> correspond in order to the <task-argument-definition>s in the <task-argument-list-definition> in the <task-definition>. The <task-interface-argument-definition>s and the <task-argument-definition>s must match in:

— number

— structure of the <data-type-definition>s: The <data-type-definition> associated with the <data-type-identifier> in <task-interface-argument-definition> must match the <data-type-definition> associated with the <data-type-identifier> in the <task-argument-definition>. The <field-definition>s within the <data-type-definition> must match in order, type and size.

— passing characteristics.

5. If a server task modifies any arguments passed to it as INPUT, it is undefined whether the modification is propagated to the client program.

SEE ALSO

None.

Chapter 10

Task Definition Language

For each executable statement in this chapter, the General Rules describe the execution of the statement in two parts:

- normal execution flow
- deviation from the normal execution flow.

Because a task can include several levels of nested executable statements, the execution flow description includes the effects of the contained statement execution on the containing statement execution, using one of the following completion reasons:

- Default Flow Control

  Execution without any exception and without any explicit flow control.

- Explicit Flow Control

  As a result of <exit-block>, <exit-task> or <go-to>.

- Exception.

Executable statement completion reasons describe the flow of executable statements that contain other executable statements. General Rules for a containing executable statement include the effect of the completion of the contained executable statement on the execution flow of the containing executable statement.

10.1 <task-definition>

NAME

<task-definition> — Defines a task.

SYNOPSIS

```
<task-definition> ::
      <composable-task>
    | <noncomposable-task>
```

SYNTAX RULES

None.

GENERAL RULES

None.

EXCEPTIONS

None.

SEE ALSO

Section 10.1.2, Section 10.1.1.

10.1.1 <composable-task>

NAME

<composable-task> — Defines a composable task.

SYNOPSIS

```
<composable-task> ::
    COMPOSABLE TASK <task-identifier> IN <task-group-identifier>
        <task-argument-list-definition>
        <task-attribute-list>
        <statement-list>
    END [TASK] ;
```

SYNTAX RULES

1. A <composable-task> cannot include a <call-presentation-procedure> with a non-transactional attribute and a <call-presentation-send-list> or a <call-presentation-receive-list>, and it cannot include a <call-presentation-procedure> with a <call-presentation-transceive-list>.

2. A <composable-task> cannot include a <transaction-block>.

GENERAL RULES

1. The normal execution flow for a <composable-task> is:

   — Initialize the workspaces in <task-argument-list-definition> defined as INPUT and INOUT with data passed from the client program.

   — Execute any <statement-list>; if the <statement-list> terminates with explicit flow control, the explicit flow control is ignored.

   — Return to the client program the workspaces defined in <task-argument-list-definition> as OUTPUT and INOUT.

2. Deviation from the normal execution flow occurs when the <statement-list> completes with an exception with the level set to propagated; if the exception is:

   — Fatal transaction exception, the returned exception type is set to permanent transaction.

   — Transient transaction exception and the task is not restartable, the returned exception type is set to permanent transaction.

   — Transient transaction exception and the task is restartable, the returned exception type is unchanged.

   — Permanent transaction and non-transaction, the returned exception type is unchanged.

3. <task-identifier> is used in a <task-group-specification> to include a task in a task group.

4. A <composable-task> executes within the client program's transaction.

IMPLEMENTOR NOTES

1. An implementation is not required to support a <call-presentation-procedure> with a transactional attribute and a <call-presentation-send-list> in a <composable-task> called by a local client task.

EXCEPTIONS

None.

SEE ALSO

Section 10.1.3, Section 10.2, Section 10.3.

10.1.2 <noncomposable-task>

NAME

<noncomposable-task> — Defines a non-composable task.

SYNOPSIS

```
<noncomposable-task> ::
    TASK <task-identifier> IN <task-group-identifier>
        <task-argument-list-definition>
        <task-attribute-list>
        <statement-list>
    END [TASK] ;
```

SYNTAX RULES

None.

GENERAL RULES

1. The normal execution flow for a <noncomposable-task> is:
   - — Initialize the workspaces in <task-argument-list-definition> defined as INPUT and INOUT with data passed from the client program.
   - — Execute any <statement-list>; if the <statement-list> terminates with explicit flow control, the explicit flow control is ignored.
   - — Return to the client program the workspaces defined in <task-argument-list-definition> as OUTPUT and INOUT.
2. Deviation from the normal execution flow occurs when the <statement-list> completes with an exception; the task completes, returning the exception to the client program, with the exception type set to non-transaction and the level to propagated.
3. <task-identifier> is used in a <task-group-specification> to include a task in a task group.
4. A <noncomposable-task> executes outside of the client program's transaction.

EXCEPTIONS

None.

SEE ALSO

Section 10.1.3, Section 10.2, Section 10.3.

10.1.3 <task-argument-list-definition>

NAME

<task-argument-list-definition> — Specifies an argument list for a task.

SYNOPSIS

```
<task-argument-list-definition> ::
      <nil>
    | USING <task-argument-list> ;

<task-argument-list> ::
    <task-argument-definition> , ...

<task-argument-definition> ::
      <data-type-identifier> <task-argument-passing>
        <task-argument-transactional>
    | <workspace-identifier> [IS] <data-type-identifier>
        <task-argument-passing>
        <task-argument-transactional>

<task-argument-passing> ::
      <nil>
    | PASSED [AS] <task-argument-passing-direction>

<task-argument-passing-direction> ::
      INPUT
    | OUTPUT
    | INOUT

<task-argument-transactional> ::
      <nil>
    | TRANSACTIONAL
    | NOT TRANSACTIONAL

<workspace-identifier> ::
    <internal-identifier>
```

SYNTAX RULES

1. If <workspace-identifier> is used, <task-argument-definition> defines <internal-identifier> as a <workspace-identifier>. Otherwise, <task-argument-definition> defines <data-type-identifier> as a <workspace-identifier>. The scope of this definition is the rest of the <task-definition>.
2. All workspace identifiers defined in <task-argument-definition>s are private workspaces.
3. The default is no arguments.
4. The default for <task-argument-transactional> depends on whether or not the task is restartable.

GENERAL RULES

1. The default for <task-argument-passing> is PASSED AS INOUT.

2. <task-argument-definition>s are passed by reference.
3. <task-argument-passing> semantics are:
   — If INPUT is specified the argument is passed to the <task-definition> from the client program.
   — If OUTPUT is specified the argument is passed from the <task-definition> to the client program.
   — If INOUT is specified the argument obeys the semantics of both INPUT and OUTPUT.
4. The <task-argument-definition>s in the <task-argument-list-definition> of the <task-definition> correspond in order to the <task-interface-argument-definition>s in the <task-interface-argument-list-definition> of the corresponding <task-interface> of the <task-group-specification>. Each <task-interface-argument-definition> and the <task-argument-definition> must match in:
   — Number.
   — Structure of the data type. The <data-type-definition> associated with the <data-type-identifier> in <task-argument-definition> must match the <data-type-definition> associated with the <data-type-identifier> in the <task-interface-argument-definition>. The <field-definition>s within the two <data-type-definition>s must match in order, type and size.
   — Passing characteristics.
5. Workspaces passed as OUTPUT arguments are initialized using the values in the <data-type-definition> for the <task-argument-definition>.
6. Workspaces passed as INPUT or INOUT arguments are initialized using the data passed from the client program.
7. If a task modifies a workspace passed as INPUT, it is undefined whether the modification is propagated to the client program.

EXCEPTIONS

None.

SEE ALSO

Section 5.3.

10.2 <task-attribute-list>

NAME

<task-attribute-list> — Specifies a list of attributes of a task.

SYNOPSIS

```
<task-attribute-list> ::
      <nil>
    | <task-attribute> ...

<task-attribute> ::
      <restartability>
    | <send-display>
    | <workspace-list-definition>
```

SYNTAX RULES

1. At most, one of each <task-attribute> can be specified.

GENERAL RULES

None.

EXCEPTIONS

None.

SEE ALSO

Section 10.2.1, Section 10.2.3, Section 10.2.2.

10.2.1 <restartability>

NAME

<restartability> — Specifies treatment of transient transaction exceptions and sets task defaults.

SYNOPSIS

```
<restartability> ::
      NOT RESTARTABLE ;
    | RESTARTABLE ;
```

SYNTAX RULES

None.

GENERAL RULES

1. NOT RESTARTABLE is the default.
2. A non-restartable task is created using NOT RESTARTABLE. The defaults for a non-restartable task are:
   — <submit-task> and <cancel-submit> execute outside of the task's transaction.
   — <call-presentation-procedure> statements with <call-presentation-send-list> and <call-presentation-receive-list> are non-transactional.
   — Workspaces declared in <workspace-list-definition> and <task-argument-list-definition> are private and non-transactional.
3. A restartable task is created using RESTARTABLE. The defaults for a restartable task are:
   — <submit-task> and <cancel-submit> execute within the task's transaction
   — <call-presentation-procedure> statements with <call-presentation-send-list> and <call-presentation-receive-list> are transactional.
   — Workspaces declared in <workspace-list-definition> and <task-argument-list-definition> are private and transactional.

CONFORMANCE NOTES

1. <restart-transaction> is not required for Support Levels 1 and 2.

EXCEPTIONS

None.

SEE ALSO

None.

10.2.2 <send-display>

NAME

<send-display> — Specifies a task can perform only <call-presentation-procedure>s with a <call-presentation-send-list>.

SYNOPSIS

```
<send-display> ::
      SEND DISPLAY ;
    | SEND DISPLAY [IS] <os-name-value> ;
```

SYNTAX RULES

1. If <os-name-value> is specified using an <s-text-workspace-field>, the field must be defined within a workspace passed to the task in <task-argument-list-definition>s as INPUT or INOUT.
2. <os-name-value> contains a display name.

GENERAL RULES

1. The default display name is the name of the display passed from the client program.
2. The display name must be defined on the TP system where the task executes.
3. A display is defined by the TP system and is assigned to the task before the first <statement> is executed.
4. When used, <send-display> can override the default display passed from the client. The display passed from the client is used when <os-name-value> is omitted.
5. The display name is set only once, when the task is invoked, and is never updated during task execution, even when the <s-text-workspace-field> used to specify <os-name-value> is defined as INOUT.
6. When <os-name-value> is specified, the following fields are set in the SYSTEM-INFO-WORKSPACE:

   DISPLAY-NAME
   Set to the value of <os-name-value>.

   DISPLAY-TP-SYSTEM
   Set to a destination for a TP system on which the display is available to the task.
7. <send-display> overrides any language passed from the client.
8. The TP system specifies the number of tasks that can be concurrently assigned to a display.
9. Once a task has started using a display, another task cannot use the display in such a way as to make the display unavailable to the first task.

CONFORMANCE NOTES

1. <send-display> is not required for Profile A. It is optional for Profile B.

EXCEPTIONS

None.

SEE ALSO

None.

10.2.3 <workspace-list-definition>

NAME

<workspace-list-definition> — Specifies a list of workspace definitions that define working storage for a task.

SYNOPSIS

```
<workspace-list-definition> ::
      WORKSPACE [IS] <workspace-definition-list> ;
    | WORKSPACES [ARE] <workspace-definition-list> ;

<workspace-definition-list> ::
    <workspace-definition> , ...

<workspace-definition> ::
      <data-type-identifier> <workspace-attribute-list>
    | <workspace-identifier> [IS] <data-type-identifier>
        <workspace-attribute-list>

<workspace-attribute-list> ::
      <nil>
    | <workspace-attribute> ...

<workspace-attribute> ::
      NOT TRANSACTIONAL
    | TRANSACTIONAL
    | SHARED
    | SHARED READ
    | SHARED UPDATE
    | PRIVATE

<workspace-identifier> ::
    <internal-identifier>
```

SYNTAX RULES

1. If <workspace-identifier> is used, <workspace-definition> defines <internal-identifier> as a <workspace-identifier>. Otherwise, <data-type-identifier> is defined as a <workspace-identifier>.
2. SHARED cannot be specified with NOT TRANSACTIONAL.
3. TRANSACTIONAL cannot be specified with NOT TRANSACTIONAL.
4. SHARED cannot be specified with PRIVATE.
5. At most, one of each of the following can be specified in each <workspace-definition>:
   — TRANSACTIONAL or NOT TRANSACTIONAL
   — SHARED or PRIVATE.
6. Workspace identifiers for private workspaces must be unique within a <task-definition>.
7. Workspace identifiers for shared workspaces must be unique within the task group.
8. A workspace identifier cannot be used for both a private workspace and a shared workspace within a <task-definition>.

GENERAL RULES

1. SHARED specifies that the workspace is a shared workspace.
2. PRIVATE specifies that the workspace is a private workspace.
3. The defaults for a non-restartable task are NOT TRANSACTIONAL and PRIVATE.
4. The defaults for a restartable task are TRANSACTIONAL and PRIVATE.
5. The default access mode for a shared workspace is UPDATE.
6. A workspace is defined by a task.
7. A private workspace is initialized to the value specified in the <data-type-definition> for the <data-type-identifier> before the first access by the task invocation and after system startup.
8. A shared workspace is initialized to the value specified in the <data-type-definition> for the <data-type-identifier> before the first access by the first task execution that references the shared workspace and after system startup.
9. If READ is specified, the TP system assumes that the task only reads the shared workspace. If the same task updates the shared workspace, the results are undefined.
10. If UPDATE is specified, the TP system assumes that the task updates the shared workspace.
11. A system workspace is a private workspace defined by the system. The following system workspaces are available to each <task-definition>:

    — EXCEPTION-INFO-WORKSPACE

    Contains information about the current exception. The <data-type-definition> is:

    ```
    TYPE EXINFO IS RECORD
        EXCEPTION-CLASS INTEGER;
        EXCEPTION-CODE INTEGER;
        EXCEPTION-CODE-GROUP UUID;
        EXCEPTION-LEVEL INTEGER;
        EXCEPTION-SOURCE INTEGER;
        EXCEPTION-PROCEDURE TEXT CHARACTER SET
            SIMPLE-LATIN SIZE 32.
        EXCEPTION-PROCEDURE-GROUP TEXT CHARACTER SET
            SIMPLE-LATIN SIZE 32.
    END RECORD;
    ```

    — EXCEPTION-CLASS is the value of the exception class raised. See Appendix C on page 489 for the list of exception classes.

    — EXCEPTION-CODE is a field that contains the value of the exception code raised. The values of this field can be:

    0 The application (EXCEPTION-SOURCE=1) raised an exception using an exception class.

    <message-number>
    The application (EXCEPTION-SOURCE=1) raised an exception using an exception code.

    implementation-defined value
    The TP system (EXCEPTION-SOURCE=0) raised an exception.

— EXCEPTION-CODE-GROUP is the field that contains the UUID of the exception group raised. For application-defined exceptions, this is the UUID defined in the <message-definition>. If the <message-definition> does not define a value, this is all zeros. For system-defined exceptions, this is the implementation-defined UUID for the system exception.

— EXCEPTION-LEVEL is a field indication whether the exception was raised by the current task execution or by a called task or procedure execution. The values of this field can be:

0 The exception was raised by the current task execution.

1 The exception was propagated to the current task execution.

— EXCEPTION-SOURCE is a field indicating the source of the exception. The source can be:

0 The exception was raised by the TP system.

1 The exception was raised by the application in a task definition using a <restart-transaction> or <raise-exception>, in a top-level processing procedure, or in a presentation procedure.

— EXCEPTION-PROCEDURE is a field that contains the name of the procedure in which the exception was detected. If the exception was raised by the TP system (EXCEPTION-SOURCE=0), this is the name of the task, presentation procedure or processing procedure in which the exception was raised. If the exception was raised in the task, this is the name of the task in which the exception was raised. If the exception was raised by the application (EXCEPTION-SOURCE=1) in a processing or presentation procedure, and the application did not set the exception procedure field in the exception variable, this contains the name of the processing or presentation procedure called by the task. Otherwise, it contains the text string set by the processing or presentation procedure.

— EXCEPTION-PROCEDURE-GROUP is a field that contains the name of the procedure group in which the exception was detected. If the exception was raised by the TP system (EXCEPTION-SOURCE=0), this is the name of the task group, presentation group or processing group in which the exception was raised. If the exception was raised in the task, this is the name of the task group of the task in which the exception was raised. If the exception was raised by the application (EXCEPTION-SOURCE=1) in a processing or presentation procedure, and the application did not set the exception procedure group field of the exception variable, this contains the name of the processing or presentation group of the processing or presentation procedure called by the task. Otherwise, this contains the text string set by the processing or presentation procedure.

— SYSTEM-INFO-WORKSPACE

Contains information about the task execution.

The <data-type-definition> is:

```
TYPE SYSINFO IS RECORD
    TASK-NAME TEXT SIZE 32;
    TASK-GROUP-NAME TEXT SIZE 32;
    TP-SYSTEM-NAME TEXT SIZE 256;
    DISPLAY-NAME TEXT SIZE 256;
    DISPLAY-TP-SYSTEM TEXT SIZE 256;
    ACTIVATION-TIME TEXT SIZE 14;
END RECORD;
```

- TASK-NAME contains the name of the task being invoked.
- TASK-GROUP-NAME contains the name of the task group.
- TP-SYSTEM-NAME contains the name of the TP system executing the <task-definition>, derived using the TP system name.
- DISPLAY-NAME contains the symbolic name for the task's display.
- DISPLAY-TP-SYSTEM contains the TP system name for the display.
- ACTIVATION-TIME contains the time at which task execution began. The format is absolute time, or YYYYMMDDHHMMSS, where:
  - YYYY is the year from 0000 to 9999
  - MM is the month from 1 to 12
  - DD is the day from 1 to 31
  - HH is the hour from 0 to 23
  - MM is the minute from 0 to 59
  - SS is the second from 0 to 59.

12. The following rules apply to system workspaces:
    - A system workspace is supplied by the TP system.
    - The values of the SYSTEM-INFO-WORKSPACE are initialized by the TP system before the task begins execution.
    - Any data stored in TEXT fields in the SYSTEM-INFO-WORKSPACE or in the EXCEPTION-INFO-WORKSPACE must be padded to the right with spaces to the length of the TEXT field.
    - The values of the EXCEPTION-INFO-WORKSPACE are set before the execution of an <exception-handler> and can only be referenced within an <exception-handler>.
    - A system workspace is a non-transactional private workspace resource.
    - The effect of a write to a system workspace by a task is undefined.
    - The effect of a write to a system workspace by a processing procedure is undefined.
13. Shared workspace context is shared between all tasks that execute within the same transaction within the same task group on the same TP system. Shared workspace context consists of:

— Access to previously-modified shared workspaces within the current transaction.

— Access to previously-read shared workspaces within the current transaction.

14. For a composable task, the semantics of a transactional private workspace and a non-transactional private workspace are the same.

IMPLEMENTOR NOTES

1. When a Task A that specifies READ access to a shared workspace directly or indirectly calls a Task B that specifies UPDATE access to the same shared workspace within the same transaction, the value of the workspace within Task A before calling Task B can be different from the initial value of the workspace within Task B. After Task B returns to Task A, the value of the workspace within Task A is consistent with the value of the workspace within Task B.

CONFORMANCE NOTES

1. Shared workspaces are not required for Support Levels 1 and 2.
2. Transactional private workspaces are not required for Support Levels 1 and 2.

EXCEPTIONS

None.

NOTES

1. A workspace identifier cannot be defined in both <task-argument-list-definition> and <workspace-list-definition>.
2. A transaction can include tasks that specify READ access to a shared workspace and tasks that specify UPDATE access to the same shared workspace. These tasks can be executed in any order. Any updates made by a task are visible to all tasks that belong to the same transaction.

SEE ALSO

None.

10.3 <statement-list>

NAME

<statement-list> — Contains a set of <statement>s.

SYNOPSIS

```
<statement-list> ::
      <nil>
    | <statement> ...

<statement> ::
    <statement-label> <statement-type> ;

<statement-label> ::
      <nil>
    | <label-identifier>:

<statement-type> ::
      <audit>
    | <assignment>
    | <call-presentation-procedure>
    | <call-procedure>
    | <call-task>
    | <cancel-submit>
    | <concurrent-block>
    | <control-field>
    | <dequeue-record>
    | <enqueue-record>
    | <exit-block>
    | <exit-task>
    | <get-message>
    | <go-to>
    | <if>
    | <raise-exception>
    | <read-queue-record>
    | <reraise-exception>
    | <restart-transaction>
    | <select-first>
    | <statement-block>
    | <submit-task>
    | <transaction-block>
    | <while>

<label-identifier> ::
    <internal-identifier>
```

SYNTAX RULES

1. A <statement-label> with an <internal-identifier> defines <internal-identifier> as a <label-identifier>. The scope of the definition is the <task-definition>.

GENERAL RULES

1. The normal execution flow for a <statement-list> is to execute all of the <statement>s in the <statement-list> sequentially and complete with default flow control.
2. Deviations from the normal execution flow occur when a <statement> in the <statement-list> completes with:
   - Explicit <exit-block> or <exit-task> flow control; the <statement-list> completes with the same explicit flow control with which the <statement> completed.
   - Explicit <go-to> flow control if the target of the <go-to> is in the <statement-list> and the normal execution flow resumes with the target <statement>; otherwise the <statement-list> completes with explicit <go-to> flow control.
   - Exception; the <statement-list> completes with the exception raised by the <statement>.

EXCEPTIONS

None.

SEE ALSO

Section 10.3.25, Section 10.3.23, Section 10.3.7, Section 10.3.17, Section 10.3.22, Section 10.3.8, Section 10.3.27, Section 10.3.4, Section 10.3.5, Section 10.3.24, Section 10.3.6, Section 10.3.10, Section 10.3.9, Section 10.3.19, Section 10.3.3, Section 10.3.1, Section 10.3.2, Section 10.3.15, Section 10.3.20, Section 10.3.21, Section 10.3.18, Section 10.3.13, Section 10.3.14, Section 10.3.16.

10.3.1 <assignment>

NAME

<assignment> — Manipulates workspace field contents.

SYNOPSIS

```
<assignment> ::
    <assignment-workspace-field-list> = <expression>

<assignment-workspace-field-list> ::  <workspace-field> ,...

<expression> ::
      <integer-expression>
    | <message-identifier>
    | <message-group-identifier>
    | <class-identifier>
    | <string-literal>
    | <uuid-string-literal>
    | <decimal-literal>
    | <octet-workspace-field>
    | <integer-workspace-field>
    | <text-workspace-field>
    | <decimal-workspace-field>
    | <record-workspace-field>
    | <array-workspace-field>
    | <uuid-workspace-field>
```

SYNTAX RULES

1. If <integer-expression>, <message-identifier> or <class-identifier> is specified, the <workspace-field> must be an <integer-workspace-field>.
2. If <message-group-identifier> or <uuid-string-literal> is specified, the <workspace-field> must be a <uuid-workspace-field>.
3. An <integer-literal> or an <integer-workspace-field> can be moved to an <integer-workspace-field>.
4. An <octet-workspace-field> can be moved to an <octet-workspace-field>.
5. A <string-literal> or <text-workspace-field> can be moved to a <text-workspace-field>.
6. A <decimal-literal> or <decimal-workspace-field> can be moved to a <decimal-workspace-field>.
7. An array can be moved to an <array-workspace-field>. The data types of the elements in each array must match. The source and destination arrays must be the same size. For varying-length arrays, the minimum sizes and maximum sizes of the source and destination arrays must be the same.
8. A record can be moved to a <record-workspace-field>. All fields in each record must match in order, data type, and size.
9. When manipulating <text-workspace-field>s, the character set used for the <expression> must match the character set used for the <workspace-field>.
10. A <uuid-workspace-field>, <message-group-identifier>, or <uuid-string-literal> can be moved to a <uuid-workspace-field>.

11. When <assignment> is used in a <noncomposable-task> outside of a <transaction-block>, it must not reference any transactional workspace fields.

GENERAL RULES

1. The normal execution flow for an <assignment> statement is to evaluate <expression>, and move that value to one or more <workspace-field>s and to complete with default flow control.
2. Deviations from the normal execution flow occur when one of the conditions listed under EXCEPTIONS occurs; the <assignment> statement completes with the exception associated with that condition, and the value of <workspace-field> is undefined.
3. The result of the <expression> is stored as the new value for the <workspace-field> in the <assignment-workspace-field-list>.

EXCEPTIONS

The following exception class can be generated:

1. AP-EXECUTION-FAULT

   An AP-EXECUTION-FAULT class exception is generated for the following exceptions:

   — During the <assignment> operation, an integer value exceeded the data type INTEGER range.

   — During the <assignment> operation, an attempt was made to divide by zero.

   — The subscript used to reference an element in a fixed-length array is invalid (referring to A(n), where n < 1 or n > the maximum size of array).

   — The subscript used to reference an element in a varying-length array is invalid (referring to A(n), where n < 1 or n > the value of the DEPENDING ON field).

   — The value of the DEPENDING ON field used when referencing an element of a varying-length array is invalid (the value of the DEPENDING ON field < the minimum size of array or the value of the DEPENDING ON field > the maximum size of array).

   — The value of the DEPENDING ON field was invalid when referencing a <workspace-field> containing a varying-length array.

SEE ALSO

Section 4.16.

10.3.2 <audit>

NAME

<audit> — Writes information to the application audit log.

SYNOPSIS

```
<audit> ::
    AUDIT <audit-information-list>

<audit-information-list> ::
    <audit-information> , ...

<audit-information> ::
      <workspace-field>
    | <string-literal>
```

SYNTAX RULES

1. When <audit> is used in a <noncomposable-task> outside of a <transaction-block>, it must not reference any transactional workspace fields.

GENERAL RULES

1. The normal execution flow for an <audit> statement is to write to the audit log and to complete with default flow control.
2. Deviations from the normal execution flow occur when one of the conditions listed under EXCEPTIONS occurs; the <audit> statement completes with the exception associated with that condition, and whether or not a write to the audit log is performed is undefined.
3. <audit-information-list> is written to the audit log using a non-transactional operation.
4. The application audit log used to record <audit-information-list> is maintained separately from the audit log used to record TP system-generated events.
5. The format of the the audit log is implementation-specific. However, each implementation must provide a management utility to convert the audit log from the implementation-specific format to a common record format (see Appendix B on page 487).

CONFORMANCE NOTES

1. <audit> statement is not required for Support Level 1.
2. A TP system can have 0 or 1 application audit log. When a TP system has an audit log, it has only one.

EXCEPTIONS

The following exception classes can be generated:

1. AP-EXECUTION-FAULT

   An AP-EXECUTION-FAULT class exception is generated for the following exceptions:

   — The subscript used to reference an element in a fixed-length array is invalid (referring to A(n), where n < 1 or n > maximum size of array).

   — The subscript used to reference an element in a varying-length array is invalid (referring to A(n), where n < 1 or n > the value of the depending on field).

— The value of the depending on field used when referencing an element of a varying-length array is invalid (the value of the depending on field < minimum size of array or the value of the depending on field > maximum size of array).

— The value of the DEPENDING ON field was invalid when referencing a <workspace-field> containing a varying-length array.

2. ENV-EXECUTION-FAULT

   An ENV-EXECUTION-FAULT exception class can be generated for the following exception:

   — The TP system in which the task is executing is unable to write the <audit-information-list> onto the audit log.

SEE ALSO

None.

10.3.3 <call-presentation-procedure>

NAME

<call-presentation-procedure> — Calls a presentation procedure.

SYNOPSIS

```
<call-presentation-procedure> ::
    CALL PRESENTATION <presentation-procedure-identifier>
        IN <presentation-group-identifier>
        <call-presentation-with>
        <call-presentation-lists>

<call-presentation-with> ::
      <nil>
    | WITH <call-presentation-attribute>

<call-presentation-attribute> ::
      NO TRANSACTIONAL [WORK]
    | TRANSACTIONAL [WORK]
    | BROADCAST LIST <os-name-value>

<call-presentation-lists> ::
      <call-presentation-send-list>
    | <call-presentation-receive-list>
    | <call-presentation-transceive-list>

<call-presentation-send-list> ::
    SENDING <call-presentation-workspace-list>

<call-presentation-receive-list> ::
    RECEIVING <call-presentation-workspace-list>

<call-presentation-transceive-list> ::
    SENDING <call-presentation-workspace-list-1>
    RECEIVING <call-presentation-workspace-list-2>

<call-presentation-workspace-list> ::
      <nil>
    | <workspace-identifier> , ...
```

SYNTAX RULES

1. The default <call-presentation-with> for a non-restartable task is NO TRANSACTIONAL WORK.
2. The default <call-presentation-with> for a restartable task is TRANSACTIONAL WORK.
3. A <call-presentation-procedure> in a <composable-task> must specify <call-presentation-send-list> and must be transactional.
4. If BROADCAST LIST is specified, <call-presentation-send-list> must be specified.
5. The value of <os-name-value> is a broadcast list name.
6. If <call-presentation-transceive-list> is specified, the <call-presentation-procedure> must either specify or default to WITH NO TRANSACTIONAL WORK.

7. When <call-presentation-procedure> is used in a <noncomposable-task>, it must be inside of a <transaction-block>.
8. A transactional receive must be the first <statement> in the <statement-list> of a <transaction-block>.

GENERAL RULES

1. A <call-presentation> statement can be classified as being one of the following:
   — A transactional send, which is a <call-presentation-procedure> statement that specifies a <call-presentation-send-list> and either specifies or defaults to WITH TRANSACTIONAL WORK.
   — A non-transactional send, which is a <call-presentation-procedure> statement that specifies a <call-presentation-send-list>, does not specify <call-presentation-broadcast> and either specifies or defaults to WITH NO TRANSACTIONAL WORK.
   — A broadcast send, which is a <call-presentation-procedure> statement that specifies a <call-presentation-send-list> and BROADCAST LIST. A broadcast send is non-transactional.
   — A transactional receive, which is a <call-presentation-procedure> statement that specifies a <call-presentation-receive-list> and either specifies or defaults to WITH TRANSACTIONAL WORK.
   — A non-transactional receive, which is a <call-presentation-procedure> statement that specifies a <call-presentation-receive-list> and either specifies or defaults to WITH NO TRANSACTIONAL WORK.
   — A transceive, which is a <call-presentation-procedure> statement that specifies a <call-presentation-transceive-list>.
2. The normal execution flow for a transactional send is to store the value of the workspaces for the presentation procedure in a transactional, durable store, and to complete with default flow control.
3. The normal execution flow for a non-transactional send is to execute the presentation procedure, providing the workspaces and to complete with default flow control.
4. The normal execution flow for a broadcast send is to store the value of the workspaces for the presentation procedure and to complete with default flow control.
5. The normal execution flow for a transactional receive is:
   — If a transaction has not been restarted, execute the presentation procedure and store the output of the presentation procedure.
   — If a transaction has been restarted, use the data stored on the first execution of the transaction.
   — Complete with default flow control.
6. The normal execution flow for a non-transactional receive is to execute the presentation procedure and complete with default flow control.
7. The normal execution flow for a transceive is to execute the presentation procedure and complete with default flow control.
8. Deviations from the normal execution flow occur when one of the conditions listed under EXCEPTIONS occurs; the <call-presentation-procedure> statement completes

with the exception associated with the condition, the values of workspaces in the RECEIVING phrase are undefined.

9. The <presentation-procedure-identifier> identifies the presentation procedure. If a customer-written presentation procedure is used, this is the name of the presentation procedure to be executed. If a vendor-supplied forms system is used, the name is interpreted by the forms system.

10. The <presentation-group-identifier> identifies the presentation group for the presentation procedure. If a customer-written presentation procedure is used, this is the name of the presentation group as specified in the <presentation-group-specification>. If a vendor-supplied forms system is used, the name is interpreted by the forms system.

11. If the same <workspace-identifier> is specified more than once in the RECEIVING phrase the results are undefined.

12. The contents of <workspace-identifier>s in the SENDING phrase are sent to the presentation procedure.

13. The contents of <workspace-identifier>s in the RECEIVING phrase are received from the presentation procedure.

14. The following general rules apply to transactional sends:

    — The presentation procedure for a transactional send is executed after the transaction commits.

    — If the transaction is rolled back, the presentation procedure is not executed.

    — When the transaction commits, the workspaces are durably stored by the TP system.

    — Any exceptions generated by the presentation procedure are handled by the TP system.

    — The TP system provides a notification and retrieval mechanism for transactional send presentation procedures that cannot be invoked successfully. The retrieval mechanism makes available the data that was durably stored as part of the commit procedure.

15. The following general rules apply to broadcast sends:

    — After the <call-presentation-procedure> statement has completed execution, the TP system asynchronously executes a presentation procedure for each display in the broadcast list. These presentation procedures are called broadcast presentation procedures.

    — Any exception generated by a broadcast presentation procedure is handled by the TP system.

    — The TP system is not required to handle the problem of a display being unavailable for a broadcast send procedure.

16. If the <call-presentation-broadcast> is not specified, a broadcast operation is not done and the task's display is used.

17. The value of the <os-name-value> uniquely identifies a broadcast list that specifies one or more display names for which to execute the broadcast send presentation procedure.

CONFORMANCE NOTES

1. Broadcast is not required for Profile A. It is an optional feature for Profile B.

EXCEPTIONS

No exceptions can be generated from the execution of a transactional <call-presentation-procedure> with a <call-presentation-send-list>. Any exception is handled by the TP system using an implementation-specific mechanism. An exception is not returned to the <call-presentation-procedure> statement.

The following exceptions can be generated for other types of <call-presentation-procedure> statements:

1. AP-INVOCATION-FAULT

   An AP-INVOCATION-FAULT class exception is generated for the following exception:

   — A specification mismatch occurred on a <call-presentation-procedure> statement for a customer-written send, receive, or transceive presentation procedure. The interface in the <presentation-group-specification> that was used when processing the calling <task-definition> into an executable task is incompatible with the interface in the called presentation procedure.

2. ENV-INVOCATION-FAULT

   An ENV-INVOCATION-FAULT class exception is generated for the following exceptions:

   — The presentation group specified by the <presentation-group-identifier> in the <call-presentation-procedure> statement is not defined within the environmental information of the TP system in which the task is executing.

   — The broadcast list specified by the <call-presentation-broadcast> is not defined within the environmental information of the TP system in which the task is executing.

3. ENV-INVOCATION-ERROR

   An ENV-INVOCATION-ERROR class exception is generated for the following exceptions:

   — The presentation group specified by the <presentation-group-identifier> in the <call-presentation-procedure> statement is not available due to a system failure.

   — The display used by the <call-presentation-procedure> statement is off line and not currently available for use.

4. AP-EXECUTION-FAULT

   An AP-EXECUTION-FAULT class exception is generated for the following exceptions:

   — The presentation procedure executed by the <call-presentation-procedure> statement caused the TP system to generate an exception. The exception was due to a access violation, an unhandled divide by zero, or other unhandled programming or logic error caused by the execution of the presentation procedure.

   — The presentation procedure executed by the <call-presentation-procedure> statement caused the TP system to generate an exception. The exception was due to a display error condition, caused when the presentation procedure tried to access the display.

- The subscript used to reference an element in a fixed-length array is invalid (referring to A(n), where n < 1 or n > maximum size of array).
- The subscript used to reference an element in a varying-length array is invalid (referring to A(n), where n < 1 or n > the value of the depending on field).
- The value of the depending on field used when referencing an element of a varying-length array is invalid (the value of the depending on field < minimum size of array or the value of the depending on field > maximum size of array).
- The value of the DEPENDING ON field was invalid when referencing a <workspace-identifier> or <workspace-field> containing a varying-length array.

5. ENV-EXECUTION-ERROR

   An ENV-EXECUTION-ERROR class exception is generated for the following exception:

   - The presentation procedure executed by the <call-presentation-procedure> statement could not complete due to a system failure of the TP system in which the task is executing.

6. REQUEST-TIMEOUT-ERROR

   A REQUEST-TIMEOUT-ERROR class exception is generated for the following exception:

   - The presentation procedure time limit environmental attribute was exceed.

7. AP-RESPONSE-FAULT

   An AP-RESPONSE-FAULT class exception is generated for the following exception:

   - The results of the <call-presentation-procedure> statement could not be returned to the task that called the presentation procedure due to a syntax error.

8. Any exception returned by the presentation procedure.

SEE ALSO

None.

10.3.4 <call-procedure>

NAME

<call-procedure> — Calls a procedure in a processing group.

SYNOPSIS

```
<call-procedure> ::
    CALL PROCEDURE <processing-procedure-identifier>
        IN <processing-group-identifier>
        <call-procedure-using>

<call-procedure-using> ::
      <nil>
    | USING <call-procedure-argument-list>

<call-procedure-argument-list> ::
    <workspace-identifier> , ...
```

SYNTAX RULES

1. Private and shared workspaces can be passed.
2. When <call-procedure> is used in a <noncomposable-task>, it must be inside of a <transaction-block>.

GENERAL RULES

1. The normal execution flow for a <call-procedure> is to call the top-level processing procedure, passing the arguments defined in the <processing-procedure> as INPUT or INOUT; when the procedure completes, to return the arguments defined as OUTPUT or INOUT to the corresponding workspaces and complete with default flow control.
2. Deviations from the normal execution flow occur when one of the conditions listed under EXCEPTIONS occurs; the <call-procedure> completes with the exception associated with the condition and the values of the workspaces corresponding to the arguments defined in the <processing-procedure> as OUTPUT or INOUT are undefined.
3. The arguments in the USING phrase must match the <processing-procedure-interface> arguments specified in the <processing-group-specification> in number, order and data types and lengths of fields within the workspaces.
4. The top-level processing procedure specified by the <processing-procedure-identifier> in the <call-procedure> participates in the same transaction as the calling task.
5. For the called top-level processing procedure, the semantic is pass by reference. The TP system can implement either a copy-in, copy-out or pass by reference mechanism for passing arguments.
6. If a procedure modifies any arguments passed to it as INPUT, it is undefined whether the modification is propagated to the caller task or client program.
7. If the same <workspace-identifier> is specified in the USING phrase more than once as an INOUT or OUTPUT argument, the results are undefined.

CONFORMANCE NOTES

1. Indexed file access is not required for Support Level 1.
2. Relative file access is not required for Support Level 1.

EXCEPTIONS

The following exception classes can be generated:

1. AP-INVOCATION-FAULT

   An AP-INVOCATION-FAULT class exception is generated for the following exception:

   — A specification mismatch occurred on a <call-procedure>. The interface in the <processing-group-specification> for the called procedure that was used when processing the calling <task-definition> into an executable task is incompatible with the interface in the called top-level processing procedure.

2. ENV-INVOCATION-FAULT

   An ENV-INVOCATION-FAULT class exception is generated for the following exception:

   — The processing group specified by the <processing-group-identifier> in the <call-procedure> is not defined within the environmental information of the TP system in which the task is executing.

3. ENV-INVOCATION-ERROR

   An ENV-INVOCATION-ERROR class exception is generated for the following exceptions:

   — The top-level processing procedure specified by the <processing-procedure-identifier> in the <call-procedure> is disabled.

   — The processing group specified by the <processing-group-identifier> in the <call-procedure> is not available due to a system failure.

4. AP-EXECUTION-FAULT

   An AP-EXECUTION-FAULT class exception is generated for the following exception:

   — The top-level processing procedure specified by the <processing-procedure-identifier> in the <call-procedure> or a nested processing procedure called before returning to the top-level processing procedure caused the TP system to generate an exception. The exception was due to an access violation, an unhandled divide by zero or other unhandled programming, or logic errors caused by the execution of the processing procedure.

   — The value of the DEPENDING ON field was invalid when referencing a <workspace-identifier>.

5. ENV-EXECUTION-ERROR

   An ENV-EXECUTION-ERROR class exception is generated for the following exception:

   — The top-level processing procedure specified by the <processing-procedure-identifier> in the <call-procedure> could not complete due to a system failure on the TP system in which the task is executing.

6. AP-RESPONSE-FAULT

   An AP-RESPONSE-FAULT class exception is generated for the following exception:

   — The results of the <call-procedure> could not be returned to the task that called the top-level processing procedure due to a syntax error.

7. Any exception returned by the top-level processing procedure.

IMPLEMENTOR NOTES

1. An AP-EXECUTION-FAULT class is generated when an implementation prohibits a procedure from calling itself directly or indirectly.

SEE ALSO

None.

10.3.5 <call-task>

NAME

<call-task> — Calls another task.

SYNOPSIS

```
<call-task> ::
    CALL TASK <task-identifier>
        <call-task-parts-list>
        <call-task-with>
        <call-task-using>

<call-task-with>   ::
      <nil>
    | [WITH] INDEPENDENT [WORK]
    | [WITH] DEPENDENT [WORK]

<call-task-parts-list> ::
      <nil>
    | <call-task-part> ...

<call-task-part> ::
      <call-task-in>
    | <call-task-at>

<call-task-in> ::
    IN <task-group-identifier>

<call-task-at> ::
    AT <os-name-value>

<call-task-using> ::
      <nil>
    | USING <call-task-argument-list>

<call-task-argument-list>::
    <workspace-identifier>, ...
```

SYNTAX RULES

1. <call-task-in> and <call-task-at> can each appear at most once in a <call-task>.
2. When the called task is in another task group, <call-task-in> is required.
3. If the called task is defined in the called task's <task-group-specification> using a <noncomposable-task-interface>, the default <call-task-with> is WITH INDEPENDENT WORK.
4. If the called task is defined in the called task's <task-group-specification> using a <composable-task-interface>, the default <call-task-with> is WITH DEPENDENT WORK.
5. DEPENDENT WORK must not be specified when calling a <noncomposable-task>.
6. INDEPENDENT WORK must not be specified when calling a <composable-task>.

7. When <call-task-at> is specified, <call-task-in> is required.
8. Private and shared workspaces can be passed.
9. When <call-task> is used in a <noncomposable task>, it must be inside of a <transaction-block>.
10. <os-name-value> contains a destination name.

GENERAL RULES

1. DEPENDENT WORK specifies that the task specified in the <call-task> participates in the same transaction as the calling task.
2. INDEPENDENT WORK specifies that the task specified in the <call-task> does not participate in the same transaction as the calling task.
3. The normal execution flow for a <call-task> is to call the task passing the arguments defined in the <task-interface> as INPUT or INOUT; when the server task completes, to return the arguments defined as OUTPUT or INOUT to the corresponding workspaces and complete with default flow control.
4. Deviations from the normal execution flow occur when one of the conditions listed under EXCEPTIONS occurs; the <call-task> completes with the exception associated with the condition and the values of the workspaces corresponding to the arguments defined in the <task-interface> as OUTPUT or INOUT are undefined.
5. If no <task-group-identifier> is specified, the <task-group-identifier> of the <noncomposable-task> or <composable-task> is used when translating the <task-definition> into a executable task.
6. The <data-type-definition>s associated with the <workspace-identifier>s named in the <call-argument-list> in the USING clause of the calling task correspond in order to the <data-type-definitions> associated with the <task-interface-argument-definition>s in the USING clause of <task-interface-argument-list-definition> for the task group specification for the called task. The corresponding <data-type-definition>s must match in number, data types, and length of fields.
7. If both <call-task-at> and <call-task-in> are omitted, the called task executes on the same TP system and in the same task group as the calling task. The client TP system does not need to use a destination to determine the server TP system because the server TP system is the same as the client TP system.
8. If <call-task-at> is omitted and <call-task-in> is specified, the client TP system uses a destination to find the server TP system. The client TP system can choose to use either a destination with the same name as the server task group or a default destination. In either case, the task group entry within the destination that matches the server task group identification and has a compatible version number is used to determine the server TP system. An implementation can choose to use either the task group name or UUID as the task group identification.
9. If <call-task-at> is specified, the client TP system uses the value of <os-name-value> to map to a destination.
10. For the called task, the semantics is pass by reference. A TP system can implement either a copy-in, copy-out or a pass by reference mechanism for passing arguments.
11. If a server task modifies any arguments passed to it as INPUT, it is undefined whether the modification is propagated to the caller task or client program.

12. If the same <workspace-identifier> is specified in the USING phrase more than once as an INOUT or OUTPUT argument, the results are undefined.

CONFORMANCE NOTES

1. Transactional communication is not required for Support Levels 1, 2 and 3.

EXCEPTIONS

The following exception classes can be generated:

1. AP-INVOCATION-FAULT

   An AP-INVOCATION-FAULT class exception is generated for the following exception:

   — A specification mismatch occurred on a <call-task>. The <task-group-specification> that was used when translating the calling <task-definition> into an executable task is incompatible with the <task-group-specification> that was used when processing the called <task-definition> into an executable task.

2. ENV-INVOCATION-FAULT

   An ENV-INVOCATION-FAULT class exception is generated for the following exceptions:

   — The task group specified by the <task-group-identifier> in the <call-task> is not defined within the environmental information of the server TP system.

   — The task specified by the <task-identifier> in the <call-task> could not be started due to access to the called task being denied.

   — The destination specified by the value of <os-name-value> in the <call-task> is not defined within the environmental information of the TP system in which the task is executing.

   — If the destination was not explicitly specified in the <call-task>, then the destination specified by the <task-group-identifier> is not defined within the environmental information of the TP system in which the task is executing.

   — The task specified by the <task-identifier> in the <call-task> could not be started because the client TP system is executing a task call on the same dialogue as a previous task call to a server TP system that accepts only a single task call per dialogue.

   — The <uuid-string-literal> in the task group specification used by the client program for the server task is not the same as the <uuid-string-literal> used by the server task.

   — The major version number used by the client program is not equal to the major version number used by the server task.

   — The major version number used by the client program is equal to the major version number used by the server task, and the minor version number used by the client program is greater than the minor version number used by the server task.

3. ENV-INVOCATION-ERROR

   An ENV-INVOCATION-ERROR class exception is generated for the following exceptions:

   — The task specified by the <task-identifier> in the <call-task> could not be started due to the task instance high water mark being exceeded on the server TP system.

— The task specified by the <task-identifier> in the <call-task> is disabled.

— The task specified by the <task-identifier> in the <call-task> could not be started due to the server TP system not being available because of system failure or not being started.

— The task group specified by the <task-group-identifier> in the <call-task> is not available due to a system failure.

— The task specified by the <task-identifier> in the <call-task> could not be started due to a communications failure.

4. AP-EXECUTION-FAULT

   An AP-EXECUTION-FAULT class exception is generated for the following exceptions:

   — Non-composable task was called to execute within the client task's transaction.

   — Composable task was called to execute independently from the client task's transaction.

   — The subscript used to reference an element in a fixed-length array is invalid (referring to A(n), where n < 1 or n > maximum size of array).

   — The subscript used to reference an element in a varying-length array is invalid (referring to A(n), where n < 1 or n > the value of the depending on field).

   — The value of the depending on field used when referencing an element of a varying-length array is invalid (the value of the depending on field < minimum size of array or the value of the depending on field > maximum size of array).

   — The value of the DEPENDING ON field was invalid when referencing a <workspace-identifier>.

5. ENV-EXECUTION-ERROR

   An ENV-EXECUTION-ERROR class exception is generated for the following exceptions:

   — The task specified by the <task-identifier> in the <call-task> could not complete successfully due to a system failure on the server TP system.

   — The task specified by the <task-identifier> in the <call-task> could not complete and return results of the call due to a communication failure between the client TP system and the remote server TP system.

6. AP-RESPONSE-FAULT

   An AP-RESPONSE-FAULT class exception is generated for the following exception:

   — The results of the <call-task> could not be returned to the calling task due to a syntax error.

7. ENV-UNSPECIFIED-FAULT

   An ENV-UNSPECIFIED-FAULT class exception is generated for the following exception:

   — The called task did not complete due to an unknown error condition in the RTI protocol.

8. Any exception returned by the called task.

IMPLEMENTOR NOTES

1. An implementation can restrict the number of incoming task calls on a dialogue to one.
2. An AP-EXECUTION-FAULT class is generated when an implementation prohibits a task from calling itself directly or indirectly.

SEE ALSO

None.

10.3.6 <cancel-submit>

NAME

<cancel-submit> — Removes a task submission request entry from the task queue.

SYNOPSIS

```
<cancel-submit> ::
    CANCEL [TASK] SUBMIT <cancel-id>
        <cancel-with>

<cancel-with> ::
      <nil>
    | [WITH] INDEPENDENT [WORK]
    | [WITH] DEPENDENT [WORK]

<cancel-id> ::
      ID <id-workspace-field>
    | IDENTIFIER <id-workspace-field>
```

SYNTAX RULES

1. When <cancel-submit> is used in a <noncomposable-task>, it must be inside of a <transaction-block>.

GENERAL RULES

1. The normal execution flow for <cancel-submit> is to remove from the task queue the task submission request entry specified by the value of <id-workspace-field>.
2. Deviations from the normal execution flow occur when one of the conditions listed under EXCEPTIONS occurs; the <cancel-submit> completes with the exception associated with the condition.
3. The default for a non-restartable task is WITH INDEPENDENT WORK.
4. The default for a restartable task is WITH DEPENDENT WORK.
5. DEPENDENT WORK specifies that the removal of the task submission request entry is done transactionally with the current task transaction.
6. INDEPENDENT WORK specifies that the removal of the task submission request entry is done immediately and without regard to the outcome of the current task transaction.
7. The contents of the <id-workspace-field> are opaque and interpreted by the TP system.
8. <cancel-submit> can remove a submitted task from a task queue only if the submitting task and the task queue are on the same TP system and the submitted task has not been forwarded or started executing.

EXCEPTIONS

The following exception classes can be generated:

1. ENV-EXECUTION-FAULT

   An ENV-EXECUTION-FAULT class exception is generated for the following exception:

   — The submitted task specified by <id-workspace-field> could not be removed from the task queue because no submitted task with the value specified by <id-

workspace-field> was found on the local task queue, or the submitted task is executing.

— The submitter TP system encountered a problem removing the task submission request from the task queue.

SEE ALSO

None.

10.3.7 <concurrent-block>

NAME

<concurrent-block> — Defines a set of statements that are executed concurrently.

SYNOPSIS

```
<concurrent-block> ::
    BLOCK [WITH] CONCURRENT EXECUTION
        <concurrent-list>
    END [BLOCK] <concurrent-block-end-label>
        <exception-handler>

<concurrent-list> ::
      <nil>
    | <statement> ...

<concurrent-block-end-label> ::
      <nil>
    | <label-identifier>
```

SYNTAX RULES

1. The <label-identifier> specified in <concurrent-block-end-label> must be the same as the statement label on the the <concurrent-block>.
2. A <concurrent-block> cannot be used outside of a <transaction-block>.
3. Any <statement> in a <concurrent-block> cannot contain the <go-to> statement.

GENERAL RULES

1. The normal execution flow for a <concurrent-block> is:
   - Concurrently execute each <statement> in the <concurrent-block>. Each <statement> executes using a separate thread of execution. If any concurrent <statement> completes with <exit-block>, this does not affect normal execution flow.
   - Wait for all of the concurrent statements to complete.
   - Complete the <concurrent-block> with default flow control.
2. Deviations from the normal execution flow occur when:
   - Any thread terminates with <exit-task> flow specified without ROLLBACK and all other threads terminate with default flow control or <exit-task> flow control without ROLLBACK, the <concurrent-block> completes with <exit-task>.
   - Any thread terminates with a non-transaction exception and all other threads terminate with default flow control, <exit-task> flow control without ROLLBACK, or non-transaction exceptions, then any other non-transaction exceptions are ignored and:
     - If there is an <exception-handler>, it is executed.
     - Otherwise, the <concurrent-block> completes with that exception.
   - Any thread terminates with an <exit-task> flow specified with ROLLBACK and all other threads terminate with default flow control or non-transaction exceptions, then the <concurrent-block> completes with <exit-task> with ROLLBACK.

— Any thread terminates with a transient transaction exception or permanent transaction exception and no thread terminates with a fatal transaction exception, the <concurrent-block> completes with that exception. Any other non-fatal transaction exceptions are ignored.

— Any thread terminates with a fatal transaction exception, the <concurrent-block> terminates with that exception. Any other exceptions are ignored.

3. A workspace can be accessed concurrently for read in multiple threads of the concurrent block.
4. A workspace can be accessed for write by only one thread of the concurrent block. Any write access by one thread is incompatible with any read of the workspace by other threads of the <concurrent-block>.
5. The order of the execution of the threads within a <concurrent-block> is implementation-specific.
6. The results of accessing a single resource concurrently by more than one thread are undefined.

IMPLEMENTOR NOTES

1. An implementation can provide sequential execution of a <concurrent-block>.
2. An implementation can impose the following restrictions:

   — At most one statement can be a <call-procedure>.

   — All other statements must be <call-task>s.

   — No two statements can make calls to composable tasks on the same TP system.

   — <concurrent-block> can be disallowed in an exception handler.

3. If an implementation does not stop execution of the task for the first exception raised by a concurrent <statement> in a <concurrent-block>, then the following rules can be used when multiple exceptions are raised in a <concurrent-block>:

   — Fatal transaction exceptions take precedence over permanent transaction exception, transient transaction exceptions, and non-transaction exceptions.

   — Permanent transaction exceptions take precedence over transient transaction exceptions and non-transaction exceptions.

   — Transient transaction exceptions take precedence over non-transaction exceptions.

CONFORMANCE NOTES

1. Concurrent blocks are not required for Support Levels 1, 2 and 3.

EXCEPTIONS

None.

SEE ALSO

Section 10.3.11, Section 10.3.

10.3.8 <control-field>

NAME

<control-field> — Selects a <statement-list> to execute based on the value of a workspace field.

SYNOPSIS

```
<control-field> ::
    CONTROL FIELD <control-field-to-match>
        <control-field-case-list>
    END [CONTROL] [FIELD] <control-field-end-label>

<control-field-to-match> ::
      <integer-workspace-field>
    | <text-workspace-field>
    | <decimal-workspace-field>

<control-field-case-list> ::
    <control-field-case> ...

<control-field-case> ::
      CASE <control-field-value> : <statement-list>
    | CASE NOMATCH : <statement-list>

<control-field-value> ::
      <integer-literal>
    | <message-identifier>
    | <class-identifier>
    | <integer-workspace-field>
    | <decimal-literal>
    | <decimal-workspace-field>
    | <string-literal>
    | <text-workspace-field>

<control-field-end-label> ::
      <nil>
    | <label-identifier>
```

SYNTAX RULES

1. The <label-identifier> specified in <control-field-end-label> must be the same as the statement label on the <control-field>.
2. If used in a <noncomposable-task> outside a <transaction-block>, <control-field-to-match> and <control-field-value> cannot reference transactional workspace fields.
3. The data type of each <control-field-value> must match the data type of the corresponding <control-field-to-match>.
4. NOMATCH, if used, can be used only once and must follow all other <control-field-value>s if any, in the <control-field> statement.

GENERAL RULES

1. The normal execution flow for a <control-field> statement is:

   — Sequentially compare the <control-field-to-match> with each <control-field-value> until each <control-field-value> is compared or one <control-field-value> matches the <control-field-to-match>.

   — When a <control-field-value> matches a <control-field-to-match>, the <statement-list> associated with the <control-field-value> is executed.

   — If no <control-field-value> matches a <control-field-to-match>, the <statement-list> associated with any NOMATCH is executed.

   — The <control-field> statement completes with default flow control.

2. Deviations from the normal execution flow occur when:

   — During the comparison of the <control-field-to-match> with a <control-field-value>, one of the conditions listed under EXCEPTIONS occurs; the <control-field> statement completes with the exception associated with the condition.

   — A <statement-list> completes with explicit flow control; the <control-field> statement completes with the same explicit flow control.

   — A <statement-list> completes with an exception; the <control-field> statement completes with that exception.

EXCEPTIONS

The following exception classes can be generated:

1. AP-EXECUTION-FAULT

   An AP-EXECUTION-FAULT class exception is generated for the following exceptions:

   — The subscript used to reference an element in a fixed-length array is invalid (referring to A(n), where n < 1 or n > maximum size of array).

   — The subscript used to reference an element in a varying-length array is invalid (referring to A(n), where n < 1 or n > the value of the depending on field).

   — The value of the depending on field used when referencing an element of a varying-length array is invalid (the value of the depending on field < minimum size of array or the value of the depending on field > maximum size of array).

SEE ALSO

None.

10.3.9 <dequeue-record>

NAME

<dequeue-record> — Returns one item from a record queue and deletes the item from the queue.

SYNOPSIS

```
<dequeue-record> ::
    DEQUEUE <dequeue-type>
        <dequeue-parts-list>
        INTO <workspace-identifier>

<dequeue-type> ::
      <dequeue-first>
    | <dequeue-key>
    | <dequeue-id>

<dequeue-first> ::
    FIRST RECORD

<dequeue-key> ::
    RECORD BY KEY <workspace-field-1> VALUE
        <dequeue-key-value>

<dequeue-id> ::
      RECORD BY ID <id-workspace-field-1>
    | RECORD BY IDENTIFIER <id-workspace-field-1>

<dequeue-key-value> ::
      <workspace-field-2>
    | <string-literal>
    | <integer-literal>
    | <decimal-literal>

<dequeue-parts-list> ::
    <dequeue-part> ...

<dequeue-part> ::
      <dequeue-queue>
    | <dequeue-wait>
    | <dequeue-element-identifier>

<dequeue-queue> ::
    FROM <os-name-literal>

<dequeue-wait> ::
      WITH WAIT
    | WITH NO WAIT

<dequeue-element-identifier> ::
      ID <id-workspace-field-2>
    | IDENTIFIER <id-workspace-field-2>
```

SYNTAX RULES

1. <dequeue-queue> must be specified once and only once in a <dequeue-record>.
2. <dequeue-wait> and <dequeue-element-identifier> can each be specified at most once in a <dequeue-record>.
3. The value of <os-name-literal> specifies the record queue from which the record is dequeued.
4. <workspace-field-1> identifies the <field-definition> within the <data-type-definition> associated with <workspace-identifier> to be used as the key field.
5. The data type of <dequeue-key-value> must match the data type of the <field-definition> specified by KEY <workspace-field-1>.
6. <workspace-field-1> and <workspace-field-2> cannot specify a varying-length array.
7. WAIT can be specified only if the <dequeue-record> is the first statement in the <transaction-block>.
8. WAIT cannot be specified in a <composable-task>.
9. When <dequeue-record> is used in a <noncomposable task>, it must be inside of a <transaction-block>.

GENERAL RULES

1. The normal execution flow for a <dequeue-record> is:
   — Dequeue the record, storing the data in workspace specified by <workspace-identifier>.
   — When <dequeue-element-identifier> is specified, the TP system returns the identifier of the entry in <id-workspace-field-2>.
   — Complete with default flow control.
2. Deviations from the normal execution flow occur when one of the conditions listed under EXCEPTIONS occurs; the <dequeue-record> completes with the exception associated with the condition, the value of <workspace-identifier> is undefined, and, if <dequeue-element-identifier> is specified, the value of <id-workspace-field-2> is undefined.
3. NO WAIT is the default wait operation.
4. Items in a record queue are ordered by the time of insertion.
5. Items in a record queue are retrieved using first-in/first-out (FIFO) order.
6. <dequeue-first> retrieves the first available record in the queue.
7. <dequeue-key> specifies that <dequeue-record> is to return the first record queue entry where the value of the field referenced by <workspace-field-1> is equal to the value specified in the <dequeue-key-value>. Duplicate key values are allowed.
8. <dequeue-id> retrieves a specific item from the queue identified by <id-workspace-field-1>.
9. Record queue keys are defined using environmental information. Record queue keys have no effect on the order of items in a queue.
10. The contents of <id-workspace-field-1> are opaque and interpreted by the TP system.

11. The contents of <id-workspace-field-2> are opaque. The value is generated by the TP system and interpreted by the TP system.
12. WAIT specifies that the dequeue record operation waits for data to become available if the record queue is empty.
13. If NO WAIT is specified and the record queue is empty, an exception is generated.
14. If WAIT is specified and the record queue is empty, the dequeue record operation waits until there is an entry in the record-queue or the wait limit expires.
15. Environmental information specifies the wait limit on a task-by-task basis.
16. If the dequeue time limit expires while the <dequeue-record> is waiting for an item to be placed in the record queue, an exception is generated.
17. WITH WAIT generates an exception if a TP system shutdown is requested while the <dequeue-record> is waiting for data.
18. The TP system generates an exception when the size of the workspace specified by the <workspace-identifier> does not match the size of the <data-type-definition> associated with the record queue specified by the <os-name-literal>. For a workspace or record that contains a varying-length array, the size of the workspace or record is determined by the DEPENDING ON field for the array.
19. <workspace-field-1> and <workspace-field-2> must be uniquely qualified <field-definition>s.

CONFORMANCE NOTES

1. WITH WAIT is not required for Support Levels 1, 2 and 3.

EXCEPTIONS

The following exception classes can be generated:

1. ENV-INVOCATION-FAULT

   An ENV-INVOCATION-FAULT class exception is generated for the following exceptions:

   — The record queue name is not defined within the environmental information of the TP system in which the task is executing.

   — The key specified by the <workspace-field-1>, <workspace-field-2> , <id-workspace-field-1> or <id-workspace-field-2> in the <dequeue-record> is not defined within the environmental information of the TP system in which the task is executing.

2. AP-EXECUTION-FAULT

   An AP-EXECUTION-FAULT class exception is generated for the following exceptions:

   — The subscript used to reference an element in a fixed-length array is invalid (referring to A(n), where n < 1 or n > maximum size of array).

   — The subscript used to reference an element in a varying-length array is invalid (referring to A(n), where n < 1 or n > the value of the depending on field).

   — The value of the depending on field used when referencing an element of a varying-length array is invalid (the value of the depending on field < minimum size of array or the value of the depending on field > maximum size of array).

— The size of workspace specified by <workspace-identifier> does not match the size of the target <data-type-definition> associated with the record queue. For a workspace or record that contains a varying-length array, the size of the workspace or record is determined by the DEPENDING ON field for the array.

— The value of the DEPENDING ON field was invalid when referencing a <workspace-field-1>, <workspace-field-2>, <id-workspace-field-1> or <id-workspace-field-2> containing a varying-length array.

3. ENV-EXECUTION-FAULT

   An ENV-EXECUTION-FAULT class exception is generated for the following exception:

   — The TP system in which the task is executing encountered a problem removing (dequeuing) an item from the record queue.

4. REQUEST-TIMEOUT-ERROR

   A REQUEST-TIMEOUT-ERROR class exception is generated for the following exceptions:

   — The dequeue record time limit environmental attribute was exceeded.

5. NO-OUTPUT-ERROR

   A NO-OUTPUT-ERROR class exception is generated for the following exceptions:

   — WAIT was not specified and the record queue is empty or no record is found so no data is returned.

6. SYSTEM-SHUTDOWN-FAULT

   A SYSTEM-SHUTDOWN-FAULT class exception is generated for the following exception:

   — A system shutdown request was issued while the queue operation is waiting (WAIT is specified) for an item to appear in the record queue.

IMPLEMENTOR NOTES

1. An implementation can impose certain restrictions on the identifiers used to define the <workspace-field> that specifies the key for a dequeue record operation, including the elemental field identifier, any records in which the elemental field identifier is nested, and the workspace name that contains the key.

   The restrictions an implementation can impose are:

   — Can be restricted to eight characters from the STDL alphanumeric character set including:

   — Letters (upper-case and lower-case characters are equivalent)

   — Numbers

   — Underscore (_) and hyphen (-) are disallowed.

   — Must begin with a letter.

NOTES

1. <workspace-identifier> must be defined in a <workspace-list-definition> of the <task-definition> that contains the <dequeue-record>.

SEE ALSO

None.

10.3.10 <enqueue-record>

NAME

<enqueue-record> — Enqueues a record onto a record queue.

SYNOPSIS

```
<enqueue-record> ::
    ENQUEUE RECORD
        <enqueue-parts-list>
        USING <workspace-identifier>

<enqueue-parts-list> ::
    <enqueue-part> ...

<enqueue-part> ::
      <enqueue-queue>
    | <enqueue-element-identifier>

<enqueue-queue> ::
    ON <os-name-literal>

<enqueue-element-identifier> ::
      ID <id-workspace-field>
    | IDENTIFIER <id-workspace-field>
```

SYNTAX RULES

1. <enqueue-queue> must be specified once and only once in an <enqueue-record>.
2. <enqueue-element-identifier> can be specified at most once in an <enqueue-record>.
3. The value of <os-name-literal> specifies the record queue onto which the record is enqueued.
4. When <enqueue-record> is used in a <noncomposable task>, it must be inside of a <transaction-block>.

GENERAL RULES

1. The normal execution flow for an <enqueue-record> is:
   — Enqueue the data to the record queue.
   — When a <id-workspace-field> is specified, the TP system returns the identifier of the entry in <id-workspace-field>.
   — Complete with default flow control.
2. Deviations from the normal execution flow occur when one of the conditions listed under EXCEPTIONS occurs; the <enqueue-record> completes with the exception associated with the condition and, if <id-workspace-field> is specified, the value of <id-workspace-field> is undefined.
3. The contents of the workspace specified by the <workspace-identifier> is inserted into the queue.
4. Items in a record queue are ordered by the time of insertion.
5. Each item in the record queue is uniquely identified by an identifier.

6. The value of <id-workspace-field> can be used in a <dequeue-record> or <read-queue-record> to uniquely identify this item in the record queue.
7. The contents of the <id-workspace-field> is opaque. The value is generated by the TP system and interpreted by the TP system.
8. The TP system generates an exception, if the size of the workspace specified by the <workspace-identifier> does not match the size of the <data-type-definition> associated with the record queue specified by the <os-name-literal>. For a workspace or data type definition that contains a varying-length array, the size of the workspace or data type definition is determined by the DEPENDING ON field for the array.

CONFORMANCE NOTES

1. Record queues are not required for Support Levels 1 and 2.

EXCEPTIONS

The following exception classes can be generated:

1. ENV-INVOCATION-FAULT

   An ENV-INVOCATION-FAULT class exception is generated for the following exception:

   — The record queue name is not defined within the environmental information of the TP system in which the task is executing.

2. AP-EXECUTION-FAULT

   An AP-EXECUTION-FAULT class exception is generated for the following exceptions:

   — The subscript used to reference an element in a fixed-length array is invalid (referring to A(n), where n < 1 or n > maximum size of array).

   — The subscript used to reference an element in a varying-length array is invalid (referring to A(n), where n < 1 or n > the value of the depending on field).

   — The value of the depending on field used when referencing an element of a varying-length array is invalid (the value of the depending on field < minimum size of array or the value of the depending on field > maximum size of array).

   — The size of workspace specified by <workspace-identifier> does not match the size of the target <data-type-definition> associated with the record queue. For a workspace or record that contains a varying-length array, the size of the workspace or record is determined by the DEPENDING ON field for the array.

   — The value of the DEPENDING ON field was invalid when referencing a <workspace-identifier>.

3. ENV-EXECUTION-FAULT

   An ENV-EXECUTION-FAULT class exception is generated for the following exception:

   — The TP system in which the task is executing encountered a problem storing an item into the record queue.

SEE ALSO

None.

10.3.11 <exception-handler>

NAME

<exception-handler> — Executes a <statement-list> when an exception is raised.

SYNOPSIS

```
<exception-handler> ::
      <nil>
    | <exception-handler-list>

<exception-handler-list> ::
    EXCEPTION [HANDLER] [IS]
        <statement-list>
    END [EXCEPTION] [HANDLER]
```

SYNTAX RULES

1. A statement that must be used inside of a <transaction-block> must be enclosed by a <transaction-block> when it is used in the <exception-handler> of a <transaction-block> or <statement-block> outside of a <transaction-block>.

GENERAL RULES

1. The normal execution flow for an <exception-handler> is:
   — Initialize the EXCEPTION-INFO-WORKSPACE to describe the exception being handled.
   — Execute the <statement-list>. Complete the <exception-handler> with default flow control.
2. Deviations from the normal execution flow occur when the <statement-list> completes with:
   — Explicit flow control; the <exception-handler> completes with the same explicit flow control.
   — Exception; the <exception-handler> completes with that exception.

EXCEPTIONS

None.

SEE ALSO

Section 10.3.

10.3.12 <exception-information>

NAME

<exception-information> — Provides the exception information for <restart-transaction> and <raise-exception>.

SYNOPSIS

```
<exception-information> ::
      [EXCEPTION] CLASS <exception-class-expression>
    | [EXCEPTION] CODE <exception-code-expression>
          <exception-group>

<exception-class-expression> ::
      <class-identifier>
    | <integer-workspace-field>
    | <integer-literal>

<exception-code-expression> ::
      <message-identifier>
    | <integer-workspace-field>
    | <integer-literal>

<exception-group> ::
      <nil>
    | IN <message-group-identifier>
    | IN <uuid-workspace-field>
    | IN <uuid-string-literal>
```

SYNTAX RULES

None.

GENERAL RULES

1. IN specifies the exception group for a system-defined exception code or the <message-group-definition> for an application-defined exception code. If omitted, then the exception code must belong to a message group with a UUID of zero.

2. If EXCEPTION CODE is specified, the TP system maps the exception code to the appropriate exception class, as defined in the <message-group-definition>. The exception code must have an equivalent message number defined in the <message-definition>.

3. If EXCEPTION CLASS is specified, the TP system:

   — Checks to make sure that the exception class is one of the standard, predefined exception classes.

   — Sets the exception code to 0.

SEE ALSO

None.

10.3.13 <exit-block>

NAME

<exit-block> — Exits the current block.

SYNOPSIS

```
<exit-block> ::
    EXIT BLOCK <transaction-control>
```

SYNTAX RULES

1. <transaction-control> can be specified on an <exit-block> only if the block to be exited is a <transaction-block>.
2. An <exit-block> can only be specified inside of a <statement-block>, <transaction-block> or <concurrent-block>.

GENERAL RULES

1. The normal execution flow for an <exit-block> is:
   - If <transaction-control> is specified, perform the transaction control. If <transaction-control> can be specified but is not specified, then perform the default for <transaction-control>.
   - Complete with explicit <exit-block> flow control.
2. Deviations from the normal execution flow occur when one of the conditions listed under EXCEPTIONS occurs; the <exit-block> completes with the exception associated with that condition.

EXCEPTIONS

The following exception classes can be generated:

1. TXN-FAILURE-ERROR

   A TXN-FAILURE-ERROR class exception is generated for the following exceptions:
   - One of the resource managers participating in the transaction was unable to commit the transaction.
   - One of the resource managers participating in the transaction failed to acknowledge a message when attempting to commit the transaction.

SEE ALSO

Section 10.3.26.

10.3.14 <exit-task>

NAME

<exit-task> — Exits the current task.

SYNOPSIS

```
<exit-task> ::
    EXIT TASK <transaction-control>
```

SYNTAX RULES

1. <transaction-control> can be specified on <exit-task> only within a <transaction-block> within a <noncomposable-task>.

GENERAL RULES

1. The normal execution flow for an <exit-task> is:
   - If <transaction-control> is specified, perform the transaction control. If <transaction-control> can be specified but is not specified, then perform the default for <transaction-control>.
   - Complete with explicit <exit-task> flow control.
2. Deviations from the normal execution flow occur when one of the conditions listed under EXCEPTIONS occurs; the <exit-task> completes with the exception associated with that condition.

EXCEPTIONS

The following exception classes can be generated:

1. TXN-FAILURE-ERROR

   A TXN-FAILURE-ERROR class exception is generated for the following exceptions:
   - One of the resource managers participating in the transaction was unable to commit the transaction.
   - One of the resource managers participating in the transaction failed to acknowledge a message when attempting to commit the transaction.

SEE ALSO

Section 10.3.26.

10.3.15 <get-message>

NAME

<get-message> — Translates a message number into message text.

SYNOPSIS

```
<get-message> ::
    GET MESSAGE NUMBER <get-message-number> <get-message-group>
        <get-message-parts-list>
        <get-message-using>
        INTO <text-workspace-field-1>

<get-message-number> ::
      <message-identifier>
    | <integer-literal>
    | <integer-workspace-field-1>

<get-message-group> ::
      <nil>
    | IN <message-group-identifier>
    | IN <uuid-string-literal>
    | IN <uuid-workspace-field>

<get-message-parts-list> ::
      <nil>
    | <get-message-part> ...

<get-message-part> ::
      <get-message-source>
    | <get-message-length>
    | <get-message-language>

<get-message-source> ::
      SOURCE [IS] APPLICATION
    | SOURCE [IS] SYSTEM
    | SOURCE [IS] <integer-workspace-field-2>

<get-message-length> ::
    LENGTH [IS] <integer-workspace-field-3>

<get-message-language> ::
    LANGUAGE [IS] <get-message-language-expression>

<get-message-language-expression> ::
      <human-language-literal>
    | <s-text-workspace-field>

<get-message-using> ::
      <nil>
    | USING <get-message-argument-list>

<get-message-argument-list> ::
    <text-workspace-field-2> , ...
```

SYNTAX RULES

1. <get-message-source>, <get-message-length>, and <get-message-language> can each appear at most once in a <get-message>.
2. <human-language-literal> is a human language name.
3. When <get-message> is used in a <noncomposable-task> outside of a <transaction-block>, it must not reference any transactional workspace fields.

GENERAL RULES

1. The normal execution flow for a <get-message> is:
   — Get a message into <text-workspace-field-1>.
   — When <get-message-length> is used, get the length of the message in characters into <integer-workspace-3>.
   — Complete with default flow control.
2. Deviations from the normal execution flow occur when one of the conditions listed under EXCEPTIONS occurs; the <get-message> completes with the exception associated with that condition, the value of <text-workspace-field-1> is undefined, and if <get-message-length> is used, the value of <integer-workspace-3> is undefined.
3. <get-message-group> specifies the exception group for a system-defined exception code or the <message-group-definition> for an application-defined exception code. If omitted, then the exception code must belong to a message group with a UUID of zero.
4. IN <message-group-identifier> can only be specified if SOURCE IS APPLICATION or when SOURCE IS <integer-workspace-2> is specified and contains 1 (for application).
5. <get-message-source> specifies whether the message number is an application-defined message or a system-defined message. The default is application-defined.
6. If <get-message-length> is specified, the length in characters of the returned text is placed in <integer-workspace-3>. For application-defined messages, the returned text is the message text from a <message-group-definition>.
7. <get-message-language> specifies the human language in which the <text-workspace> is displayed. The value of the <get-message-language-expression> must match a human language name specified in a <message-group-definition>.
8. If the message is application-defined, the value of <get-message-number> must match a message identifier defined within a <message-group-definition> associated with this <task-definition>. If a match is found, the message text associated with the message number in the <message-group-definition> is placed into the <text-workspace-field-1>.
9. If the message is system-defined, the value of <get-message-number> must match a system-defined message number. If a match is found, the message text defined by the system is placed into <text-workspace-field-1>.
10. The value of <integer-workspace-field-2> must be 1 (for application) or 0 (for system).
11. The message text from the <message-definition> is placed into the <text-workspace-field-1> based on the human language.
12. The <text-workspace-field-1> must be defined using a character set that can contain the characters for all human languages that can be encountered during task execution.

A system that supports Japanese and English can use the Kanji character set.

13. If multiple <message-group-definition>s are associated with this <task-definition> for the same <message-group-identifier> and <message-group-uuid>, the <human-language-literal> determines the contents of which <message-group-definition> to use.

14. The language used for <get-message> is determined as follows:

    — If <get-message-language> is specified, the value of <get-message-language-expression> is used.

    — If <get-message-language> is not specified and the task is called by a task on another TP system, the LANGUAGE field of the TASK-CALL-INFORMATION record is used (see Section 18.8.1 on page 407).

    — If <get-message-language> is not specified and the task is called by a task on the same TP system, the language is determined by the user profile of the local user.

15. If the human language name is ENGLISH, then the <text-workspace-field-1> can be defined using any of the character sets. The message text must contain characters from the Simple Latin or ISO Latin-1 execution character sets. If the <text-workspace-field-1> is defined to be:

    — SIMPLE LATIN, then no conversion is done.

    — ISO Latin-1, then no conversion is done.

    — KATAKANA, then the characters in the message text are mapped to the equivalent characters in the Katakana execution character set.

    — KANJI, then the characters in the message text are mapped to the equivalent characters in the Kanji execution character set.

16. If the human language name is JAPANESE, then the <text-workspace-field-1> must be defined using the Kanji character set. The message text must contain characters from the Kanji execution character set.

17. If the human language is not ENGLISH or JAPANESE, <text-workspace-field-1> is defined as follows:

    — If one of the human languages listed in Appendix F on page 527, use the character set mapping in Appendix F,

    — If other than a human language listed in Appendix F, use an implementation-specific mapping.

18. If the message is application-defined, the <message-text> from the <message-definition> is placed into the <text-workspace-field-1> replacing <message-parameter>s, if any, with <text-workspace-field-2>s. Any trailing space characters are not moved to the <text-workspace-field-1>. The digit in a <message-parameter> corresponds to the order of the <text-workspace-field-2> that applies in the <get-message-argument-list>.

19. If USING <get-message-argument-list> is specified for a system-defined message, it is ignored.

20. If there are insufficient <text-workspace-field-2>s for the <message-parameter>s, the results are undefined. If the <message-parameter>s are exhausted while <text-workspace-field-2>s remain, the excess <text-workspace-field-2>s are ignored.

21. Any <text-workspace-field-2>s must be defined using the same character set as that of the <text-workspace-field-1>.

EXCEPTIONS

The following exception classes can be generated:

1. AP-EXECUTION-FAULT

   An AP-EXECUTION-FAULT class exception is generated for the following exceptions:

   — The subscript used to reference an element in a fixed-length array is invalid (referring to A(n), where n < 1 or n > maximum size of array).

   — The subscript used to reference an element in a varying-length array is invalid (referring to A(n), where n < 1 or n > the value of the depending on field).

   — The value of the depending on field used when referencing an element of a varying-length array is invalid (the value of the depending on field < minimum size of array or the value of the depending on field > maximum size of array).

   — <integer-workspace-field-2> contains a value other than 0 or 1.

   — The value of the DEPENDING ON field was invalid when referencing a <workspace-field> containing a varying-length array.

   — The character set supported data type of the <text-workspace-field-1> or <text-workspace-field-2> cannot handle the human language of the message text.

   — IN <message-group-identifier> is specified and the value of <integer-workspace-field-2> is 0 (for system).

2. NO-OUTPUT-ERROR

   A NO-OUTPUT-ERROR exception class can be generated for the following exception:

   — The value of the <get-message-number> specified in the <get-message> did not match a message number defined in the <message-group-definition>, so no message text is returned.

   — The value of the human language name specified in the <get-message> did not match a human language name defined in the <message-group-definition>.

   — The value of <uuid-workspace-field> or <uuid-string-literal> did not match a known exception group or message group identifier did not match a known <message-group-definition>.

SEE ALSO

None.

10.3.16 <go-to>

NAME

<go-to> — Transfers control to the specified statement.

SYNOPSIS

```
<go-to> ::
      GO TO <label-identifier> <transaction-control>
    | GOTO <label-identifier> <transaction-control>
```

SYNTAX RULES

1. <transaction-control> can be specified on a <go-to> only when a <transaction-block> is exited.

GENERAL RULES

1. The normal execution flow for a <go-to> is:

   — If <transaction-control> is specified, perform the transaction control. If <transaction-control> can be specified but is not specified, then perform the default for <transaction-control>.

   — Complete with explicit <go-to> flow control.

2. Deviations from the normal execution flow occur when one of the conditions listed under EXCEPTIONS occurs; the <go-to> completes with the exception associated with that condition.

3. The target statement of the <go-to> is the <statement> which defined <label-identifier>.

4. The target statement must be a <statement> in a <statement-list> which contains the <go-to> either directly or recursively through any statement other than <concurrent-block>. This allows blocks to be exited to any level but does not allow blocks to be entered anywhere but at their beginning.

EXCEPTIONS

The following exception classes can be generated:

1. TXN-FAILURE-ERROR

   A TXN-FAILURE-ERROR class exception is generated for the following exceptions:

   — One of the resource managers participating in the transaction was unable to commit the transaction.

   — One of the resource managers participating in the transaction failed to acknowledge a message when attempting to commit the transaction.

SEE ALSO

Section 10.3.26.

10.3.17 <if>

NAME

<if> — Chooses an execution path based on a condition test.

SYNOPSIS

```
<if> ::
    IF <boolean-expression> THEN
        <statement-list>
    <else>
END [IF] <if-end-label>

<else> ::
      <nil>
    | ELSE <statement-list>

<if-end-label> ::
      <nil>
    | <label-identifier>
```

SYNTAX RULES

1. The <label-identifier> specified in <if-end-label> must be the same as the statement label on the <if>.
2. If used in a <noncomposable-task> outside a <transaction-block>, <boolean-expression> cannot reference a transactional workspace field.

GENERAL RULES

1. The normal execution flow for an <if> statement is:
   - — Evaluate the <boolean-expression>.
   - — If the <boolean-expression evaluates true, the <statement-list> associated with the THEN is executed.
   - — If the <boolean-expression> evaluates false, any <statement-list> associated with the ELSE is executed.
   - — The <if> statement completes with default flow control.
2. Deviations from the normal execution flow occur when:
   - — Evaluating a <boolean-expression> and one of the conditions listed under EXCEPTIONS occurs; the <if> statement completes with the exception associated with the condition.
   - — A <statement-list> completes with explicit flow control; the <if> statement completes with the same explicit flow control.
   - — A <statement-list> completes with an exception; the <if> statement completes with that exception.

EXCEPTIONS

The following exception classes can be generated:

1. AP-EXECUTION-FAULT

   An AP-EXECUTION-FAULT class exception is generated for the following exceptions:

   — The subscript used to reference an element in a fixed-length array is invalid (referring to A(n), where n < 1 or n > maximum size of array).

   — The subscript used to reference an element in a varying-length array is invalid (referring to A(n), where n < 1 or n > the value of the depending on field).

   — The value of the depending on field used when referencing an element of a varying-length array is invalid (the value of the depending on field < minimum size of array or the value of the depending on field > maximum size of array).

   — The value of the DEPENDING ON field was invalid when referencing a <workspace-field> containing a varying-length array.

SEE ALSO

Section 4.15, Section 10.3.

10.3.18 <raise-exception>

NAME

<raise-exception> — Raises an exception.

SYNOPSIS

```
<raise-exception> ::
    RAISE  <exception-information> <exception-rollback>

<exception-rollback> ::
      <nil>
    | [WITH] ROLLBACK [TRANSACTION]
```

SYNTAX RULES

1. When <raise-exception> is used in a <noncomposable-task> outside of a <transaction-block>, it must not reference any transactional workspace fields.

GENERAL RULES

1. The normal execution flow for a <raise-exception> is to determine the class and code of the exception to raise and:

   — If ROLLBACK is specified, complete the <raise-exception> with a permanent transaction exception with that class and code.

   — If ROLLBACK is not specified, complete the <raise-exception> with an exception with the class, code, and type associated with the raised class.

2. Deviations from the normal execution flow occur when one of the conditions listed under EXCEPTIONS occurs; the <raise-exception> completes with the exception associated with that condition.

EXCEPTIONS

The following exception classes can be generated:

1. AP-EXECUTION-FAULT

   An AP-EXECUTION-FAULT class exception is generated for the following exceptions:

   — The subscript used to reference an element in a fixed-length array is invalid (referring to A(n), where n < 1 or n > maximum size of array).

   — The subscript used to reference an element in a varying-length array is invalid (referring to A(n), where n < 1 or n > the value of the depending on field).

   — The value of the depending on field used when referencing an element of a varying-length array is invalid (the value of the depending on field < minimum size of array or the value of the depending on field > maximum size of array).

   — The exception code specified in the <raise-exception> is not defined within an associated <message-group-definition>.

   — The exception class specified in the <raise-exception> is not one of the standard, predefined exception classes.

   — The value of <uuid-workspace-field> or <uuid-string-literal> did not match a known exception group or the value of <message-group-identifier> did not match a known <message-group-definition>.

SEE ALSO

Section 10.3.12.

10.3.19 <read-queue-record>

NAME

<read-queue-record> — Reads one item from a record queue without deleting it.

SYNOPSIS

```
<read-queue-record> ::
    READ QUEUE <read-queue-type>
       <read-queue-parts-list>
       INTO <workspace-identifier>

<read-queue-type> ::
      <read-queue-first>
    | <read-queue-next>
    | <read-queue-id>
    | <read-queue-key-first>
    | <read-queue-key-next>

<read-queue-first> ::
    FIRST RECORD

<read-queue-next> ::
    NEXT RECORD

<read-queue-id> ::
      RECORD BY ID <id-workspace-field-1>
    | RECORD BY IDENTIFIER <id-workspace-field-1>

<read-queue-key-first> ::
    FIRST RECORD
        BY KEY <workspace-field-1>
        VALUE <read-queue-key-value>

<read-queue-key-value> ::
      <workspace-field-2>
    | <string-literal>
    | <integer-literal>
    | <decimal-literal>

<read-queue-key-next> ::
    NEXT RECORD BY KEY

<read-queue-parts-list> ::
    <read-queue-part> ...

<read-queue-part> ::
      <read-queue-queue>
    | <read-queue-wait>
    | <read-queue-element-identifier>

<read-queue-queue> ::
    FROM <os-name-literal>
```

```
<read-queue-wait> ::
      WITH WAIT
    | WITH NO WAIT

<read-queue-element-identifier> ::
      ID <id-workspace-field-2>
    | IDENTIFIER <id-workspace-field-2>
```

SYNTAX RULES

1. The value of <os-name-literal> specifies the record queue from which the record is dequeued.
2. <read-queue-queue> must be specified once and only once in a <read-queue-record>.
3. <read-queue-wait> and <read-queue-element-identifier> can each be specified at most once in a <read-queue-record>.
4. <workspace-field-1> identifies the <field-definition> within the <data-type-definition> associated with <workspace-identifier> to be used as the key field.
5. The data type of <read-queue-key-value> must match the data type of the <field-definition> specified by KEY <workspace-field-1>.
6. <workspace-field-1> and <workspace-field-2> cannot specify a varying-length array.
7. WAIT can be specified only if the <read-queue-record> is the first statement in the <transaction-block>.
8. WAIT cannot be specified in a <composable-task>.
9. When <read-queue-record> is used in a <noncomposable task>, it must be inside of a <transaction-block>.

GENERAL RULES

1. The normal execution flow for a <read-queue-record> is:
   — Read the record, storing the data in workspace specified by <workspace-identifier>.
   — When <read-queue-element-identifier> is specified, the TP system returns the identifier of the entry in <id-workspace-field-2>.
   — Complete with default flow control.
2. Deviations from the normal execution flow occur when one of the conditions listed under EXCEPTIONS occurs; the <read-queue-record> completes with the exception associated with the condition, the value of <workspace-identifier> is undefined, and, if <read-queue-element-identifier> is specified, the value of <id-workspace-field-2> is undefined.
3. NO WAIT is the default for <read-queue-wait>.
4. Items in a record queue are ordered by the time of insertion.
5. Items in a record queue are retrieved using first-in/first-out (FIFO) order.
6. <read-queue-first> retrieves the first available record in the queue.
7. <read-queue-next> retrieves the next available record in the queue.

8. Within a task group context and a transaction for a record queue, the previous <read-queue-record> statement for a <read-queue-next> must be a <read-queue-first> or <read-queue-next>. The previous <read-queue-type> operation for a <read-queue-key-next> must be a <read-queue-key-first> or a <read-queue-key-next>.
9. <read-queue-key-first> specifies that <read-queue-record> is to return the first available record queue entry where the value of the field referenced by <workspace-field-1> is equal to the value specified in the <read-queue-key-value>. Duplicate key values are allowed.
10. <read-queue-key-next> specifies that <read-queue-record> is to return the next available record queue entry with the same key value as specified on the previous <read-queue-key-first>.
11. <read-queue-id> retrieves a specific item from the queue identified by <id-workspace-field-1>.
12. Record queue keys are defined using environmental information. Record queue keys have no affect on the order of items in a queue.
13. The contents of <id-workspace-field-1> are opaque and interpreted by the TP system.
14. The contents of <id-workspace-field-2> are opaque. The value is generated by the TP system and interpreted by the TP system.
15. WAIT specifies that the read queue record operation waits for data to become available if the record queue is empty.
16. If NO WAIT is specified and the record queue is empty, an exception is generated.
17. If WAIT is specified and the record queue is empty, the read queue record operation waits until there is an entry in the record-queue or the wait limit expires.
18. Environmental information specifies the wait limit on a task-by-task basis.
19. If the dequeue time limit expires while the <read-queue-record> is waiting for an item to be placed in the record queue, an exception is generated.
20. WITH WAIT generates an exception, if a TP system shutdown is requested while the <read-queue-record> is waiting for data.
21. The TP system generates an exception when the size of the workspace specified by the <workspace-identifier> does not match the size of the <datatype-definition> associated with the record queue specified by the <os-name-literal>. For a workspace or record that contains a varying-length array, the size of the workspace or record is determined by the DEPENDING ON field for the array.
22. <workspace-field-1> and <workspace-field-2> must be uniquely qualified <field-definition>s.

CONFORMANCE NOTES

1. WITH WAIT is not required for Support Levels 1, 2 and 3.

EXCEPTIONS

The following exception classes can be generated:

1. ENV-INVOCATION-FAULT

   An ENV-INVOCATION-FAULT class exception is generated for the following exceptions:

— The record queue name is not defined within the environmental information of the TP system in which the task is executing.

— The key specified by the <workspace-field-1> in the <read-queue-record> is not defined within the environmental information of the TP system in which the task is executing.

2. AP-EXECUTION-FAULT

   An AP-EXECUTION-FAULT class exception is generated for the following exceptions:

   — The subscript used to reference an element in a fixed-length array is invalid (referring to A(n), where n < 1 or n > maximum size of array).

   — The subscript used to reference an element in a varying-length array is invalid (referring to A(n), where n < 1 or n > the value of the depending on field).

   — The value of the depending on field used when referencing an element of a varying-length array is invalid (the value of the depending on field < minimum size of array or the value of the depending on field > maximum size of array).

   — The size of workspace specified by <workspace-identifier> does not match the size of the target <data-type-definition> associated with the record queue. For a workspace or record that contains a varying-length array, the size of the workspace or record is determined by the DEPENDING ON field for the array.

   — The value of the DEPENDING ON field was invalid when referencing a <workspace-field> containing a varying-length array.

   — A <read-queue-next> operation was unable to complete successfully because the previous operation was neither a <read-queue-first> nor a <read-queue-next>.

   — A <read-queue-key-next> operation was unable to complete successfully because the previous operation was neither a <read-queue-key-first> nor a <read-queue-key-next>.

3. ENV-EXECUTION-FAULT

   An ENV-EXECUTION-FAULT class exception is generated for the following exception:

   — The TP system in which the task is executing encountered a problem reading an item on the record queue.

4. REQUEST-TIMEOUT-ERROR

   A REQUEST-TIMEOUT-ERROR class exception is generated for the following exceptions:

   — The dequeue record time limit environmental attribute was exceeded.

5. NO-OUTPUT-ERROR

   A NO-OUTPUT-ERROR class exception is generated for the following exceptions:

   — WAIT was not specified and the record queue is empty or no record is found and no data is returned.

6. SYSTEM-SHUTDOWN-FAULT

   A SYSTEM-SHUTDOWN-FAULT class exception is generated for the following exception:

— A system shutdown request was issued while the queue operation is waiting (WAIT is specified) for an item to appear in the record queue.

SEE ALSO

None.

10.3.20 <reraise-exception>

NAME

<reraise-exception> — Reraises an exception being processed by an exception handler.

SYNOPSIS

```
<reraise-exception>::
    RERAISE EXCEPTION
```

SYNTAX RULES

1. <reraise-exception> can only be used in an exception handler.

GENERAL RULES

1. The normal execution flow for <reraise-exception> is to terminate with the same exception that caused the exception handler to be executed, preserving the contents of EXCEPTION-INFO-WORKSPACE and the exception type.

EXCEPTIONS.

None.

SEE ALSO

None.

10.3.21 <restart-transaction>

NAME

<restart-transaction> — Raises a transient transaction exception.

SYNOPSIS

```
<restart-transaction> ::
    RESTART [TRANSACTION] [WITH] <exception-information>
```

SYNTAX RULES

1. <restart-transaction> cannot be specified within a non-restartable task.
2. When <restart-transaction> is used in a <noncomposable task>, it must be inside of a <transaction-block>.

GENERAL RULES

1. The normal execution flow for a <restart-transaction> is to determine the class and code of the transient transaction exception to raise and complete with a transient transaction exception.
2. Deviations from the normal execution flow occur when one of the conditions listed under EXCEPTIONS occurs; the <restart-transaction> completes with the exception associated with that condition.

CONFORMANCE NOTES

1. <restart-transaction> is not required for Support Levels 1 and 2.

EXCEPTIONS

The following exception classes can be generated:

1. AP-EXECUTION-FAULT

   An AP-EXECUTION-FAULT class exception is generated for the following exceptions:

   — The subscript used to reference a element in an fixed-length array is invalid (referring to A(n), where n < 1 or n > maximum size of array).

   — The subscript used to reference an element in a varying-length array is invalid (referring to A(n), where n < 1 or n > the value of the depending on field).

   — The value of the depending on field used when referencing an element of a varying-length array is invalid (the value of the depending on field < minimum size of array or the value of the depending on field > maximum size of array).

   — The exception code specified in the <restart-transaction> is not defined within an associated <message-group-definition>.

   — The exception class specified in the <restart-transaction> is not one of the standard, predefined exception classes.

   — The value of <uuid-workspace-field> or <uuid-string-literal> did not match a known exception group or the value of <message-group-identifier> did not match a known <message-group-definition>.

SEE ALSO

Section 10.3.12.

10.3.22 <select-first>

NAME

<select-first> — Selects a <statement-list> to execute based on a condition test.

SYNOPSIS

```
<select-first> ::
    SELECT FIRST [TRUE] [OF]
        <select-first-case-list>
    END [SELECT] [FIRST] <select-first-end-label>

<select-first-case-list> ::
    <select-first-case> ...

<select-first-case> ::
      CASE <boolean-expression> : <statement-list>
    | CASE NOMATCH : <statement-list>

<select-first-end-label> ::
      <nil>
    | <label-identifier>
```

SYNTAX RULES

1. The <label-identifier> specified in <select-first-end-label> must be the same as the statement label on the <select-first>.
2. NOMATCH, if used, can be used only once and must follow all other <boolean-expression>s, if any, in the <select-first> statement.
3. If used in a <noncomposable-task> outside a <transaction-block>, <boolean-expression> cannot reference a transactional workspace field.

GENERAL RULES

1. The normal execution flow for a <select-first> statement is:
   - Sequentially evaluate the <boolean-expression>s until they are all evaluated or one evaluates true.
   - If a <boolean-expression> evaluates true, the <statement-list> associated with the <boolean-expression> is executed.
   - If no <boolean-expression> evaluates true, the <statement-list> associated with any NOMATCH is executed.
   - The <select-first> statement completes with default flow control.
2. Deviations from the normal execution flow occur when:
   - Evaluating a <boolean-expression> and one of the conditions listed under EXCEPTIONS occurs; the <select-first> statement completes with the exception associated with the condition.
   - A <statement-list> completes with explicit flow control; the <select-first> statement completes with the same explicit flow control.
   - A <statement-list> completes with an exception; the <select-first> statement completes with that exception.

EXCEPTIONS

The following exception classes can be generated:

1. AP-EXECUTION-FAULT

   An AP-EXECUTION-FAULT class exception is generated for the following exceptions:

   — The subscript used to reference an element in a fixed-length array is invalid (referring to A(n), where n < 1 or n > maximum size of array).

   — The subscript used to reference an element in a varying-length array is invalid (referring to A(n), where n < 1 or n > the value of the depending on field).

   — The value of the depending on field used when referencing an element of a varying-length array is invalid (the value of the depending on field < minimum size of array or the value of the depending on field > maximum size of array).

   — The value of the DEPENDING ON field was invalid when referencing a <workspace-field> containing a varying-length array.

SEE ALSO

Section 4.15, Section 10.3.

10.3.23 <statement-block>

NAME

<statement-block> — Groups multiple statements into a single unit for which exceptions can be handled.

SYNOPSIS

```
<statement-block> ::
    BLOCK
        <statement-list>
    END [BLOCK] <statement-block-end-label>
        <exception-handler>

<statement-block-end-label> ::
      <nil>
    | <label-identifier>
```

SYNTAX RULES

1. The <label-identifier> specified in <statement-block-end-label> must be the same as the statement label on the <statement-block>.

GENERAL RULES

1. The normal execution flow for a <statement-block> is to execute the <statement-list> and complete with default flow control. If the <statement-list> completes with explicit <exit-block> flow control, this does not affect the normal execution flow.
2. Deviations from the normal execution flow occur when the <statement-list> completes with:
   - Explicit <exit-task> or <go-to> flow control; the <statement-block> completes with the same explicit flow control.
   - Exception, and <statement-block> is outside of a <transaction-block>; if the exception type is:
     - Fatal, the <statement-block> completes with the exception.
     - Transient transaction, permanent transaction or non-transaction, then:
       - If there is an <exception-handler>, the <exception-handler> is executed.
       - Otherwise, the <statement-block> completes with a permanent transaction exception.
   - Exception, and <statement-block> is inside of a <transaction-block> or <composable-task>; if the exception type is:
     - Fatal, permanent transaction, or transient transaction; the <statement-block> completes with that exception.
     - Non-transaction, then:
       - If there is an <exception-handler>, it is executed.
       - Otherwise, the <statement-block> completes with that exception.
3. Execution of <exception-handler> can complete with:
   - Default flow control; the <statement-block> completes with default flow control.

— Explicit flow control; the <statement-block> completes with the same explicit flow control.

— Exception; the <statement-block> completes with that exception.

EXCEPTIONS

None.

SEE ALSO

Section 10.3, Section 10.3.11.

10.3.24 <submit-task>

NAME

<submit-task> — Submits a task for delayed execution.

SYNOPSIS

```
<submit-task> ::
    SUBMIT TASK <task-identifier>
        <submit-parts-list>
        <submit-with>
        <submit-using>

<submit-with> ::
      <nil>
    | [WITH] INDEPENDENT [WORK]
    | [WITH] DEPENDENT [WORK]

<submit-parts-list> ::
      <nil>
    | <submit-part> ...

<submit-part> ::
      <submit-in>
    | <submit-at>
    | <submit-id>
    | <submit-repeat>
    | <submit-trigger>

<submit-in> ::
    IN <task-group-identifier>

<submit-at> ::
    AT <os-name-value>

<submit-id> ::
      ID <id-workspace-field>
    | IDENTIFIER <id-workspace-field>

<submit-using> ::
      <nil>
    | USING <submit-argument-list>

<submit-argument-list> ::
    <workspace-identifier> , ...
```

SYNTAX RULES

1. <submit-in>, <submit-at>, <submit-id>, <submit-trigger> and <submit-repeat> can each appear at most once in a <submit-task>.
2. When the called task is in another task group, <submit-in> is required.
3. When <submit-at> is specified, <submit-in> is required.
4. <os-name-value> contains either a destination name or a distribution list name.

5. When <os-name-value> specifies a distribution list name, <submit-id> cannot be specified.
6. Private and shared workspaces can be passed.
7. When <submit-task> is used in a <noncomposable-task>, it must be inside of a <transaction-block>.

GENERAL RULES

1. The normal execution flow for a <submit-task> is:
   — Create the task submission request entry on the task queue.
   — When <submit-id> is specified, the TP system returns the identifier of the entry in <id-workspace-field>.
   — Complete with default flow control.
2. Deviations from the normal execution flow occur when one of the conditions listed under EXCEPTIONS occurs; the <submit-task> completes with the exception associated with the condition and, if <id-workspace-field> is specified, the value of <id-workspace-field> is undefined.
3. Each entry in the task queue is uniquely identified by an identifier.
4. The value returned in <id-workspace-field> can be used in a <cancel-submit> to uniquely identify this item in the task queue.
5. The contents of the <id-workspace-field> is opaque. The value is generated by the TP system and interpreted by the TP system.
6. The default for a non-restartable task is WITH INDEPENDENT WORK.
7. The default for a restartable task is WITH DEPENDENT WORK.
8. DEPENDENT WORK specifies that the <submit-task> request is available for processing when the submitter task's transaction commits.
9. INDEPENDENT WORK specifies that the <submit-task> request is immediately available for processing regardless of whether or not the submitting task's transaction commits or rolls back.
10. <task-identifier> is the name of the <task-definition> to be submitted. The <task-identifier> specified in the <submit-task> must match the name of a <task-interface> defined in the submitted task's <task-group-specification>.
11. The submitting task does not wait for the submitted task to complete before continuing execution.
12. If no <task-group-identifier> is specified, the <task-group-identifier> of the <noncomposable-task> or <composable-task> is used when translating the <task-definition> into a executable task.
13. The <data-type-definition>s associated with the <workspace-identifier>s named in the <submit-argument-list> in the USING clause of the submitting task correspond in order to the <data-type-definition>s associated with the <interface-argument>s in the USING clause of <task-argument-specification> for the task group specification for the submitted task.The corresponding <data-type-definition>s must match in number, data types, and length of fields.

14. The <task-interface-argument-list-definition> specified for <task-identifier> in the <task-group-specification> must have the passing characteristic of INPUT only.

15. If both <submit-at> and <submit-in> are omitted, then the submitter TP system stores a single task submission request on a task queue. The submitter TP system does not use the destination to identify the server TP system, the client TP system, or to indicate whether or not the task submission request is forwarded within a transaction, because the server TP system and client TP system are the same as the submitter TP system.

16. If <submit-at> is omitted and <submit-in> is specified, the submitter TP system uses a destination to find information required for the task submission. The implementation on the submitter TP system can choose to use either a destination with the same name as the server task group or use a default destination. In either case, the task group entry within the destination that matches the server task group identification and has a compatible version number is used to determine the client TP system and whether or not to forward the task submission request within a transaction. An implementation can choose to use either the task group name or UUID as the task group identification. The submitter TP system stores a single task submission request on a local task queue.

17. If <submit-at> is specified and contains a destination name, the submitter TP system uses the destination name to map to a client TP system, and to indicate whether or not the task submission request is forwarded within a transaction. The client TP system uses the destination name to identify the server TP system. The submitter TP system stores a single task submission request on a local task queue.

18. If <submit-at> is specified and contains a distribution list name, the submitter TP system uses the distribution list name to map to a distribution list. The submitter TP system uses each destination name in the distribution list to map to a client TP system and determines whether or not to forward the task submission request within a transaction. The client TP system uses the destination name to identify the server TP system. The submitter TP system stores a single task submission request on a local task queue for each entry on the distribution list.

19. Whether a task is submitted with INDEPENDENT WORK or DEPENDENT WORK is independent of whether or not the submitted task is composable.

CONFORMANCE NOTES

1. Local task queue and therefore <submit-task> is not required for Support Level 1.
2. Forwarding is not required for Support Level 1.
3. Forwarding within a transaction is not required for Support Levels 1, 2 and 3.
4. Task dequeuer is not required for Support Levels 1 and 2.
5. Multiple-entry distribution lists are not required for Support Levels 1, 2 and 3.

EXCEPTIONS

The following exception classes can be generated:

1. AP-INVOCATION-FAULT

   An AP-INVOCATION-FAULT class exception is generated for the following exception:

   — A specification mismatch occurred on a <submit-task>. The <task-group-specification> that was used when processing the submitting <task-definition>

into an executable task is incompatible with the <task-group-specification> that was used when processing the submitted <task-definition> into an executable task.

2. ENV-INVOCATION-FAULT

   An ENV-INVOCATION-FAULT class exception is generated for the following exceptions:

   — The distribution list specified by the <os-name-value> in the <submit-task> is not defined within the environmental information of the submitter TP system.

   — If the <os-name-value> representing a destination name or distribution list is not specified in the <submit-task> and if a default destination is used, that default destination is not defined within the environmental information of the submitter TP system.

   — The destination specified by the <os-name-value> for a destination name or a destination name included in a distribution list of a <submit-task> is not defined within the environmental information of the submitter TP system.

3. ENV-INVOCATION-ERROR

   An ENV-INVOCATION-ERROR class exception is generated for the following exception:

   — The task queue is disabled.

4. ENV-EXECUTION-FAULT

   An ENV-EXECUTION-FAULT class exception is generated for the following exception:

   — The submitter TP system encountered a problem storing the task submission request onto the task queue.

5. AP-EXECUTION-FAULT

   An AP-EXECUTION-FAULT class exception is generated for the following exceptions:

   — The subscript used to reference an element in a fixed-length array is invalid (referring to A(n), where n < 1 or n > maximum size of array).

   — The subscript used to reference an element in a varying-length array is invalid (referring to A(n), where n < 1 or n > the value of the depending on field).

   — The value of the depending on field used when referencing an element of a varying-length array is invalid (the value of the depending on field < minimum size of array or the value of the depending on field > maximum size of array).

   — The value of the DEPENDING ON field was invalid when referencing a <workspace-identifier>.

IMPLEMENTOR NOTES

1. An implementation is not required to support a <submit-task> within a <concurrent-block>.

SEE ALSO

Section 10.3.24.2, Section 10.3.24.1.

10.3.24.1 <submit-repeat>

NAME

<submit-repeat> — Repeats a <submit-task>.

SYNOPSIS

```
<submit-repeat> :: REPEATING [EVERY] <delta-time-value>
```

SYNTAX RULES

None.

GENERAL RULES

1. The task dequeuer repeats the <submit-task> after the specified interval of time.
2. Triggers in <submit-trigger> determine the first call for the <submit-task>.
3. After the successful completion of a <submit-task> the call is repeated using the interval specified by <delta-time-value>.
4. The repeat specification for the <submit-task> is processed by the client TP system.

EXCEPTIONS

None.

SEE ALSO

Section 4.17.2.

10.3.24.2 <submit-trigger>

NAME

<submit-trigger> — Holds a task submission request entry for later processing.

SYNOPSIS

```
<submit-trigger> ::
      HOLD FOR OPERATOR
    | HOLD FOR <delta-time-value>
    | HOLD UNTIL <absolute-time-value> <submit-hold-time-of>

<submit-hold-time-of> ::
      <nil>
    | OF SUBMITTER [SYSTEM]
    | OF CLIENT [SYSTEM]
```

SYNTAX RULES

None.

GENERAL RULES

1. HOLD specifies activation triggers for the <submit-task>.
2. HOLD FOR OPERATOR specifies that the <submit-task> is to be held until released by a computer operator at the client TP system.
3. HOLD FOR <delta-time> specifies that the <submit-task> is held for the specified amount of time.
4. HOLD UNTIL <absolute-time> specifies that the <submit-task> is held until the specified time.
5. <absolute-time-value> can be specified as the local time on the submitter's TP system by specifying OF SUBMITTER SYSTEM or as the time at the client's TP system by specifying OF CLIENT SYSTEM.
6. The default for <submit-hold-time-of> is OF SUBMITTER SYSTEM.
7. An <absolute-time-value> earlier than the current TP system time causes the task submission request to be treated as if no trigger has been specified.
8. When no <submit-trigger> is specified, a task submission request entry is immediately eligible for processing by the client TP system.
9. <absolute-time-value> specified OF SUBMITTER SYSTEM is converted to universal time by the submitter TP system before forwarding the task submission to the client TP system.
10. <absolute-time-value> specified OF CLIENT SYSTEM is not modified by the submitter TP system before forwarding the task submission to the client TP system.
11. <delta-time-value> is converted into absolute universal time by the submitter TP system before forwarding the task submission to the client TP system.
12. The activation triggers for the <submit-task> are processed by the client TP system.
13. If a client TP system receives a task submission with a trigger time in the past, the client TP system treats the task submission as if it has no trigger.

EXCEPTIONS

None.

SEE ALSO

Section 4.17.2, Section 4.17.1.

10.3.25 <transaction-block>

NAME

<transaction-block> — Demarcates a set of operations that are executed within the same transaction.

SYNOPSIS

```
<transaction-block> ::
    BLOCK [WITH] TRANSACTION
        <statement-list>
    END [BLOCK] <transaction-block-end-label>
        <exception-handler>

<transaction-block-end-label> ::
      <nil>
    | <label-identifier>
```

SYNTAX RULES

1. The <label-identifier> specified in <transaction-block-end-label> must be the same as the statement label on the <transaction-block>.
2. A <transaction-block> cannot be nested within another <transaction-block>'s <statement-list>.

GENERAL RULES

1. The normal execution flow for a <transaction-block> is:
   - Start a transaction.
   - Execute the <statement-list>; if the <statement-list> completes with explicit <exit-block> flow control that does not specify ROLLBACK, it does not affect the normal execution flow.
   - Commit the transaction.
   - Complete the <transaction-block> with default flow control.
2. Deviations from the normal execution flow occur when:
   - The <statement-list> completes with explicit <exit-block> flow control that specifies ROLLBACK, then:
     - Roll back the transaction.
     - Complete the <transaction-block> with default flow control.
   - The <statement-list completes with <exit-task> or <go-to> flow control that does not specify ROLLBACK, then:
     - Commit the transaction.
     - Complete the <transaction-block> with the same explicit flow control.
   - The <statement-list> completes with <exit-task> or <go-to> flow control that specifies ROLLBACK, then:
     - Roll back the transaction.
     - Complete the <transaction-block> with the same explicit flow control.

- The <statement-list> completes with an exception; the transaction is rolled back. If the exception type is:
  - Fatal, the <transaction-block> completes with the exception.
  - Transient transaction, then:
    - If the task is restartable and the retry limit has not been exceeded, the <transaction-block> is executed again.
    - Otherwise, if there is an <exception-handler>, the <exception-handler> is executed.
    - Otherwise, the <transaction-block> completes with a permanent transaction exception.
  - Permanent transaction or non-transaction, then:
    - If there is an <exception-handler>, the <exception-handler> is executed.
    - Otherwise, the <transaction-block> completes with a permanent transaction exception.

CONFORMANCE NOTES

1. Multi-resource transactions are not required for Support Levels 1 and 2.

EXCEPTIONS

None.

SEE ALSO

Section 10.3, Section 10.3.11.

10.3.26 <transaction-control>

NAME

<transaction-control> — Explicitly rolls back or commits the current transaction.

SYNOPSIS

```
<transaction-control> ::
      <nil>
    | WITH ROLLBACK [TRANSACTION]
    | WITH COMMIT [TRANSACTION]
```

SYNTAX RULES

None.

GENERAL RULES

1. The default is COMMIT.
2. If COMMIT is specified, any transactional resources modified by tasks within the transaction tree are committed as part of the same transaction.
3. If ROLLBACK is specified, any transactional resources modified by tasks with the transaction tree are rolled back.

EXCEPTIONS

None.

SEE ALSO

None.

10.3.27 <while>

NAME

<while> — Repeats the execution of a <statement-list> based on a condition test.

SYNOPSIS

```
<while> ::
    WHILE <boolean-expression> DO
        <statement-list>
    END [WHILE] <while-end-label>

<while-end-label> ::
      <nil>
    | <label-identifier>
```

SYNTAX RULES

1. The <label-identifier> specified in <while-end-label> must be the same as the statement label on the <while>.
2. If used in a <noncomposable-task> outside a <transaction-block>, <boolean-expression> cannot reference a transactional workspace field.

GENERAL RULES

1. The normal execution flow for a <while> is:
   — Evaluate the <boolean-expression>.
   — If the <boolean-expression> evaluates true, the <statement-list> is executed and the <while> is executed again.
   — If the <boolean-expression> evaluates false, the <while> completes with normal flow control.
2. Deviations from the normal execution flow occur when:
   — Evaluating a <boolean-expression> and one of the conditions listed under EXCEPTIONS occurs; the <while> completes with the exception associated with the condition.
   — The <statement-list> completes with explicit flow control; the <while> completes with the same explicit flow control.
   — The <statement-list> completes with an exception; the <while> completes with that exception.

EXCEPTIONS

The following exception classes can be generated:

1. AP-EXECUTION-FAULT

   An AP-EXECUTION-FAULT class exception is generated for the following exceptions:
   — The subscript used to reference an element in a fixed-length array is invalid (referring to A(n), where n < 1 or n > maximum size of array).
   — The subscript used to reference an element in a varying-length array is invalid (referring to A(n), where n < 1 or n > the value of the depending on field).
   — The value of the depending on field used when referencing an element of a varying-length array is invalid (the value of the depending on field < minimum

size of array or the value of the depending on field > maximum size of array).

— The value of the DEPENDING ON field was invalid when referencing a <workspace-field> containing a varying-length array.

SEE ALSO

Section 4.15, Section 10.3.

Chapter 11

Processing Procedures

11.1 Processing Procedure

A processing procedure is a customer-written program written in COBOL or C that typically performs computations and SQL or file operations. A top-level processing procedure is a procedure called directly by a task. A nested processing procedure is a processing procedure called from a top-level processing procedure or from another nested processing procedure.

Processing procedures can:

- Access both transactional and non-transactional files.
- Access SQL tables.
- Access workspaces when passed to the top-level processing procedure as arguments on the <call-procedure>.
- Access context created by a previously executed processing procedure, if the processing procedure is in the same processing group and executes within the same transaction.
- Create context to be accessed by a subsequently executed processing procedure, if the processing procedure is in the same processing group and executes within the same transaction.
- Call a nested processing procedure.
- Call a task according to the rules defined in Section 13.3 on page 265.

Processing procedures cannot:

- Send data to a display.
- Accept data from a display.

11.2 Top-level Processing Procedure

A top-level processing procedure receives control from the <task-definition>, when the <task-definition> executes a <call-procedure>. A top-level processing procedure:

1. Receives the initial input, if any, through the contents of workspaces passed as arguments on the <call-procedure>.
2. Returns the results of the <call-procedure>. The possible results of a <call-procedure> are:

   Normal Execution — The output arguments are returned through the contents of workspaces passed as arguments on the call.

   Exception — If an exception is returned to the task, the output arguments are undefined.

11.3 Top-level Processing Procedure Arguments

The number, order and type of arguments defined in the top-level processing procedure must match the number, order, and type of arguments for the <processing-procedure> in the <processing-group-specification>.

The processing procedure arguments have the following semantics:

- The arguments can be passed by reference or passed using copy-in, copy-out semantics.
- The argument is passed from the calling <task-definition> to the called top-level processing procedure, if INPUT is specified on the argument within the <processing-group-specification>.
- The argument is passed from the called processing procedure to the calling <task-definition>, if OUTPUT is specified on the argument within the <processing-group-specification>.
- If INOUT is specified on the argument within the <processing-group-specification>, the argument obeys the semantics of both INPUT and OUTPUT.
- The initial value of workspaces passed as OUTPUT is undefined.

11.4 Nested Processing Procedure

A nested processing procedure receives control from the top-level processing procedure or another nested processing procedure, when the top-level processing procedure or nested processing procedure executes a procedure call. A nested processing procedure:

1. Receives the initial input, if any, through the contents of the arguments passed on the procedure call.
2. Returns the results of the procedure call. The possible results of a procedure call are:

   Normal Execution — The arguments are returned to the calling processing procedure.

   Error Returned — All errors in the execution of a nested processing procedure are returned to the calling processing procedure as specified in the COBOL or C standard.

11.5 Nested Processing Procedure Arguments

The mechanism used to pass arguments between procedures written using the same programming language is specified by the programming language standard.

The mechanism used to pass arguments between a procedure written using COBOL and a procedure written using C is defined in Appendix E on page 523. The mechanism used to pass arguments between a procedure written using C and a procedure written using COBOL is defined in Appendix E.

11.6 Context

The scope of context sharing for the execution of processing procedures is a processing group. The processing group context is maintained until the end of the transaction in which the context was created. The processing group context consists of:

- File Context

  Context that is associated with a file descriptor is called file context. File context consists of:

  — File position indicator.

  Specifies the next record to be accessed within a given file during certain input/output operations. The setting of the file position indicator is affected by file operations as defined in standard COBOL and C.

  — For transactional relative and indexed files accessed with dynamic or random mode.

  Access to the current value of any file record previously read, modified, or written in this transaction.

  — For non-transactional relative and indexed files accessed with dynamic or random mode and stream files.

  Access to any data in the file.

  — For C-ISAM files.

  Access to file descriptors.

- SQL Context

  Context that is associated with SQL is called SQL context. SQL context consists of:

  — cursors

  — access to previously modified SQL data within the current transaction

  — access to previously read SQL data within the current transaction.

Implementor Notes:

1. Within a given transaction an implementation must minimally support the sharing of cursors for multiple calls to the same top-level processing procedure and nested processing procedures called by processing procedures from the same task.
2. It is implementation-defined whether or not external variables can be shared among processing procedures.

Processing procedures have access to both file context and SQL context. A processing procedure can only modify the existing context by:

- performing file operations
- performing SQL operations.

Implementor Note:

An implementation need not share file and SQL context when a processing procedure calls a procedure written using another language.

Note: Sequential calls from one task to another task must be performed WITH DEPENDENT WORK in order to share transaction context.

11.7 File Access

File access rules:

1. All relative and sequential file access must be done through COBOL procedures. Standard COBOL statements are used to read, write, and update records in relative and sequential files.
2. Access to indexed files must be done through COBOL or C procedures. Standard COBOL statements are used to read, write, and update records in indexed files; standard ISAM functions are used in C to read, write, and update records in indexed files.
3. All stream file access must be done through C procedures. Standard C functions are used to read, write, and update streams of data in stream files.
4. All files must be opened by the TP system using environmental information except for:
   — Non-transactional sequential files, which can be opened and closed using standard COBOL.
   — Non-transactional stream files, which can be opened and closed using standard C.
5. The TP system must open the files prior to the first file access by a processing procedure.
6. A processing procedure accesses a file using a file descriptor.
7. All transaction demarcation must be left to the task definition.
8. On the first access to a indexed or relative file within the current transaction, file context is undefined.
9. On the first access to a sequential file within the current transaction, the processing procedures can rely on the following file context:
   — The file position at the beginning of the file if the file was opened for INPUT.
   — The file position at the beginning of the file if the file was opened for OUTPUT.
   — The file position at the end of the file if the file was opened for EXTEND.
10. On the first access to a stream file within the current transaction, the processing procedures can rely on the following file context:
    — The file position at the beginning of the file if the file was opened for READ.
    — The file position at the end of the file if the file was opened for APPEND.
11. When the transaction is complete (committed or rolled back), all context maintained within the processing group for that transaction is undefined.
12. The results of concurrently accessing a non-transactional sequential file opened using standard COBOL or a stream file opened using standard C are undefined.

Implementor Note:

An implementation can require that updates be made only on the most recently read record for a file descriptor.

11.8 SQL Access

SQL access rules:

1. Processing procedures use embedded SQL statements, including explicit cursor control, to access a collection of data.
2. The first SQL statement in the processing procedure causes the SQL environment to be implicitly associated with the processing group.
3. All transaction demarcation must be left to the task definition. SQL COMMIT and ROLLBACK statements are not allowed in the processing procedure.
4. When the transaction is complete (committed or rolled back), all context maintained in SQL for the processing group for that transaction is undefined.

Implementor Note:

Within a transaction, an implementation can restrict SQL access to C or COBOL but not both.

The following SQL operations are not supported in processing procedures:

- Diagnostics management
- Connection management
- Session management
- Schema definition
- Schema manipulation.

11.9 File Transfer

Standard XFTAM functions can be performed from processing procedures on files previously opened and closed by C or COBOL procedures.

Both the source and target files must be of the same file type.

11.10 Error Handling Rules

Error handling rules:

1. All errors in the execution of a file operation in COBOL and C that do not stop the transaction from completing are returned to the processing procedure, as specified in the COBOL or C standard.
2. All errors in the execution of SQL statements are returned to the processing procedure, as specified in the SQL standard.
3. All errors in the execution of a processing procedure are returned to the calling processing procedure, as specified in the COBOL or C standard.
4. All non-transaction errors in the execution of a task called by a processing procedure are returned to the procedure. Transaction errors are returned to the task that called the top-level processing procedure that caused the task to be invoked.
5. All errors in the execution of file I/O or SQL statements that stop the transaction from completing are returned to the processing procedure, as specified in the SQL standard, or a transaction exception can optionally be raised by the TP system instead.

A top-level processing procedure generates an exception by setting external variables. See Section 14.2 on page 270 and Section 15.2 on page 280 for more information on setting the external variables in COBOL and C.

1. A top-level processing procedure generates non-transaction and transaction exceptions by setting either the exception code or class.
2. If a top-level processing procedure generates an exception, then the output arguments of the procedure are undefined in the task definition.

Chapter 12

Presentation Procedures

12.1 Send Presentation Procedure

12.1.1 General Rules

The send presentation procedure:

1. Accepts data from a <call-presentation-procedure> statement. Can associate static data and specific operations or functions.
2. Can access data stored by a presentation procedure previously executed within the same presentation group for the same display.
3. Can store data to be accessed by a subsequently executed presentation procedure within the same presentation group for the same display.
4. Can send data to the display.
5. Can accept data from the display.
6. Can access static data defined within the send presentation procedure.
7. Can return exceptions to the task.

12.1.2 Transactional Send Procedure

The following rules are specific to a transactional send presentation procedure. The transactional send presentation procedure:

- Replace General Rule 7. with: Does not return exceptions to the task. Exceptions generated by transactional send procedures are handled by the TP system.

12.1.3 Broadcast Send Procedure

The following rules are specific to a broadcast send presentation procedure. The broadcast send presentation procedure:

- Can access a display concurrently with another presentation procedure.
- Replace General Rule 2. with: Cannot access data stored by a previously executed presentation procedure.
- Replace General Rule 3. with: Cannot store data for access by a subsequently executed presentation procedure.
- When sending data to a display, the state of the display determines when the data is received by the display:
  - — If a read operation is currently being executed to the display, the broadcast operation executes without interfering with the read operation.
  - — If a write operation is currently being executed, the broadcast operation executes without preventing the write operation from completing.

— If there is no outstanding operation for the display, the broadcast data is sent.

- Replace General Rule 5. with: Cannot accept data from the display.
- Replace General Rule 7. with: Does not return exceptions to the task. Exceptions generated by broadcast send procedures are handled by the TP system.

12.2 Receive Presentation Procedure

12.2.1 General Rules

The receive presentation procedure:

1. Returns data to a <call-presentation-procedure> statement.
2. Can associate static data and perform specific operations or functions.
3. Can access data stored by a presentation procedure previously executed within the same presentation group for the same display.
4. Can store data to be accessed by a subsequently executed presentation procedure within the same presentation group for the same display.
5. Can send data to the display.
6. Can accept data from the display.
7. Can access static data defined within the receive presentation procedure.
8. Can return exceptions to the task.

12.3 Transceive Presentation Procedure

12.3.1 General Rules

The transceive presentation procedure:

1. Accepts data from a <call-presentation-procedure> statement.
2. Returns data to a <call-presentation-procedure> statement.
3. Can associate static data and perform specific operations or functions.
4. Can access data stored by a presentation procedure previously executed within the same presentation group for the same display.
5. Can store data to be accessed by a subsequently executed presentation procedure within the same presentation group for the same display.
6. Can send data to the display.
7. Can accept data from the display.
8. Can access static data defined within the transceive presentation procedure.
9. Can return exceptions to the task.

12.4 Presentation Context

Presentation context is data that can be shared between the execution of one presentation procedure and the subsequent execution of another presentation procedure.

Presentation context consists of static and dynamic data. Static data is defined by the presentation procedure and cannot be modified. Dynamic data is obtained from the task and the display. Subsequent input operations from the display or additional data sent from the task can modify the dynamic data.

Presentation context can be shared between presentation procedures for a given display, if both procedures are contained within the same presentation group.

A TP system must maintain at least one presentation context per display.

See also Section 12.5.1 on page 261.

12.5 Customer-written Presentation Procedures

A customer can use standard COBOL or C to write a presentation procedure.

The <presentation-group-specification>:

- Specifies arguments to be accepted from the task.
- Specifies arguments to be returned to the task.
- Specifies an optional initialization procedure to establish any initial context.
- Specifies a optional termination procedure to destroy context no longer required.

12.5.1 Context

The presentation initialization procedure defined using the <presentation-group-specification> is invoked to establish any initial context needed by the customer-written presentation procedures.

The presentation initialization procedure is implicitly invoked for each display by the TP system before any presentation procedure within the specified presentation group is executed when presentation group context is established. The TP system passes the display name and the default language as text strings to the presentation initialization procedure.

The presentation initialization procedure returns a presentation group handle. The *presentation group handle* is an integer value that is passed to all presentation procedures executed on behalf of this display within the presentation group. Presentation procedures can use the presentation group handle to access context established by the presentation initialization procedure, or to store context to be used by another presentation procedure. The presentation termination procedure is invoked implicitly by the TP system once the presentation context is no longer required.

Customer-written presentation procedures are responsible for retaining any required context. An implementation must supply a mechanism for storing presentation context between presentation procedure executions.

12.6 Vendor-supplied Forms

A vendor-supplied forms system can be used to implement the presentation procedures.

12.7 Initialization and Termination Procedures

The presentation initialization procedure has the following arguments:

- Display-name — read-only, 255-character text string.

  Allows the initialization routine to access the display. Displays are associated using environmental information in an implementation-specific manner.

- Client-language — read-only, 255-character text string.

  Names the language passed to the task on the task call.

- Presentation-group-handle — write-only, integer.

  A customer-generated identifier by which the procedures in the presentation group share context.

The presentation termination procedure has the following argument:

- Presentation-group-handle — read-only, integer.

The following list of arguments is passed to the presentation procedure specified by the <presentation-procedure-identifier> in the send procedure:

- The presentation-group-handle — read-only, integer.

  This is the value returned by the initialization procedure. If there is no initialization procedure, the value of the handle is undefined.

- The records in the order defined in the <presentation-procedure-send-argument-list>.

The following list of arguments is passed to the presentation procedure specified by the <presentation-procedure-identifier> in the receive procedure:

- The presentation-group-handle — read-only, integer.

  This is the value returned by the initialization procedure. If there is no initialization procedure, the value of the handle is undefined.

- The records in the order defined in the <presentation-procedure-receive-argument-list>

The following list of arguments is passed to the presentation procedure specified by the <presentation-procedure-identifier> in the transceive procedure:

- The presentation-group-handle — read-only, integer.

  This is the value returned by the initialization procedure. If there is no initialization procedure, the value of the handle is undefined.

- The records in the order defined in the <presentation-procedure-transceive-argument-list>.

Chapter 13

Client Programs

13.1 Customer-written Client Programs

Customer-written client programs are customer-written programs that execute inside of the TP system environment and call tasks within the same TP system environment. Customer-written client programs can be written using standard C or COBOL.

Customer-written client programs can call non-composable tasks that execute on the same TP system as the customer-written client program. A customer-written client program uses a client stub procedure to call a non-composable task. Client stub procedures are generated from the <task-group-specification>. See Section 13.5 on page 267 for more information on client stub procedures.

The customer-written client program performs a procedure call to the client stub procedure. A client stub procedure call is a procedure invocation in which the customer-written client program:

1. Supplies the initial input, if any, using arguments passed on the call to the client stub procedure.
2. Uses the natural language attribute from the user profile to set the natural language.
3. Sets the default display name.
4. Waits for the client stub procedure to complete before continuing execution.
5. Receives the final results of the client stub procedure execution, if any. The results can be either:

Normal Completion	Output arguments are returned to the client program.
Exception	Output arguments are undefined.

13.1.1 Exceptions

A customer-written client program uses the same external variable mechanism as presentation and processing procedures for exception handling. Instead of setting the external variables to generate an exception, the customer-written client program must examine the exception variables upon completion of the client stub procedure call. If exception class contains a value of zero, then no exception was returned. If exception class contains a non-zero value, then an exception is returned. See Section 14.2 on page 270 and Section 15.2 on page 280 for defining external exception variables using COBOL and C.

A customer-written client program can only call non-composable tasks. Therefore, only non-transaction exceptions can be returned from the execution of a task. Appendix C on page 489 lists the exception classes that can be returned to the customer-written client program.

13.2 Menu Client Programs

A menu client program has the same semantics as the client program defined in Section 3.3.1 on page 47. A menu client program is a vendor-supplied client program that:

1. Allows for the selection of a task from a display.
2. Uses the natural language attribute from the user profile to set the natural language.
3. Sets the default display name.
4. Invokes the task selected.
5. Reports exceptions from the invoked task to the display.

Tasks invoked by a menu client program must:

- be non-composable
- have no task arguments
- have no text fields passed by a terminal user at task selection time.

An implementation must provide a mechanism to allow a user to request the cancellation of a task started by a menu client.

13.3 Processing Procedures

A processing procedure can call tasks that are composable, non-composable, on the same TP system, or on a remote TP system. A processing procedure uses a client stub procedure to call a task. Client stub procedures are generated from the <task-group-specification>. See Section 13.5 on page 267 for more information on client stub procedures.

The processing procedure performs a procedure call to the client stub procedure. A client stub procedure call is a procedure invocation in which the processing procedure:

1. Supplies the initial input, if any, using arguments passed on the call to the client stub procedure.
2. Uses the natural language attribute from the user profile to set the natural language.
3. Sets the default display name to null.
4. Waits for the client stub procedure to complete before continuing execution.
5. Receives the final results of the client stub procedure execution, if any. The results can be either:

   Normal Completion — Output arguments are returned to the processing procedure.

   Non-transaction Exception — Output arguments are undefined and control is returned to the processing procedure .

   Transaction Exception — Output arguments are undefined and control is returned to the task that performed the top-level processing procedure call that resulted in an exception.

A processing procedure uses the external variable mechanism for exception handling described in Section 14.2 on page 270 for COBOL and Section 15.2 on page 280 for C.

When control is returned to a processing procedure when a non-transaction exception occurs in the called task, the processing procedure must examine the exception variables upon completion of the client stub procedure call. If exception class contains a value of zero, then no exception was returned. If exception class contains a non-zero value, then an exception is returned.

13.4 External Clients

An external client program can call non-composable tasks. An external client program uses a client stub procedure to call a non-composable task in a TP system. Client stub procedures are generated from the <task-group-specification> and made available to an external client program using an implementation-specific mechanism. See Section 13.5 on page 267 for more information on client stub procedures.

The client stub procedure is executed by the external client program the same way any other function or procedure is executed within a standard programming language. The client stub procedure call is a procedure invocation in which the external client program:

1. Supplies the initial input, if any, using arguments passed on the call to the client stub procedure.
2. Sets the natural language attribute.
3. Sets the default display name to null.
4. Waits for the client stub procedure to complete before continuing execution.
5. Receives the final results of the client stub procedure execution, if any. The results can be either:

   Normal Execution — Output arguments are returned to the client program.

   Exception — Output arguments are undefined.

13.4.1 Exceptions

An external client program uses the same external variable mechanism as presentation and processing procedures and customer-written client programs for exception handling. Instead of setting the external variables to generate an exception, the external client program must examine the exception variables upon completion of the external client stub procedure call. If exception class contains a value of zero, then no exception was returned. If exception class contains a non-zero value, then an exception is returned. See Section 14.2 on page 270 and Section 15.2 on page 280 for defining external exception variables using COBOL and C.

An external client program can only call non-composable tasks. Therefore, only non-transaction exceptions can be returned from the execution of a task. Appendix C on page 489 lists the exception classes that can be returned to the external client program.

13.5 Stub Procedures

1. A client stub procedure is called from a customer-written client program, external client program, or processing procedure to invoke a task. A client stub procedure:

   — Is called from the customer-written client program, external client program, or processing procedure using a procedure call.

   — Receives the initial input, if any, through the arguments passed on the procedure call to the client stub procedure.

   — Invokes the task from which the client stub procedure was produced, and passes arguments between the client stub procedure and the invoked task. The task executes in the task group from which the client stub procedure was produced.

   — Waits for the invoked task to complete and returns the results, if any, to the customer-written client program, external client program, or processing procedure.

2. A client stub procedure can be produced by an implementation-specific mechanism for each <noncomposable-task-interface> and <composable-task-interface> included in the <task-group-specification> The name of the client stub procedure is the name of the <noncomposable-task-interface> or <composable-task-interface> from which it was produced.

3. An implementation-specific mechanism can be used to change the name of the client stub procedure to differentiate tasks with the same name in different task groups.

4. A C procedure must be able to call a client stub procedure using a pointer.

5. An implementation must provide a mechanism for transferring a client stub procedure to an external client.

6. The argument list for a client stub procedure is:

   — Arguments defined in <task-interface-arguments> for the task.

Chapter 14

Use of COBOL

This chapter describes how to use standard COBOL to write procedures for use in TP applications. The following procedures can be written using COBOL:

- processing procedures
- customer-written presentation procedures
- customer-written client programs.

14.1 Data Type Mapping Information

The following procedures pass data between an STDL <task-definition> and a COBOL procedure:

- top-level processing procedure
- customer-written presentation procedures
- customer-written client programs.

See Appendix E on page 523 for further information on data type mapping and inter-language call semantics.

14.2 Exception Information

14.2.1 Raising Exceptions

A top-level processing procedure or customer-written presentation procedure can raise an exception to the task that called the procedure by performing the following sequence:

1. Setting either the ECLASS or ECODE field of the EINFO external variable:
   — If the ECLASS field is set, it must contain a system-defined class value.
   — If the ECODE field is set, it must contain an application-defined message code that has been defined using a <message-definition>.
   — If both are set, the TP system uses the ECODE.
2. Optionally set the EPROC and EPGROUP fields to indicate where the exception occurred.
3. If ECODE is set, then ECGROUP must be set if the <message-group-definition> in which ECODE appears specifies a UUID. Otherwise, the results are undefined.
4. Exiting the top-level processing or presentation procedure that was called by the task.

The TP system examines the EINFO external variable whenever a COBOL top-level processing or presentation procedure exits. The TP system initializes the fields of EINFO before calling the top-level processing or presentation procedure. ECLASS and ECODE are set to zero. EPROC and EPGROUP are set to spaces. If either ECLASS or ECODE is set to a nonzero value by the top-level processing or presentation procedure, then the TP system raises an exception in the task for the application as follows:

- The exception source is set to 1, for application.
- If ECODE is set:
  — If the ECGROUP field is not set, the TP system validates ECODE to check that the exception code corresponds to a message code within any message group that does not have a UUID and that is available on the TP system.
  — If the ECGROUP field is set, the TP system validates ECODE to check that the exception code corresponds to a message code within the message group indicated by the ECGROUP field.
  — If the exception code is valid, the TP system raises an exception with the exception class defined for the message code and given exception code.
  — If the exception code is not valid, the TP system raises an exception class of AP-EXECUTION-FAULT.
  — If the ECLASS is also set and the exception class defined does not match the exception class associated with the given exception code, then the TP system raises an exception class of AP-EXECUTION-FAULT.
- If ECODE is not set:
  — The TP system validates the exception class set by ECLASS.
  — If the exception class is valid, the TP system raises an exception class and an exception code of 0.
  — If the exception class is not valid, the TP system raises an exception class of AP-EXECUTION-FAULT.

The TP system operation on the output arguments is undefined when a top-level processing or presentation procedure raises an exception.

The COBOL definition for the EINFO external variable is derived from converting the data type definition shown in Figure 14-1 according to the rules in Appendix E on page 523.

```
TYPE EINFO IS RECORD
    ECLASS INTEGER;
    ECODE INTEGER;
    EPROC TEXT CHARACTER SET SIMPLE-LATIN SIZE 32;
    EPGROUP TEXT CHARACTER SET SIMPLE-LATIN SIZE 32;
    ESOURCE INTEGER;
    ECGROUP UUID;
END RECORD;
```

Figure 14-1 EINFO Data Type Definition

14.2.2 Examining Exceptions Returned

A customer-written client program can receive an exception from an invoked task. When an exception is raised, the customer-written client program can examine the exception data in the EINFO external variable. The format of the EINFO variable is defined in Figure 14-1. If ECLASS contains a value of zero, then no exception is returned. If ECLASS contains a non-zero value, then an exception is returned. If an exception is returned, the customer-written client program can examine ESOURCE to determine whether the exception raised was generated by the TP system or by the application.

14.2.3 Exception Information COPY Files

An implementation must provide the following COBOL COPY files for processing procedures, customer-written presentation procedures, and customer-written client programs written in COBOL:

EINFO — The EINFO data type definition can be copied into source files using the text-name TEINFO.

ECLASS — The ECLASS exception class definition with its exception class value as specified in Section A.2 on page 486 can be copied into source files using the text-name TECLASS (see Section 14.3 on page 272).

14.3 Processing Procedures

14.3.1 File Access

Standard COBOL statements are used to read, write, and update records in indexed, relative, and sequential files. The COBOL source program must:

- Include a FILE-CONTROL paragraph in the INPUT-OUTPUT SECTION with an entry for each file used in the procedure. For each file, the file control statement can contain the following clauses:

  — SELECT — required.

  — ASSIGN — required, may have to be edited when porting processing procedures.

  — ORGANIZATION — required if the file is not SEQUENTIAL.

  — RECORD KEY — required if the file is indexed.

  — RELATIVE KEY — required if the file is relative organisation.

  — FILE STATUS

  — RESERVE — optional.

  — ACCESS MODE — required.

  No other clauses can be used.

- Include a FILE SECTION with a file descriptor for each file used in the procedure. For each file descriptor the following clauses can be used:

  — FD with the file-name identifier — required.

  The file-name must be mapped to the file descriptor specified for the processing procedure using implementation-specific environmental information (see Section 17.1.2.1 on page 295 for further information).

  — EXTERNAL — required.

  — BLOCK CONTAINS — optional.

  — RECORD — optional.

  — ALTERNATE RECORD KEY — optional.

  No other clauses can be used.

Additional environmental information may not be required in order to specify the FILE-CONTROL and FILE SECTION in a COBOL procedure.

14.3.2 Restrictions

The restrictions on standard COBOL used to develop processing procedures are that the COBOL procedure:

- Cannot include an OPEN statement, except for a non-transactional sequential file. A non-transactional sequential file can be opened either by the TP system or by a COBOL procedure.
- Cannot include an CLOSE statement, except for a non-transactional sequential file. A non-transactional sequential file can be closed either by the TP system or by a COBOL procedure.

- Cannot include a STOP RUN statement.
- Can include SORT/MERGE statement only for non-transactional sequential files first opened and then closed by a COBOL procedure.
- Cannot rely on the initial value (if any) of an element of the WORKING STORAGE SECTION after the element is modified during the execution of the current procedure.
- Cannot rely on initial values contained in WORKING STORAGE SECTION (if any) when a preceding execution of the procedure modifies them. If no modifications are made to WORKING STORAGE SECTION, values maintain initial state if any was specified.
- Cannot share common variables, except that processing procedures can share the EINFO external variable defined in Figure 14-1 on page 271. The restriction does not apply to FD names defined in the FILE SECTION for file descriptors, or any external variables associated with an FD.
- Cannot contain user-defined words or system-names with the same name as a <class-identifier>s contained within the ECLASS copy file. This restriction only applies if the ECLASS copy file was included in the COBOL program.
- Cannot contain user-defined words or system-names with the same name as the fields defined within the EINFO copy file. This restriction only applies if the EINFO copy file was included in the COBOL program.

14.4 Top-level Processing Procedure

14.4.1 Arguments

Arguments can be passed to COBOL top-level processing procedures. The LINKAGE SECTION describes the data items to be used. The COBOL copy file generated from the <data-type-definition> can be used to declare the arguments in the COBOL top-level processing procedures. The Procedure Division Header must have a USING phrase that specifies the arguments.

The number, order and type of arguments defined in the COBOL top-level processing procedure must match the number, order, and type of arguments defined in the <processing-procedure-interface> defined in the <processing-group-specification>.

14.5 Presentation Procedures

14.5.1 Arguments

Arguments can be passed to COBOL presentation procedures. The LINKAGE SECTION describes the data items to be used. The COBOL copy file generated from the <data-type-definition> can be used to declare the arguments in the COBOL presentation procedures. The Procedure Division Header must have a USING phrase that specifies the arguments.

The number, order and type of arguments defined in the COBOL procedure must match the number, order, and type of arguments defined in the <presentation-group-specification>. The arguments of a presentation procedure are defined in Section 12.7 on page 262.

A presentation procedure includes the following additional argument: presentation group handle.

14.5.2 Restrictions

The following restrictions are placed on standard COBOL used to develop presentation procedures:

- Cannot access transactional resources Because presentation procedures do not execute within a transaction, they are unable to access transactional resources.
- Cannot include a STOP RUN statement
- Cannot include SORT/MERGE statement
- Cannot rely on values contained in working storage when the presentation procedure is initially invoked from the task.
- Cannot share common variables with other procedures.
- Cannot contain user-defined words or system-names with the same name as a <class-identifier>s contained within the ECLASS copy file. This restriction only applies if the ECLASS copy file was included in the COBOL program.
- Cannot contain user-defined words or system-names with the same name as the fields defined within the EINFO copy file. This restriction only applies if the EINFO copy file was included in the COBOL program.

14.6 STDL Identifier to COBOL Word Conversion

When converting an STDL identifier into a COBOL word, use the following rules:

- If the identifier contains lower-case Latin characters, convert them to upper-case.
- Replace underscores with hyphens.
- No other characters are changed.
- If the STDL identifier is 31 characters in length, truncate the thirty-first character to produce a COBOL word of 30 characters in length.

14.7 Generated COBOL COPY Files

An implementation must provide a way to generate the following COBOL COPY files for processing procedures, customer-written presentation procedures, and customer-written client programs written using COBOL:

- Message COPY File

  Creates a COBOL record definition that contains the message group UUID and the message code definitions for the messages in the message group source file. The format of the record definition is as follows. The 01 level item is the message group name:

  ```
  01 <message-group-identifier>
  ```

  <message-group-identifier> is converted to a COBOL word using the rules in Section 14.6 on page 276.

  — A subordinate item for the message group UUID:

  ```
  02 UUID
  ```

  A message group UUID is converted to a COBOL array of 4 items, each with a data type of PIC S9(9) USAGE BINARY and values that initialize the UUID field to the UUID of the message group. The actual numbers depend on the host computer's architecture.

  — The subordinate items are the message values, one for each message defined in the message group:

  ```
  02 <message-identifier> PIC S9(9) USAGE BINARY
  VALUE IS <message-number>
  ```

  Each <message-identifier> is converted to a COBOL word using the rules in Section 14.6 on page 276.

  Messages contained in a message group source file are translated into a COPY file, the name of which is based on the message group source file name.

  An implementation can enforce a read-only use of message group definitions in the COPY file. In that case message group definitions are exempt from the restrictions in Section 14.3.2 on page 272.

  Two message source files with the same name cannot be translated into the same COPY file.

- Data Type Definition COPY File

  Creates a COBOL formatted version of any <data-type-definition>s in a source file. For a source file containing one or more record definitions, the translation process produces a single file containing COBOL definitions that correspond to the definitions in the source file. The mapping of STDL data types to COBOL data types is referenced Section 14.1 on page 269.

  When creating a COBOL COPY file, a <data-type-definition> is processed as follows:

  — The <data-type-identifier> of the <data-type-definition> is used as the 01 level-number data item. <data-type-identifier> is converted to a COBOL word using the rules in Section 14.6 on page 276.

  — Each <field-definition> within a <record-type> is assigned a level-number one greater than the record in which the <field-definition> appears.

  — The order of the data items in the COBOL COPY file must match the order of the <field-definition>s in the <record-type>.

— A <field-definition> of type ARRAY is assigned a COBOL level-number and an OCCURS clause specifying the array size. The level-number for the array is one greater than the record in which the <field-definition> of type ARRAY appears, and the level-number for an additional dimension is one greater than the level-number for the ARRAY in which the current <field-definition> appears.

— A <field-definition> defined as an elemental array field is assigned a level-number one greater than the <field-definition> to which it is subordinate.

— A <field-definition> of type UUID is defined as an array of 4 items, each with a data type of PIC S9(9) USAGE BINARY.

A vendor must support a mechanism for passing the contents of a <field-definition> of type UUID to the corresponding COBOL data item.

— When a <data-type-identifier> appears as a <field-definition>, the data items subordinate to the <data-type-identifier> are assigned level numbers one greater than the level number for the <data-type-identifier>.

— Any comment text in the data type definition must appear in the generated COBOL COPY file. The format is implementation-specific.

Records contained in a source file are translated into a COPY file, the name of which is based on the source file name.

Chapter 15

Use of C

This chapter describes how to use standard C to write procedures for use in TP applications. The following procedures can be written using C:

- processing procedures
- customer-written presentation procedures
- customer-written client programs.

15.1 Data Type Mapping Information

The following procedures pass data between an STDL <task-definition> and a C procedure:

- top-level processing procedures
- customer-written presentation procedures
- customer-written client programs.

See Appendix E on page 523 for further information on data type mapping and inter-language call semantics.

15.2 Exception Information

15.2.1 Raising Exceptions

A top-level processing procedure or customer-written presentation procedure can raise an exception to the task that invoked the procedure by performing the following sequence:

1. Setting either the eclass or the ecode field of the einfo external variable:
   — If the eclass field is set, it must contain a system-defined class value.
   — If the ecode field is set, it must contain an application-defined message code that has been defined using a <message-definition>.
   — If both are set, the TP system uses the ecode field.
2. Optionally setting the eproc and epgroup fields to indicate where the exception occurred.
3. If ecode is set and the <message-group-definition> in which it appears specifies a UUID, then ecgroup must be set. If not, the results are undefined.
4. Exiting the top-level processing or presentation procedure that was called by the task.

The TP system examines the einfo external variable whenever a top-level processing or presentation procedure written in C exits. The TP system initializes the fields of einfo before calling the top-level processing or presentation procedure. eclass and ecode are set to zero. eproc and epgroup are set to spaces. If either eclass or ecode is set to a nonzero value by the top-level processing or presentation procedure, then the TP system raises an exception in the task for the application as follows:

- The exception source is set to 1, for application.
- If ecode is set:
  — If the ecgroup field is not set, the TP system validates ecode to check that the exception code corresponds to a message code within any message group that does not have a UUID and that is available on the TP system.
  — If the ecgroup field is set, the TP system validates ecode to check that the exception code corresponds to a message code within the message group indicated by the ecgroup field.
  — If the exception code is valid, the TP system raises an exception with the exception class defined for the message code and given exception code.
  — If the exception code is not valid, the TP system raises an exception class of AP-EXECUTION-FAULT.
  — If the eclass is also set and the exception class defined does not match the exception class associated with the given exception code, then the TP system raises an exception class of AP-EXECUTION-FAULT.
- If ecode is not set:
  — The TP system validates the exception class set by eclass.
  — If the exception class is valid, the TP system raises an exception class and an exception code of 0.
  — If the exception class is not valid, the TP system raises an exception class of AP-EXECUTION-FAULT.

The TP system operation on the output arguments is undefined if the top-level processing or presentation procedure raises an exception.

The C definition for the einfo external variable is derived from converting the data type definition shown in Figure 15-1 according to the rules in Section 15.6 on page 286 and in Appendix E on page 523.

```
TYPE EINFO IS RECORD
    ECLASS INTEGER;
    ECODE INTEGER;
    EPROC TEXT CHARACTER SET SIMPLE-LATIN SIZE 32;
    EPGROUP TEXT CHARACTER SET SIMPLE-LATIN SIZE 32;
    ESOURCE INTEGER;
    ECGROUP UUID;
END RECORD;
```

Figure 15-1 EINFO Data Type Definition

15.2.2 Examining Exceptions Returned

Customer-written client programs can receive exceptions from tasks invoked. When an exception is raised, the customer-written client program can examine the exception data in the variable einfo. The format of the einfo variable is defined in Figure 15-1. If eclass contains a value of zero, then no exception was returned. If eclass contains a nonzero value, then an exception was returned. If an exception is returned, the customer-written client program can examine esource to determine whether the exception raised was generated by the TP system or by the application.

15.2.3 Exception Information Header Files

An implementation must provide the following header files for processing procedures, customer-written presentation procedures, and customer-written client programs written in C:

einfo.h Contains the einfo data type definition defined in Figure 15-1.

eclass.h Equates the symbolic name (<class-identifier>) for each exception class with its exception class value as specified in Section A.2 on page 486. Each <class-identifier> is defined using the following C definition:

```
#define <class-identifier> <exception-class-value>
```

Each <class-identifier> is converted to a C identifier using the rules in Section 15.6 on page 286.

15.3 Processing Procedures

15.3.1 File Access

Standard C direct I/O functions are used to read, write and update records in stream files. For files specifies by the TP system, the C source program uses the file descriptor name defined within environmental information to access the stream file, by associating a stream with the file descriptor.

For files specifies by the TP system, the C source program must declare the file descriptor as a pointer to a FILE structure. FILE is an object type capable of recording all the information needed to control a stream, including its file position indicator, a pointer to its associated buffer (if any), an error indicator that records whether a read/write error has occurred, and an end-of-file indicator that records whether the end of the file has been reached. The FILE structure must be defined extern.

Standard ISAM I/O functions are used to read, write, and update records in indexed files. The C source program uses the file descriptor name defined within environmental information to access the indexed file by associating an indexed file with the file descriptor. C-ISAM "isfds" are defined as external variables for access by the platform linker. isfds must be defined extern.

15.3.2 Restrictions

- The restrictions on standard C used to develop processing procedures are that the C procedure:
  - Cannot use an object whose identifier is defined with file scope, except for the following:
    - processing procedures can share the einfo external variable defined in Figure 15-1 on page 281
    - external variables declared for file descriptors
    - C-ISAM external variables:
      - iserrno
      - isstat1, isstat2
      - isrecnum.
    - identifiers defined using const and not volatile.
- Cannot declare or define user defined identifiers with the same name as the <class-identifier>s contained within the eclass.h header file. This restriction only applies if the eclass.h header file is included in the C program.
- Cannot declare or define user defined identifiers with the same name as the identifiers defined by the einfo.h header file. This restriction applies only if the einfo.h header file is included in the C program.

Any memory allocated during the execution of a processing procedure must be deallocated before the top-level processing procedure terminates. An object whose identifier is defined with file scope is either a variable defined as extern or a variable defined as static that is shared among nested functions within a translation unit defined in standard C. See ISO C for further information on identifiers, objects and file scope.

The following C functions are not allowed in processing procedures:

<setjmp.h> setjmp, longjmp.

<signal.h> signal, raise.

<stdlib.h> calloc, free, malloc, realloc, abort, atexit, exit, getenv, system.

Implementor Note:

A server is not required to support the following <stdio.h> functions: remove, rename, tmpfile, tmpnam, fflush, setbuf, fprintf, fscanf, printf, scanf, vfprintf, vprintf, fgetc, fgets, fputc, fputs, getc, getchar, gets, putc, putchar, puts, ungetc, fread, fwrite, fgetpos, fseek, fsetpos, ftell, rewind, clearerr, feof, ferror and perror.

The following operations are not supported when accessing C-ISAM files:

- isaddindex()
- isbuild()
- isclose()
- isdelindex()
- iserase()
- isopen()
- isrelease()
- isrename()
- isunlock()
- islock().

15.4 Top-level Processing Procedures

15.4.1 Arguments

Arguments are passed to top-level C processing procedures from tasks using pointers. The C include file generated from the <data-type-definition> can be used to declare the arguments in the C processing procedures. Procedure names that are defined in C must be functions of type void.

The number, order and type of arguments defined in the C procedure must match the number, order and type of arguments defined in the <processing-procedure-interface> defined in the <processing-group-specification>.

15.5 Presentation Procedures

15.5.1 Arguments

Arguments are passed to C presentation procedures from tasks using pointers. The C include file generated from the <data-type-definition> can be used to declare the arguments in the C presentation procedures. Procedure names that are defined in C must be functions of type void.

The number, order and type of arguments defined in the C procedure must match the number, order, and type of arguments defined in the <presentation-group-specification>. The arguments of a presentation procedure are defined in Section 12.7 on page 262. A presentation procedure includes the following additional argument: presentation group handle.

Text strings within structures (records) are not null terminated.

15.5.2 Restrictions

The following restrictions are placed on standard C used to develop presentation procedures:

- Cannot access transactional resources.

  Presentation procedures do not execute within transactions and therefore are unable to access transactional resources.

- Cannot share common variables with other procedures.
- Cannot declare or define user defined identifiers with the same name as the <class-identifier>s contained within the eclass.h header file. This restriction applies only if the eclass.h header file was included in the C program.
- Cannot declare or define user-defined identifiers with the same name as the identifiers defined by the einfo.h header file. This restriction applies only if the einfo.h header file is included in the C program.

15.6 STDL to C Identifier Conversion

When converting an STDL identifier into a C identifier, use the following rules:

- If the identifier contains upper-case Latin characters, convert them to lower-case.
- Replace hyphens with underscores.
- C does not support Kanji characters in identifiers.
- No other characters are changed.

15.7 Generated C Header Files

An implementation must provide a way to generate the following C header files for processing procedures, customer-written presentation procedures, and customer-written client programs written using C:

- Message Include File

  Creates an intialized structure variable for the message group which contains the message group UUID and the message code definitions for the messages in the message group. The format of the initialized structure definition is as follows:

```
struct
{
uuid_t uuid;
long int <message-identifier>; . . .
} <message-group-identifier> = {<uuid-value>,
<message-number>, . . .};
```

  — The structure variable has the name of the message group. The <message-group-identifier> is converted to a C identifier using the rules in Section 15.6 on page 286.

  — The last structure member is the UUID of the message group. uuid_t is either the DCE UUID definition or an implementation-defined mapping of a UUID. Whether the uuid_t is the DCE UUID definition or an implementation-defined mapping of a UUID, interoperation must be preserved.

  — Each <message-identifier> structure member corresponds to each message defined in the message group. The name of a member is <message-identifier>, and the value is <message-number>.

  Each <message-identifier> is converted to a C identifier using the rules in Section 15.6 on page 286.

  Messages contained in a message group source file are translated into a C include file, the name of which is based on the message group source file name.

- Data Type Definition Include File

  Creates a C formatted version of any <data-type-definition>s in a source file. For a source file containing one or more data type definitions, the translation process produces a single file that contains C definitions corresponding to the definitions in the source file. The mapping of STDL data types to C data types is referenced in Section 15.1 on page 279.

  The translated format of a <data-type-definition> is as follows:

  — The <data-type-identifier> of the <data-type-definition> is used as the name of the C structure tag.

  — Each <field-definition> within a <record-type> is a named member of the structure variable corresponding to the record in which the <field-definition> appears.

  — The order of the members of the structure variable within the C include file must match the order of the <field-definition>s in the <record-type>.

  — A <field-definition> that refers to a previously-defined <data-type-identifier> is as follows:

```
struct <data-type-identifier> <field-identifier>.
```

— An implementation is required to provide a typedef for a UUID data type for the include file.

— Any comment text in the data type definition must appear in the generated C header file. The format is implementation-specific.

C identifiers are derived from the corresponding STDL identifiers using the rules in Section 15.6 on page 286.

Records contained in a source file are translated into a C include file, the name of which is based on the source file name.

- Task Group Definition Include File

  Contains a C external <task-identifier> function declaration for each task in a <task-group-specification> according to the rules listed in Section 13.5 on page 267. This include file is used by client programs. Task function declarations return a type of void, and all arguments are passed as pointers. <task-identifier>s are converted into function declarations using the rules in Section 15.6 on page 286.

  The task group specification in the task group source file is translated into a function declaration include file, the name of which is based on the task group source file name.

Structured Transaction Definition Language (STDL)

Part 2:

STDL Environment, Execution and Protocol Mapping Specification

X/Open Company Ltd.

Chapter 16

Introduction to Part 2

16.1 Scope

Part 2, STDL Environment, Execution and Protocol Mapping Specification defines the basic concepts and facilities that must be implemented by a conforming implementation in order to support STDL applications. This includes:

- environmental management facilities
- mapping of the STDL semantics to the RTI protocol in the referenced X/Open TxRPC specification and Appendix H on page 533.

16.2 Organisation of Part 2

This part is organised as follows:

- Chapter 16 defines the scope of this part, the notations and conventions used, and discusses general conformance criteria.
- Chapter 17 on page 293 discusses environmental requirements that are not part of the application programming interface.
- Chapter 18 on page 319 describes the semantic mapping among STDL tasks, processing procedures, presentation procedures, transaction managers, resource managers and the RTI protocol.
- Appendix G on page 529 defines the encoding rules for STDL data types.
- Appendix H on page 533 defines STDL's use of the DCE RPC protocol.

Unless stated otherwise, all chapters and appendices are normative.

16.3 Syntax Notation

Refer to Section 2.3 on page 29.

16.3.1 Definitions

The following terms used in this document are defined in other specifications:

Standard COBOL or COBOL
Refers to ISO 1989: 1985 (including ISO 1989/Amendment 1).

Standard C or C
Refers to ISO C.

Standard programming languages
Refers to standard COBOL and standard C.

Standard SQL or SQL
Refers to the referenced X/Open SQL specification.

RTI
Refers to the Remote Task Invocation (RTI) protocol defined in the referenced X/Open TxRPC specification.

Standard ISAM
Refers to the referenced X/Open ISAM specification.

Standard XFTAM
Refers to the referenced X/Open XFTAM specification.

TxRPC Specification
Refers to the referenced X/Open TxRPC specification.

16.4 Conformance

Refer to Section 2.5 on page 32.

Chapter 17

Environmental Information

Environmental information is divided into the following categories:

- Translation Information

  Used during the processing of definitions and programming language sources into executable entities.

- Execution Information

  Information used to control and regulate the execution of TP system entities.

17.1 Translation Information

This section discusses the data required at translation time. Translation is an implementation-specific concept. In this document, *translation* and *translation time* refers to the processing required to generate an entity that can be executed by the TP system. A set of source files are translated to create an executable entity. Source files do not have to be translated at the same time. An implementation can translate source files at different times and store the information in an implementation-specific repository.

Environmental information that is not contained in another source file can also be required in order to produce an executable entity from source file S1. An implementation can specify this information during the translation of S1 or specify this information as part of the execution environment.

The following sections discuss the source code or definitions that are translated and the associated environmental information.

17.1.1 Translating Source Files

An implementation must be able to process source files as follows:

- A record source file is translated at least into:
  - — a COBOL copy file that defines record layouts as records
  - — a C include file that defines record layouts as structures.
- A task group source file is translated at least into:
  - — a client stub
  - — a COBOL copy file that defines record layouts as records
  - — a C include file that defines record layouts as structures and the tasks as C function prototypes.
- A task source file is translated at least into executable tasks.
- A processing group source file is translated at least into:
  - — a COBOL copy file that defines record layouts as records
  - — a C include file that defines record layouts as structures.

- A presentation group source file can be translated into:
  — a COBOL copy file that defines record layouts as records
  — a C include file that defines record layouts as structures.
- A message group source file is translated at least into:
  — a COBOL copy file that defines message names as records
  — a C include file that defines the message names as an initialized structure variable.

When translating an STDL source file, an implementation must able to accept preprocessor replacement macro definitions from outside the source file. These preprocessor replacement macro definitions must define <pre-identifier>s and the definition can be <nil> or a <pre-define-replacement-list> as defined in Section 4.14.1 on page 102.

17.1.2 Procedures

This section uses the generic term *procedure* to refer to a processing procedure. Section 3.2.3 on page 43. defines processing procedures.

Procedures typically perform database or file operations. File and database access information is used to logically connect the file or database to the procedure. The access information specifies how the file or database is accessed. STDL data type definitions also can be referenced in procedures. Figure 17-1 shows the association of data type definitions and access information for files and databases to the procedure.

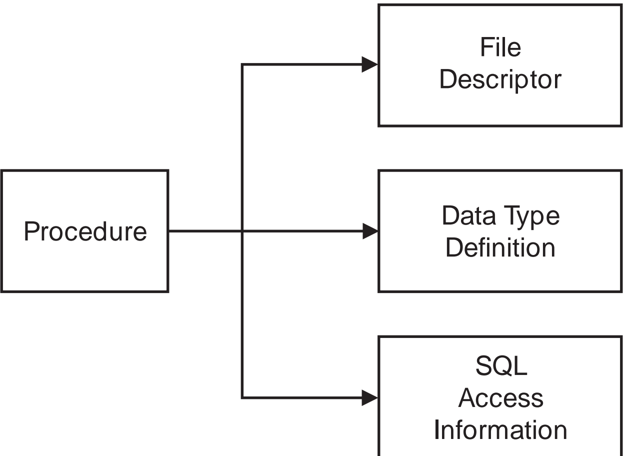


Figure 17-1 File and Database Descriptors for a Procedure

Processing procedures share the same file descriptor, data type definition and SQL access information.

The following sections describe information referenced by the procedure.

17.1.2.1 File Descriptor

A file descriptor provides a connection between the application program and a physical file. The file descriptor is used in customer-written COBOL and C procedures. A procedure can share file context with another procedure if both procedures specify the same file descriptor.

The following file access information is specified:

1. Name

   Identifies the file descriptor. The file descriptor name is specified in customer-written C and COBOL procedures that access files.

2. File Specification

   The implementation-specific name for the file. This name is used to locate the physical file within the TP system.

3. Open Mode

   Specifies how the file is to be opened. This depends on the language:

   a. For procedures written using COBOL:

      INPUT — Retrieves records from the file.

      EXTEND — Adds records to the end of an existing file. The extend open mode can be used only for files in the sequential access mode.

      I-O — Retrieves and updates records.

      OUTPUT — Adds records from the beginning of an existing file. The output open mode can be used only for files in the sequential access mode.

   b. For procedures written in C:

      Read — Access a file for read.

      Update — Access a file for update (read and write).

      Append — Access a file for update; begin writing at the end of the file.

4. Access Mode

   The access mode describes how the program selects records within the file.

   The following access modes are available for procedures written in COBOL:

   a. Sequential

      Specifies that the records in a file are accessed in a sequential order.

   b. Random

      Specifies that the records in a file are accessed in a random order.

   c. Dynamic

      Specifies that the records in a file can be accessed either sequentially or randomly.

   Access modes are defined in the COBOL standard. A file with a sequential organisation can be accessed using the sequential access mode only. Files with a relative or indexed organisation can be accessed with any access mode.

   Standard C provides support for byte stream and ISAM read and write operations to a file. Therefore, only stream file and ISAM organisation and access modes as specified in

standard C can be used from a procedure written using C.

5. Sharing

   Specifies the availability of transactions to access the file concurrently. The following levels of sharing are specified:

   a. EXCLUSIVE

      Specifies that only one transaction at a time can access the file. After a transaction is completed, the file is available to another transaction.

      A non-transactional file with any open mode other than INPUT must specify exclusive access.

   b. SHARED READ

      Allows other concurrently executing transactions read only access to the file.

      A non-transactional file with an open mode of INPUT can specify SHARED READ.

   c. SHARED

      Specifies that the file is available to multiple, concurrently executing transactions.

      A non-transactional file cannot specify SHARED access.

      — Isolation Level

        Level of isolation used to read data from the file. Used only for SHARED access.

   Implementor Note:

   If an implementation provides only one level of isolation, the isolation level attribute is not required.

   Procedures written using COBOL can use sharing modes listed in a., b. and c. Procedures written in C can use sharing modes listed in a. and b.

17.1.2.2 Data Type Definition

The data type definition specifies the format or layout of a record. STDL data type definitions that are referenced by a procedure must be made available when translating the procedure. An implementation-supplied mechanism converts data type definitions to COBOL COPY files and C include files. These files are included in the source procedure program by using either the COBOL COPY statement or the C #include directive.

17.1.2.3 SQL Access Information

The SQL access information is associated with the procedures in which the SQL tables are accessed. The SQL access information is implementation-specific. The following SQL access information can be associated with a procedure. A procedure can share database context with another procedure if they share the same implementation-specific SQL access information.

Implementor Notes:

1. An implementation is not required to share SQL context between procedures written using different programming languages.
2. An implementation can require all procedures called within the scope of a single processing procedure to use the same SQL access information.

The following SQL access information is specified:

1. Isolation Level

   Level of isolation to read data from SQL.

Implementor Notes:

1. Name, file specification and open mode are COBOL file concepts. The SQL standard does not explicitly contain equivalent concepts.
2. If an implementation provides only one level of isolation, the isolation level attribute is not required.

17.1.3 Transferring a Task Group Specification

If a client program or external client calls a task on another TP system, the task group specification and associated data type definitions for that task must be processed by the TP system or external environment from which the task call originates.

The source text of all data type definitions referenced in the task group specification are appended into a single file. The task group specification is appended to the data type definitions to form a single source file for copying. The single source file is transferred to any client TP system that can invoke the task and must be translated on the destination system.

17.1.4 Moving an Application

In order to move a TP application from one TP system to another TP system, all source files must be transferred and translated within the destination environment. Any naming conflicts must be resolved at the destination system.

17.2 Execution Information

The model used in this section defines the environmental information and functionality required, but does not imply any specific implementation.

This section describes the attributes that must be maintained within a TP system that supports STDL applications. An implementation is not required to implement the exact entities defined in this section, provided the the functionality specified by the attributes and entities is equivalently implemented.

17.2.1 TP Entity

A transaction processing entity (TP entity) is a named entity within the TP system that contains environmental information called attributes. The value of some attributes is maintained by the TP system; others are defined externally through interfaces provided by the TP system. A *system manager* uses these interfaces to define entities and their attributes.

The information in this chapter is used by the TP system in addition to that which is specified in STDL.

A *transaction processing entity* (*TP entity*) is a manageable component of the TP system. A TP entity is defined using:

1. A name that identifies the entity.
2. A set of attributes that describes the entity.
3. A set of abstract services that perform operations on the entity. The following types of entities are defined:
   a. Static

      A *static entity* is an entity that can only be defined before the TP system is started.

   b. Dynamic

      A *dynamic entity* is an entity that can be defined after the TP system has been started. However, there is no requirement to define the entity while the TP system is executing tasks.

   c. Volatile

      A *volatile entity* is an entity that can be defined while the TP system is executing tasks. No volatile entities are defined in this section.

The following three types of attributes are defined for a TP system entity:

1. Static

   A *static attribute* is an attribute that can only be defined before the TP system is started. There is no requirement to modify the attribute after the TP system has started.

2. Dynamic

   A *dynamic attribute* is an attribute that can be defined after the TP system has been started. However, there is no requirement to modify the attribute while the TP system is executing tasks.

3. Volatile

   A *volatile attribute* is an attribute that can be modified while the TP system is executing tasks.

Implementor Note:

Static, dynamic and volatile are terms used to describe the minimum requirements. An implementation can allow static entities to be defined as dynamic, and static attributes to be defined as dynamic or volatile. Similarly, dynamic attributes can be defined as volatile attributes.

This chapter uses the term *entity* to refer to a TP entity.

17.2.2 TP System Entity Containment

The *TP system* is a collection of entities that are managed as a unit. The entities can be considered as logically containing one another. For some types of containment, the containing entity must contain exactly 1 entity of a type. This is shown as 1, 1:1. For other types of containment, the containing entity can contain 0, 1, or more entities of a type. This is shown as 1, 0:n. For still other types of containment, the containing entity must contain 1 or more entities of a type. This is shown as 1, 1:n.

Whether or not different types of entities share the same name scope is undefined. The entities are not necessarily contained within the hierarchy. The hierarchy is logical and is not meant to imply that all the entities have to be contained within the TP system, only that the TP system has to be able to access the entities.

Figure 17-2 on page 300 shows the managed entities within an STDL system and the containment relationships among them.

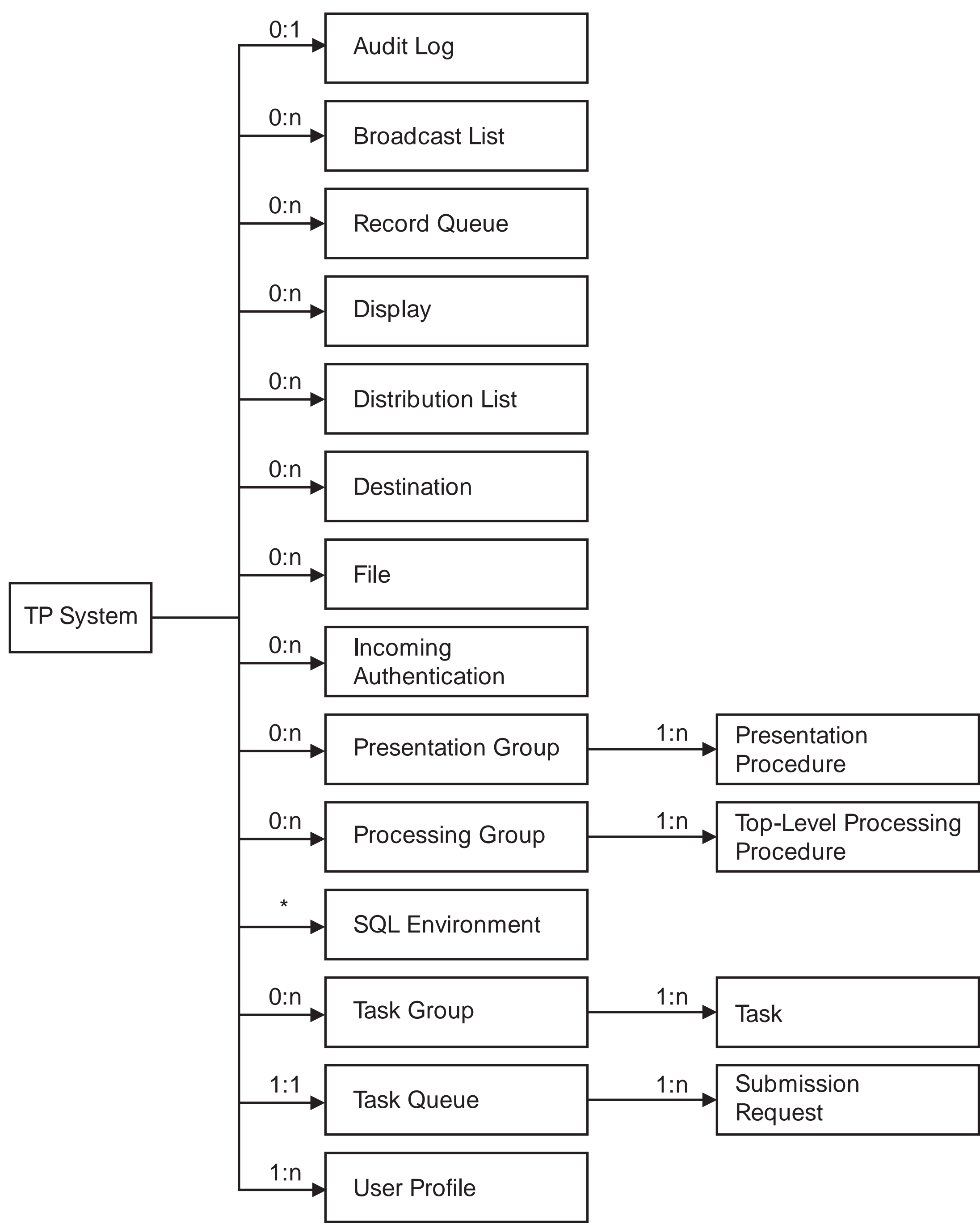


Figure 17-2 Entities Managed Within a TP System

The following sections discuss each of the entities that exist within an STDL TP system.[3]

17.2.3 TP System

The TP system entity is defined in Section 3.2 on page 40. All other entities are contained within, logically contained within, or accessible from a TP system.

The TP system entity:

1. Is defined by the system manager.
2. Is a static entity.
3. Has the following attributes:
   a. Name

      Identifies the TP system.

   b. Task Instance High Water Mark

      The high water mark is reached when the TP system has sufficiently depleted the resources needed to accept additional task calls. When the high water mark is reached, any additional task calls result in an exception of the ENV-INVOCATION-ERROR class. After the amount of resources used falls below the low water mark, task calls no longer result in an exception being generated.

      High water mark is a static attribute.

   c. Task Instance Low Water Mark

      After the high water mark has been reached, and the resource usage falls below the low water mark, task calls no longer result in an exception being generated.

      Low water mark is a static attribute.

      Implementor Note:

      An implementation can choose the level of granularity for high and low water marks. It must be supported at least at the TP system level, and can also be supported at the task group level

      Conformance Note:

      High and low water marks are not required for Support Levels 1, 2 and 3.

   d. Outgoing Authentication

      Information passed by the dialogue client TP system to the dialogue server TP system so that the dialogue server TP system can authenticate a principal on the dialogue client TP system, as defined in Section 3.4.3 on page 57.

      — Principal Name

---

3. Destination name can be implemented using a name resolution service that can span multiple TP systems. The only real requirement for containment is to ensure that remote task calls work properly.

Text name that identifies the principal.

— List of Authentication Information

One entry exists in this list for each authentication mechanism supported. The choice of which authentication data to use is determined by attributes of the destination entity.

— Authentication Mechanism Type

Default or customer-written.

— Authentication Data

Authentication data required for the authentication mechanism type.

Implementor Note:

The implementation chooses the levels of granularity for specifying the outgoing authentication. These levels of granularity can be at the TP system or implementation-defined entities within the TP system.

e. System Failure Information

A TP system failure results in the halt of the TP system execution, and is usually followed by a TP system restart. Any transaction currently executing at the time the TP system fails is rolled back.

The mechanism for handling TP system failure is implementation-specific. Therefore, the information required to handle the system failure is also implementation-specific.

System failure information is a static attribute.

f. Universal Time Offset

Specifies the time difference between the TP system's local time and universal time.

Universal time offset is a static attribute.

4. Supports the following abstract services:

a. Create

Defines a new TP system.

b. Delete

Deletes an existing TP system.

c. Set

Changes the attributes of the TP system.

17.2.4 Audit Log

The audit log entity is defined in Section 3.8.11 on page 72.

The audit log entity:

1. Is logically contained within the TP system entity.
2. Is a static entity.
3. Is defined before the TP system is in service.
4. Supports the following abstract services:
   a. Format

      Converts data from the implementation-specific audit log to a standard format.

      Appendix B on page 487 specifies the standard record format for audit log records.

17.2.5 Broadcast List

The broadcast list entity is defined in Section 3.2.5 on page 44.

The broadcast list entity:

1. Is logically contained within or accessible from the TP system entity.
2. Is a dynamic entity.
3. Can be defined by the system manager.
4. Can be defined while the TP system is in service.
5. Has the following attributes:
   a. Name

      OS name that identifies the broadcast list.

      Is specified in the <broadcast-send> within a <task-definition>.

   b. Display

      A list of one or more displays to which a SEND request is sent.

      Is a dynamic attribute. Items can be removed or added to the list.
6. Supports the following abstract services:
   a. Create

      Defines a new broadcast list.

   b. Delete

      Deletes an existing broadcast list.

   c. Set

      Changes the attributes of a broadcast list.

Conformance Note:

Broadcast lists are required only for Profile A. They are optional for Profile B.

17.2.6 Record Queue

The record queue entity is defined in Section 3.8.10 on page 72.

The record queue entity:

1. Is logically contained within the TP system entity.
2. Can be defined by the system manager.
3. Is a static entity.
4. Can be defined before the TP system is in service.
5. Has the following attributes:
   a. Name

      Identifies the record queue. The name is used on <enqueue-record>, <dequeue-record> and <read-queue-record> in the <task-definition>.

   b. Data Type Definition

      Identifies the associated <data-type-definition> for the record queue.

      Is a static attribute.

   c. Mapping

      Associates the record queue name with a physical queue.

      This is a static attribute.

   d. Key

      Identifies the <field-definition> within the <data-type-definition> to be used to retrieve data from the queue.

      A queue can have multiple keys. The maximum number of keys is implementation-specific and cannot be greater than the number of fields in the associated <data-type-definition>.

      Is a static attribute.

   e. Durability

      Specifies whether the record queue is a durable resource.

6. Supports the following abstract services:
   a. Create

      Defines a new record queue.

   b. Delete

      Deletes an existing record queue.

   c. Set

      Changes the attributes of a record queue.

17.2.7 Display

The display entity is defined in Section 3.2.5 on page 44.

The display entity:

1. Is logically contained within the TP system entity.
2. Can be defined by the system manager.
3. Is a static entity.
4. Has the following attributes:
   a. Name

      Identifies the display.

      The display name can be passed on the task invocation request, the <send-display>, or the <broadcast-send> within the <task-definition>.

   b. Mapping

      Associates the display with a physical device. The physical device can be capable of input, output or input-output.

      This is a static attribute.

   c. Alternate Mapping

      Associates a list of one or more physical devices to which to send output if the device specified in the mapping attribute is not available. It is implementation-specific which device in the list is used.

      This is a static attribute.
5. Supports the following abstract services:
   a. Create

      Defines a new display.

   b. Delete

      Deletes an existing display.

   c. Set

      Changes the attributes of a display.

Conformance Note:

Displays are required for Profile A. Alternate mapping is not required for Profile A and is an optional feature for Profile B.

17.2.8 Distribution List

The distribution list entity is defined in Section 3.3.2 on page 50.

The distribution list entity:

1. Is logically contained within or accessible from the TP system entity.
2. Can be defined by the system manager.
3. Is a dynamic entity.
4. Has the following attributes:
   a. Name

      Identifies the distribution list.

      The name is specified on the <submit-task>.
   b. Destination

      Lists one or more destinations at which to execute the task.

      This attribute can be modified while the TP system is in service. Items can be added and removed from the list.

      This is a dynamic attribute.
5. Supports the following abstract services:
   a. Create

      Defines a new distribution list.
   b. Delete

      Deletes an existing distribution list.
   c. Set

      Changes the attributes of a distribution list.

Conformance Note:

Multiple-entry distribution lists are not required for Support Levels 1, 2 and 3.

17.2.9 Destination

The destination entity is defined in Section 3.3.1 on page 47. An implementation can choose to have a default destination entity for a TP system.

The destination entity:

1. Is logically contained within or accessible from the TP system entity.

   The destination name is modeled as being within the TP system entity, but can be implemented using a name resolution service outside the TP system as long as the service is accessible from the TP system. A name resolution service is outside the scope of this specification.
2. Can be defined by the system manager.
3. Is a dynamic entity.

4. Has the following attributes:

   a. Name

      Identifies the destination.

      A <call-task> or <submit-task> either explicitly supplies a destination name, or the task group name is used as the destination name. The destination name is used by the client TP system or the submitter TP system to obtain the environmental information.

   b. Task Group List

      This list is indexed using the task group identification.

      Provides the following for each task group at the destination:

      — Task Group Identification

        An implementation can choose to use either the task group name or UUID.

      — Version Number

      — RTI-AP-Title

        See the referenced X/Open TxRPC specification.

      — Call Authentication Mechanism Type

        The authentication mechanism type to use for a remote task call to the server TP system for this task group at this destination.

      This is a dynamic attribute.

   c. Client

      Provides the RTI-AP-Title of the task enqueuer at the client TP system.

      This is a dynamic attribute.

   d. Transactional Forwarding

      Specifies whether the forwarding operation is done within the submitter TP system's transaction or if the forwarding operation is done outside of a transaction.

      This is a dynamic attribute.

   e. Forwarding Authentication Mechanism Type

      The authentication mechanism type to use for a remote task submission to the client TP system for this destination.

5. Supports the following abstract services:

   a. Create

      Defines a new destination.

   b. Delete

      Deletes an existing destination.

   c. Set

      Changes the attributes of a destination.

17.2.10 File

The file entity is defined in Section 3.8 on page 69.

A file entity:

1. Is logically contained within the TP system entity.
2. Can be defined by the system manager.
3. Is a static entity.
4. Has the following attributes:
   a. File Specification

      Identifies the file.

      The TP system uses the file specification to access the file. The file specification is not directly referenced in STDL.

   b. Organization

      Specifies how the records in the file are organised. The following file organisations are supported:

      — Sequential

      — Relative

      — Indexed

      — Stream

      The first, second and third organizations refer to files accessed from procedures written using COBOL. The third and fourth organizations refer to files accessed from procedures written using C.

      This is a static attribute.

   c. Data Type Definition

      Identifies the <data-type-definition> associated with the file.

      This is a static attribute.

   d. Transactional

      Specifies whether the file is a transactional resource. This attribute is applicable only to files of indexed, sequential and relative organisations.

      This is a static attribute.

   e. Key

      Identifies the <field-definition> within the <data-type-definition> that is to be used to organise and retrieve records. Only indexed files order records by key.

      This is a static attribute.

5. For C-ISAM, the attributes are:
   a. Locking Mode
   b. Variable or Fixed-length Record

   c. Access Mode.

6. Supports the following abstract services:

   a. Create

      Defines a new file.

   b. Delete

      Deletes an existing file.

   c. Set

      Changes the attributes of a file.

17.2.11 Incoming Authentication

Used by the server TP system to authenticate incoming task calls. The server TP system contains an incoming authentication for each principal that can invoke a task on the server TP system.

An incoming authentication entity:

1. Is logically contained within or accessible from the TP system entity.
2. Can be defined by the system manager.
3. Is a static entity.
4. Has the following attributes:

   a. Principal Name

      Text name that identifies the principal.

   b. List of Authentication Information

      One entry exists in this list for each authentication mechanism supported. The choice of which authentication data to use is determined by the dialogue client TP system.

      — Authentication Mechanism Type

        Default or customer-written.

      — Authentication Data

        Information required by the authentication mechanism to verify the principal's identity, using the authentication mechanism type.

17.2.12 Presentation Group

The presentation group entity is defined in Section 3.2.7 on page 45.

A presentation group entity:

1. Is logically contained within the TP system entity.
2. Can be defined by the TP system.
3. Is a static entity.
4. Has no required abstract services.

17.2.13 Presentation Procedure

The presentation procedure entity is defined in Section 3.2.6 on page 45.

A presentation procedure entity:

1. Is logically contained within the presentation group entity.
2. Can be defined by the TP system.
3. Is a static entity.
4. Has no required abstract services.

17.2.14 Processing Group

The processing group entity is defined in Section 3.2.4 on page 43.

A processing group entity:

1. Is logically contained within a TP system entity.
2. Can be defined by the TP system.
3. Is a static entity.
4. Has no required abstract services.

17.2.15 Top-level Processing Procedure

The top-level processing procedure entity is defined in Section 3.2.3 on page 43.

A top-level processing procedure entity:

1. Is logically contained within a processing group entity.
2. Can be defined by the system manager.
3. Is a static entity.
4. Has the following attributes:
   a. Maximum Fault Limit

      Specifies the maximum number of exceptions that can result from executions of this top-level processing procedure.

      Each execution of this top-level processing procedure that results in an exception contained within the exception list increments the fault count. If the fault count exceeds the maximum fault limit, the top-level processing procedure is disabled. The procedure can be enabled by the operator.

      The following exception classes are included in the exception list:

      — FATAL-TIMEOUT-FAULT

      — FATAL-EXECUTION-FAULT

      — AP-INVOCATION-FAULT

      — ENV-INVOCATION-FAULT

      — AP-EXECUTION-FAULT

      — ENV-EXECUTION-FAULT

— AP-RESPONSE-FAULT

— AP-PROCESSING-FAULT.

This is a dynamic attribute.

The default maximum fault limit is unlimited.

b. Fault Count

Maintains the number of exceptions resulting from executions.

The fault count attribute is incremented each time an execution of the top-level processing procedure results in an exception contained within the exception list. The fault count attribute is compared against the maximum fault limit attribute to determine whether the top-level processing procedure should be disabled. The fault count can be reset by the operator.

This is a volatile attribute.

The initial value of the fault count is zero.

Conformance Note:

Fault count is not required for Support Levels 1, 2 and 3.

c. Enable/Disable

Indicates whether or not the top-level processing procedure entity is accepting invocation requests. If a task attempts to invoke a procedure entity that has been disabled, an exception of the ENV-INVOCATION-ERROR class is returned.

The enable/disable attribute affects only invocation requests. Existing executions of this procedure are not affected.

This is a volatile attribute. This attribute can be modified by both the TP system and the system manager. If a procedure is disabled because of maximum fault limit being exceeded, the value of the enable/disable attribute is durably preserved.

The default value is to accept invocation requests (enable).

5. Supports the following abstract services:

   a. Set

   Changes the attributes of the top-level processing procedure.

17.2.16 SQL Environment

An SQL environment is an implementation-specific entity that contains one or more SQL databases. The number of databases within an SQL environment is implementation-specific.

The SQL database entity within an SQL environment is defined in Section 3.8 on page 69. All attributes and operations of the SQL database entity are implementation-specific or defined within the SQL standard.

17.2.17 Task Group

The task group entity is defined in Section 3.2.2 on page 42.

A task group entity:

1. Is logically contained within a TP system entity.
2. Can be defined by the TP system.
3. Is a dynamic entity.
4. Supports the following abstract services:

   a. Upgrade

      Modifies an existing task group.

      A task group can be upgraded after a TP system is in service. Existing task group context is not retained in the upgraded task group. Executions of tasks in the existing task group can run to completion while new invocation requests are received by the upgraded task group.

      Existing executions of tasks can be affected during upgrade.

17.2.17.1 Task

The task entity is defined in Section 3.2.1 on page 41.

A task entity:

1. Is logically contained within a task group entity.
2. Can be defined by the system manager.
3. Is a dynamic entity.
4. Has the following attributes:

   a. Access Control

      Restricts access to a task entity.

      The access control attribute can specify the access control for a task entity. The access control information is checked for all task invocation requests.

      The default access control is unspecified.

      This is a static attribute.

   b. CPU Time Limit

      Specifies the amount of execution cycles an execution instance of this task entity is allowed to consume. The level of granularity for the CPU time limit is implementation-specific. However, an implementation must provide some mechanism, either within an execution instance or outside of an execution instance, to detect the occurrence of excessive CPU resource consumption at least among the following:

      — the task definition

      — any top-level processing procedures called by the task and any other processing procedures invoked by them through calls to nested processing procedures.

If the time limit is exceeded, execution of the task is terminated and an exception of the FATAL-TIMEOUT-FAULT class is returned to the client program. The actual accounting of execution cycles is implementation-specific.

This is a dynamic attribute.

The default value for the CPU time limit is unlimited.

c. Transaction Time Limit

Specifies the amount of elapsed time a transaction is allowed to consume. The transaction time limit does not include the elapsed time of a transactional <call-presentation-procedure> statement with a <call-presentation-receive-list> or a <dequeue-record> or <read-queue-record> WITH WAIT.

The transaction time limit can be used to detect a deadlock. If the transaction time limit is exceeded, a TXN-TIMEOUT-ERROR class exception is generated.

If the first statement in the BLOCK WITH TRANSACTION is a transactional <call-presentation-procedure> statement with a <call-presentation-receive-list> or a <dequeue-record> or <read-queue-record> WITH WAIT, the timer is not started until after the first statement completes.

The level of granularity for which the transaction time limit is specified is implementation-specific, provided the level is equal to or less than a transaction.

An implementation can exclude the elapsed time of a <call-presentation-procedure> from the transaction time limit.

A transaction time limit associated with a composable task can generate an exception only if the time limit expires before the composable task returns to the client program.

This is a dynamic attribute.

The default value for transaction time limit is unlimited.

d. Presentation Procedure Time Limit

Specifies the amount of elapsed time a <call-presentation-procedure> statement is allowed to consume.

If the execution of a <call-presentation-procedure> statement exceeds the specified time, the <call-presentation-procedure> statement is terminated and an exception of the REQUEST-TIMEOUT-ERROR class is generated in the executing task.

This is a dynamic attribute.

The default value for presentation procedure time limit is unlimited.

e. Dequeue Record Time Limit

Specifies the amount of elapsed time a <dequeue-record> or <read-queue-record> is allowed to consume.

If the execution of the <dequeue-record> or <read-queue-record> exceeds the specified time, the dequeue or read operation is terminated and an exception of the REQUEST-TIMEOUT-ERROR class is generated in the executing task.

This is a dynamic attribute.

The default value for dequeue record time limit is unlimited.

f. Maximum Fault Limit

Specifies the maximum number of exceptions that can result from executions of this task entity.

Each execution of this task entity that results in an exception within the exception list increments the count. If the count exceeds the maximum fault count limit, the task is disabled. The task can be re-enabled by an operator.

The following exception classes are included in the exception list:

— FATAL-TIMEOUT-FAULT

— FATAL-EXECUTION-FAULT

— AP-INVOCATION-FAULT

— ENV-INVOCATION-FAULT

— AP-EXECUTION-FAULT

— ENV-EXECUTION-FAULT

— AP-RESPONSE-FAULT

— AP-PROCESSING-FAULT.

This is a dynamic attribute.

The default value for maximum fault limit is unlimited.

g. Fault Count

Maintains the number of exceptions resulting from executions.

The fault count attribute is incremented each time an execution of the task results in an exception contained within the exception list. The fault count attribute is compared against the maximum fault limit attribute to determine whether the task is to be disabled. The fault count can be reset by the operator.

This is a volatile attribute.

Initial value is zero.

h. Restart Count

Indicates the number of times the TP system has restarted the current transaction block within a non-composable task.

Only a non-composable task containing the restartable attribute can be restarted when a transient transaction exception occurs in a transaction block.

If the restart count exceeds the restart limit, a permanent transaction exception is generated. The class, code and source of the exception are not changed. The restart count is initialized to zero at the beginning of each transaction.

This is a volatile attribute.

i. Restart Limit

Specifies the number of times a transaction can be restarted within a non-composable task following a transient transaction exception.

Once the restart count exceeds the restart limit, a permanent transaction exception is generated. A restart limit of zero indicates that no restart is performed for the task.

This is a dynamic attribute.

The default value for the restart limit is zero.

j. Execution Priority

Schedules the use of critical resources between concurrently-executing tasks.

The definition of critical resources and scheduling mechanisms is implementation-specific.

This is a dynamic attribute.

The default value for the execution priority is unspecified.

Implementor Note:

An implementation can allow a composable task that is called by a task from within the same TP system to inherit the execution priority of the calling task.

k. Enable/Disable

Indicates whether or not this task is accepting invocation requests.

If a client attempts to call a task that has been disabled, an exception of the ENV-INVOCATION-ERROR class is returned to the client.

The enable/disable attribute affects only invocation requests. Existing executions of this task are not affected.

This is a volatile attribute. This attribute can be modified by both the TP system and the system manager. If a task is disabled because of maximum fault limit being exceeded, the value of the enable/disable attribute is durably preserved.

The default is enable.

5. Supports the following abstract services:

   a. Set

      Changes the attributes of the task.

17.2.18 Task Queue

The task queue entity is defined in Section 3.3.2 on page 50.

A task queue entity:

1. Is logically contained within the TP system entity.
2. Can be defined by the TP system.
3. Is a static entity.
4. Has the following attributes:

   a. Enable/Disable

      Specifies whether the task queue entity is accepting requests. An attempt to submit a request to a disabled queue generates an ENV-INVOCATION-ERROR class exception. This is a volatile attribute.

The default is to accept requests (enable).

b. Discard/Retain Requests

Specifies whether or not all of the requests on the task queue are discarded during a TP system restart.

This is a volatile attribute.

The default is to retain requests during a TP system restart.

c. Retry Interval

Information used by the TP system to determine the retry policy of the task queue. The retry interval specifies the amount of time the TP system waits before retrying the request. The required unit of granularity for the retry interval is 1 minute.

The default is not to wait before retrying the operation.

This is a static attribute.

d. Retry Limit

Specifies the number of times the TP system attempts to call a task or forward a request on the task queue. The default is to attempt the call once.

This is a static attribute.

e. Access Control

Restricts the ability another TP system to forward a task submission request to this TP system.

The access control attribute can specify the access control for a task queue from another TP system. The access control information is checked for each task forwarding request.

The default access control is unspecified.

This is a static attribute.

5. Supports the following abstract services:

a. Set

Changes the attributes of the task queue.

17.2.18.1 Submission Request

Each item in the task queue is a *submission request* entity.

A submission request entity:

1. Is logically contained within a task queue entity.
2. Is defined by the TP system as the result of the execution of a <submit-task>.
3. Is a volatile entity.
4. Has the following attributes:

a. Hold/Release

Allows the operator to release items that were previously put on operator hold.

This is a volatile attribute.

The default is release.

b. Retry Count

Maintains the number of retries on forwards or task calls.

The retry count attribute is incremented each time a forward or call of the submission request results in an exception listed in Section 3.3.2 on page 50. The retry count attribute is compared against the retry limit of the task queue to determine whether the forward or call is retried.

This is a volatile attribute.

The initial value of the retry count is zero.

5. Supports the following abstract services:

   a. Delete

      Deletes an existing submission request.

   b. Set

      Changes the attributes of the submission request.

   c. Stop Repeat

      Stops a repeating task.

17.2.19 User Profile

The *user profile* entity represents a user of the TP system. The TP system executes tasks on behalf of a user. Information from the user profile can be passed on a task call. The TP system associates a user profile with each task execution.

A user profile entity:

1. Is logically contained within or accessible from the TP system entity.
2. Can be defined by the system manager.
3. Is a static entity.
4. Has the following attributes:

   a. Name

      Identifies the user. The user profile name is not visible to STDL.

   b. User Authentication

      Information used by the local TP system or system on which the local TP system executes to verify the identity of the user. The format and use of this information is implementation-specific.

      This is a static attribute.

   c. Natural Language

      The natural language for the user. This information is used by menu client programs and customer-written client programs to set the language for tasks invoked by the menu client or customer-written client programs. This information can also be used to set the language for a task with <send-display> when the display is being used by

a user.

This is a static attribute.

5. Supports the following abstract services:

   a. Create

      Defines a new user profile.

   b. Delete

      Deletes an existing user profile.

   c. Set

      Changes the attributes of a user profile.

Chapter 18

Application Execution

18.1 Introduction

To support the execution of STDL tasks, a TP system performs three main types of execution:

- non-task client program execution
- task execution
- submitted task operations.

All processing within a TP system that is required by this document is in support of one of these three types of execution. Multiple executions can be present concurrently within a TP system.

This chapter describes the interactions between executions and:

- RTI service primitives defined in the referenced X/Open TxRPC specification and Appendix H on page 533
- processing and presentation procedures
- local resource managers.

This description is done by using a set of state machines with abstract service primitives. For each state machine, a set of service primitives, a list of other state machines used, variables, predicates, events, actions, states and a state table are defined.

18.2 Scope

This chapter describes a model of a conformant TP system and describes the mapping of that model to the service primitives and parameters described in the referenced X/Open TxRPC specification and Appendix H on page 533.

The semantics expressed in this chapter and the semantics expressed in the referenced X/Open TxRPC specification and Appendix H on page 533 other than the API are normative. A conformant implementation must provide these semantics.

The particular state machines and their abstract service primitives are not normative. They are specified as a way to express the required semantics. A conformant implementation does not have to implement the particular state machine and its abstract service primitives as described in this chapter.

All information from Part 1, STDL Specification and Chapter 17 on page 293 replicated in this chapter takes precedence over the information presented in this chapter.

The information not replicated from Part 1, STDL Specification and Chapter 17 on page 293 includes:

- Mapping of a <task-group-specification> onto an RPC interface (see Section 18.9.1.1 on page 424).
- RPC interface for the Task Enqueuer (see Section 18.9.1.2 on page 426).
- Actions that occur when a task is cancelled due to a:
  - cancel request from a client (such as a menu client)
  - cancel from the RTI protocol machine
  - unsolicited rollback from the RTI protocol machine

    This may cancel only part of a task.
  - transaction timeouts

    This may cancel only part of a task.
  - CPU timeouts.
- Mapping of RTI-CALL-FAILURE reason codes onto STDL exception classes (see Section 18.9.2 on page 433).
- Additional information about mapping exceptions to events.
- Information about marshalling and unmarshalling task arguments (see Section 18.7.3 on page 369, Section 18.8.1 on page 407, Section 18.9.1 on page 423 and Section 18.9.2 on page 433).
- Description of the task call data type definition (see Section 18.9.1.1 on page 424).
- Description of the task forwarding information data type definition (see Section 18.9.1.2 on page 426).
- Description of the exception information data type definition (see Section 18.9.1.3 on page 430).
- Specification of the parameters for RTI services (see Section 18.9.2 on page 433 and Section 18.9.3 on page 447).

18.3 Modelling Method

This section presents the semantics listed in Section 18.2 on page 320 by defining a model of STDL task execution including external clients and queued task forwarding. This is done by defining a set of state machines and using the protocol machine defined in the referenced X/Open TxRPC specification. These machines are abstract and are presented only to define the semantics listed in Section 18.2. Conforming implementations of STDL do not need to structure their implementations to reflect in any way the structure of the state machines presented in this chapter.

18.3.1 State Machines and State Machine Instances

A *state machine* is a set of rules and context definition. A state machine instance is defined as an execution of a state machine within the rules for that context.

The rules for a state machine are defined in terms of:

- A set of events that can occur.

  These events can be one of the following:

  — an event sent from another state machine

  — an internally generated event.

- Optionally a set of variables.

  Each state machine instance has its own copy of the variables.

- Optionally, a set of predicates which are applied to events.

  These predicates can depend on one of the following:

  — a parameter of an event

  — internally stored variable.

- A set of actions that state machine instances can execute.

  These actions can include:

  — sending events to other state machine instances

  — saving state and data in internal variables

  — generating internal events.

- A set of states that a state machine instance can be in.

  This set of states always includes a null state.

- A mapping of events, states and actions called a state table.

  For each state, the state table defines which events can occur and what happens when the event (possibly modified by predicates) occurs. When an event occurs in a state, the state table defines which state a state machine instance has after the event and possibly one or more actions to take when transitioning to that state.

When an event is sent to a state machine that has no applicable instance, that event must be legal for the null state. If that event causes a transition to another state, a state machine instance is created.

18.3.2 Requests and Indications

When the rules of two state machines define that events are sent between instances of the state machines, one state machine is said to be the service user, and the other state machine is said to be the service provider. The service provider offers a set of service primitives that describe the events that can be sent between the service provider and the service user. These service primitives are:

Requests — Issued by a service user and received by the service provider.

Indications — Issued by a service provider and received by the service user.

A service provider can in turn be a service user of another state machine or of itself.

The diagrams in the next section show the relationships among state machines. Arrows are used to show which state machine is the service user and which state machine is the service provider. The arrow points from the service user to the service provider. Requests flow in the direction of the arrow. Indications flow in the other direction.

Figure 18-1 shows this convention.

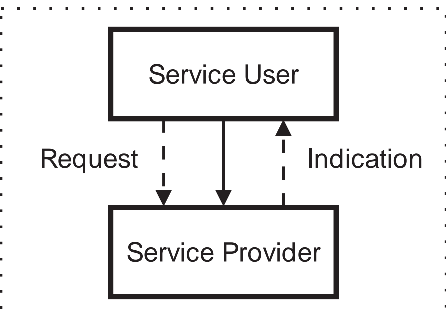


Figure 18-1 Requests and Indications

Normally a service user issues a request to a service provider before that service provider issues any indications to the service user. However, there are two cases where the service provider issues an indication to a service user before the service user issues any requests to that service provider. These two cases are:

- The RTI protocol machine issues an RTI-CALL-TASK indication to the Call state machine which creates a new Call state machine instance.
- The Concurrent Block state machine issues a CONC-THREAD-EXECUTE indication to the Concurrent Thread state machine which creates a new Concurrent Thread state machine instance.

18.3.3 State Machine Instance Context

A state machine instance context consists of:

- the current state
- the contents of the variables for the state machine, if any
- the set of state machine instances for which the state machine is the service user, if any
- the set of state machine instances that are the service user for this state machine, if any.

18.3.4 Thread Contexts

State machine instances are created only by an indication or a request. In this model, a state machine instance created by an indication executes asynchronously from the state machine instance that issued the indication. The creation of these state machines instances create thread contexts. A *thread context* is a set of state machine instances that execute synchronously. A thread context models a thread of execution.

In this model, a state machine instance created by a request executes synchronously with the state machine instance that issued the request. State machine instances created by requests execute within the thread context of the state machine instance that issued the request.

When a request or indication outside of a thread context arrives at a thread context, only one of the state machines within the thread context handles that request or indication. The state transitions caused by the request or indication complete before the next external request or indication is processed by any state machine within the thread context. This convention is important in understanding how the semantics of asynchronous events are modeled in this chapter.

As noted above, the state machine instances whose creation creates new thread contexts are:

- call state machine (when created by an indication from the RTI protocol machine)
- concurrent thread state machine.

In addition, there are some state machine instances created on requests from state machines that are outside of the scope of this chapter. For the purposes of this chapter, the creation of these state machine instances also create thread contexts. These state machines are:

- Client Program Call
- Task Dequeuer
- Task Forwarder.

18.3.5 Execution Contexts

To define which state machine instances are available to other state machine instances, the execution context defines a scope of access to state machine instances. An *execution context* is a set of state machine instances that share access to common state machine instances. An execution context can contain multiple thread contexts and a thread context can span execution contexts.

The description of the state machines includes information about whether or not multiple instances of the same state machine exist within an execution context and the criteria by which requests are directed to a particular state machine instance.

A separate execution context is created for each:

- Task state machine instance modelling a non-composable task execution.

  In this case, the execution context models the task group context.

- Task state machine instance modelling a task invoked by a client in a different task group or on a remote system.

  In this case, the execution context models the task group context.

- Task Forwarder state machine instance.

  In this case, the execution context models the context for the Task Forwarder.

- Task Enqueuer state machine instance.

  In this case, the execution context models the context for the Task Enqueuer.

- Task Dequeuer state machine instance.

  In this case, the execution context models the context for the Task Dequeuer. If the task dequeuer calls composable tasks on the same TP system, then this execution context also models the task group context for the server task.

- Client Program Call state machine instance.

  In this case, the execution context models the context for the client program (customer-written, external client, or menu client).

All other state machine instances execute in the same execution context as the state machine instance whose request or indication created the state machine instance.

18.4 Conventions

18.4.1 Mapping of Terms and Concepts

Many of the terms used in this chapter are unique to this chapter. This is due to the nature of the model presented in this chapter. The following list contains some mapping of terms and concepts from the previous chapters of this specification to the terms and concepts in this chapter:

- Task

  The execution of a task is modeled by the Task state machine and the state machine used by the Task state machine.

- Task Group

  The runtime interface provided by a task group is mapped to an RPC interface.

- Task Group Context

  A task group context is modeled by an execution context when that execution context contains a Task state machine instance.

  The parts of a task group context are modeled as follows:

  — Transactional Context

    Modelled by a Transaction state machine, optionally a Transactional Receive state machine, and by zero or more Resource Manager state machines (record queue context and processing group context) and Call state machines (transactional dialogue context). Note that shared workspace context is not modeled.

  — Non-transactional Context

    Modelled by a Presentation Procedure (presentation group context) state machine instance.

- Actions on a Submitter TP System when Forwarding a Task

  Modelled by a Task Forwarder state machine.

- Actions on a Client TP System when Accepting a Forwarded Task

  Modelled by a Task Enqueuer state machine.

- Actions on a client TP system when calling a task using the information in a task queue

  Modelled by a Task Dequeuer state machine.

18.4.2 State Machine Description Conventions

This section describes the conventions used for the state machines that describe the runtime semantics of a TP system that supports STDL applications.

Each state machine is described in a separate section which contains the following sections:

- Service Primitives Provided

  This section defines the requests and indications that the state machine provides to service users. The service primitive names are unique within this document and the referenced X/Open TxRPC specification. For each service primitive, either a list of parameters is defined or there is a statement that the service primitives have no parameters.

- Service User

  This section defines how the service user is determined for this state machine.

- State Machines Used

  This section defines the state machines that are used by this state machine. This state machine is a service user to those other state machines. The use of these state machines is defined in the Actions section.

- Variables

  This section defines a set of named variables. Each variable name begins with the letter V. The scope of the variable names is the state machine being described. These variables are used either in predicates, as conditionals in actions, or to store and retrieve values in actions.

  Each variable definition specifies whether the variable is a Boolean or stores data. All variables that are Booleans are set to false when the state machine instance transitions from the null state.

- Predicates

  This section defines a set of named predicates (condition tests). Each predicate name begins with the letter P. The scope of the predicate names is the state machine being described. For each predicate there is a definition of the predicate. The definition can involve testing parameters of the event with which the predicate is associated or testing a variable.

- Events

  This section defines a set of named events. Each event name begins with the letter E. The scope of the event names is the state machine being described. For each event there is a definition of the event.

  Events definitions can reference one or more:

  — requests from service users of this state machine

  — indications from service providers to this state machine

  — internal events.

- Actions

  This section defines a set of named actions. Each action name begins with the letter A. The scope of action names is the state machine being described. For each action there is a definition of the action.

  The action definition can perform one or more of the following:

  — issue a request

  — issue an indication

  — store data in a context variable

  — perform internal processing.

- States

  This section defines a set of named states. Each state name begins with the letter S. The scope of state names is the state machine being described. For each state there is a brief description of the state.

- State Transition Table

  This section defines the possible transitions between the states of the state machine. The states of the state machine are represented by the columns in the table. The header of the column contains the state name. Condition expressions are listed as row headers (down the left hand side). If the condition causes a state transition for a state, the box in the column for the state and the row for the condition will have an entry. The first line of the entry consists of the next state name. The following lines consist of 0, 1 or more actions that are to be taken in the order specified when the state transition occurs. The state transition table is represented using one or more figures. In some complex situations, a number in parenthesis is used to refer to an item in a notes list after the figures.

  The condition expression follows these rules:

  — The condition expression contains one or more events and zero or more predicates.

  — An event is true if the event occurs — only one event occurs at a time, so the condition expression must be constructed such that only one event must occur for the condition to be true.

  — Parentheses are used to create subexpressions. The subexpressions are evaluated first.

  — The + operator is used as an AND operator to join events, predicates, and subexpressions. For example, for E-a+P-b to be true, event E-a must occur and the predicate P-b must be true.

  — The or operator is used as an OR operator to join events and subexpressions. For example, for E-a + (P-b or P-c) to be true, event E-a must occur and either the predicate P-b or the predicate P-c must be true.

  — The ¬ operator is used as a NOT operator to negate a predicate or subexpression. For example, for ¬P-b to be true, the predicate P-b must be false.

18.5 Model

18.5.1 State Machine Categories

A component of the model is a state machine with its associated service primitives, if any. These state machines are divided into the following categories:

- Client Program Call

  This state machine models the work done when a customer-written client program, a menu client program or an external client calls a task.

- Task Execution

  These state machines model the execution of a task in enough detail to relate the execution of a task to the other state machines.

- Submitted Task Operations

  These state machines model the operations involved in forwarding submitted tasks and invoking submitted tasks in enough detail to relate these operations to the other state machines.

- Communication

  These state machines model the communications between local and remote executions

- Resource Manager

  These state machines model the interactions between resource managers on the local TP system and task executions and submitted task operations.

- Procedure Execution

  These state machines model the execution of presentation and processing procedures for task execution and the context kept for these procedures.

- RTI

  The protocol machine defined in the referenced X/Open TxRPC specification and Appendix H on page 533.

Figure 18-2 on page 329 shows the overall relationships among these categories of state machines.

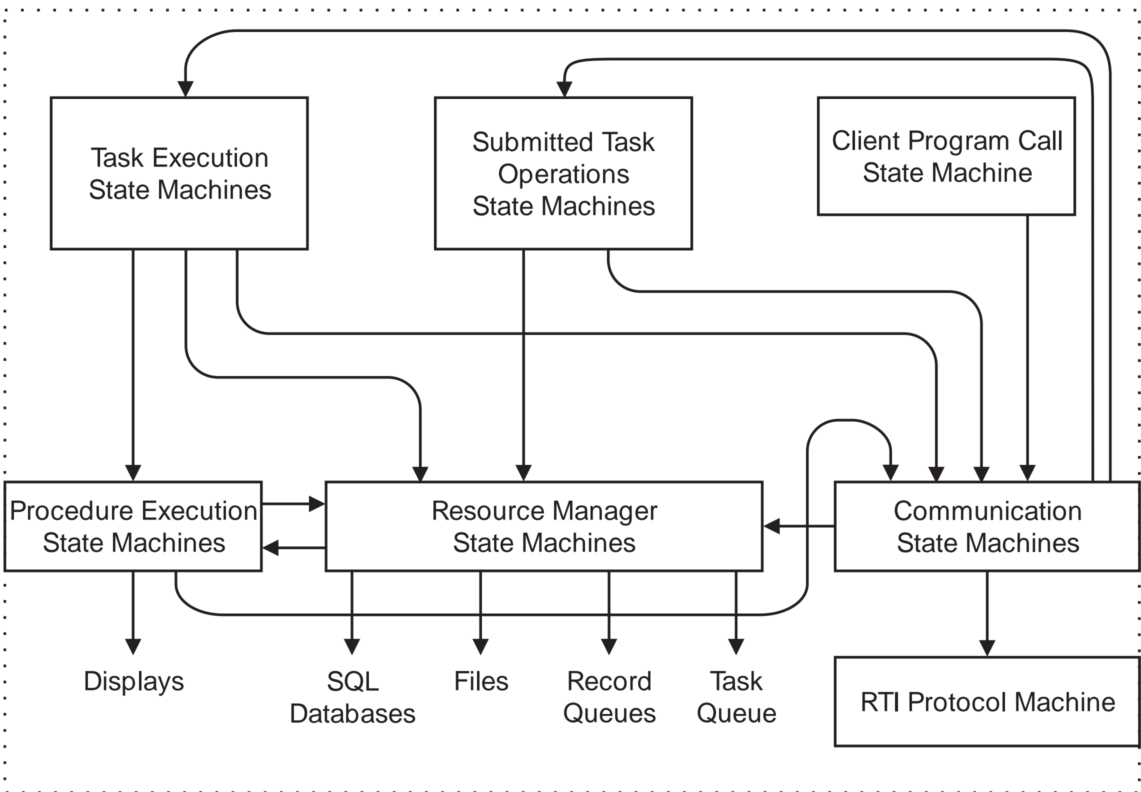


Figure 18-2 Overall Relationships Between State Machines

18.5.1.1 *Client Program Call*

There is one Client Program Call state machine:

- Client Program Call

  The Client Program Call state machine models the actions by customer-written client programs, menu client programs, and external client programs when they call tasks. It uses the Call state machine.

18.5.1.2 *Task Execution*

There are five Task Execution state machines:

- Task

  The Task state machine models the overall execution of a task. It uses the Non-transactional Statement List, Transactional Statement List, Transaction, and Presentation Procedure state machines as needed to execute the task.

- Non-transactional Statement List

  The Non-transactional Statement List state machine models the execution of a statement list that is outside of a <transaction-block>. It uses the Transaction, Transactional Statement List, Transactional Receive, and Non-transactional Statement List state machines as needed to execute the statement list.

- Transactional Statement List

  The Transactional Statement List state machine models the execution of a statement list inside of a transaction. It uses the Concurrent Block, Call, Processing Procedure, Presentation Procedure, Transactional Receive, Resource Manager, and Transactional Statement List state machines as needed to execute the statement list.

- Concurrent Block

  The Concurrent Block state machine models the execution of a <concurrent-block>. It issues indications to the Concurrent Thread state machine to execute the threads within the concurrent block.

- Concurrent Thread

  The Concurrent Thread state machine models the execution of a concurrent statement thread in a concurrent block. It uses the Transactional Statement List state machine to execute the concurrent statement.

18.5.1.3 Submitted Task Operations

There are three Submitted Task Operation state machines:

- Task Dequeuer

  The Task Dequeuer state machine models the calling of tasks based on the contents of the task queue. It uses the Transaction, Resource Manager, and Call state machines to dequeue tasks from the task queue and call them.

- Task Forwarder

  The Task Forwarder state machine models the forwarding of submitted tasks from a submitter TP system. It uses the Transaction, Resource Manager, and Call state machines to dequeue tasks from the task queue and forward them.

- Task Enqueuer

  The Task Enqueuer state machine models the acceptance of submitted tasks at a client TP system. It uses the Transaction and Resource Manager state machines to accept submitted tasks and queue them to the task queue.

18.5.1.4 Communication

There are two Communication state machines:

- Call

  The Call state machine models the work done when:

  — One execution calls another execution locally (client and server are local to each other).

  — One execution calls another execution remotely (client to remote server).

  — A remote call arrives for a local execution (server from remote client).

  The Call state machine uses the Transaction state machine, Task state machine, Task Enqueuer state machine, and the RTI Protocol Machine.

- Transaction

  The Transaction state machine models the work done to do transaction management within a TP system. It uses the RTI Protocol Machine, the Resource Manager state machine, and the Call state machine.

18.5.1.5 Resource Manager

There are two Resource Manager state machines:

- Transactional Receive

  The Transactional Receive state machine models the processing needed by transactional receives. It uses the Presentation Procedure state machine.

- Resource Manager

  The Resource Manager state machine models local resource managers which handle transactional sends, files, SQL databases, record queues, transactional workspaces, and the task queue. It uses the Presentation Procedure state machine when handling transactional sends. Note that transactional workspaces are not explicitly modeled in this chapter.

18.5.1.6 Procedure Execution

There are two Procedure Execution state machines:

- Processing Procedure

  The Processing Procedure state machine models the work done when a processing procedure is called. It uses the Resource Manager and Call state machines.

- Presentation Procedure

  The Presentation Procedure state machine models the work done when a presentation procedure is called (including initialization and termination procedures). The Presentation Procedure state machine models the interaction of customer-written presentation procedures and vendor-supplied forms systems and a display. It is self-contained.

18.5.2 Relationships Among State Machines

Figure 18-3 on page 332 shows the relationships among all of the state machines described in this chapter. This is a detailed picture of Figure 18-2 on page 329.

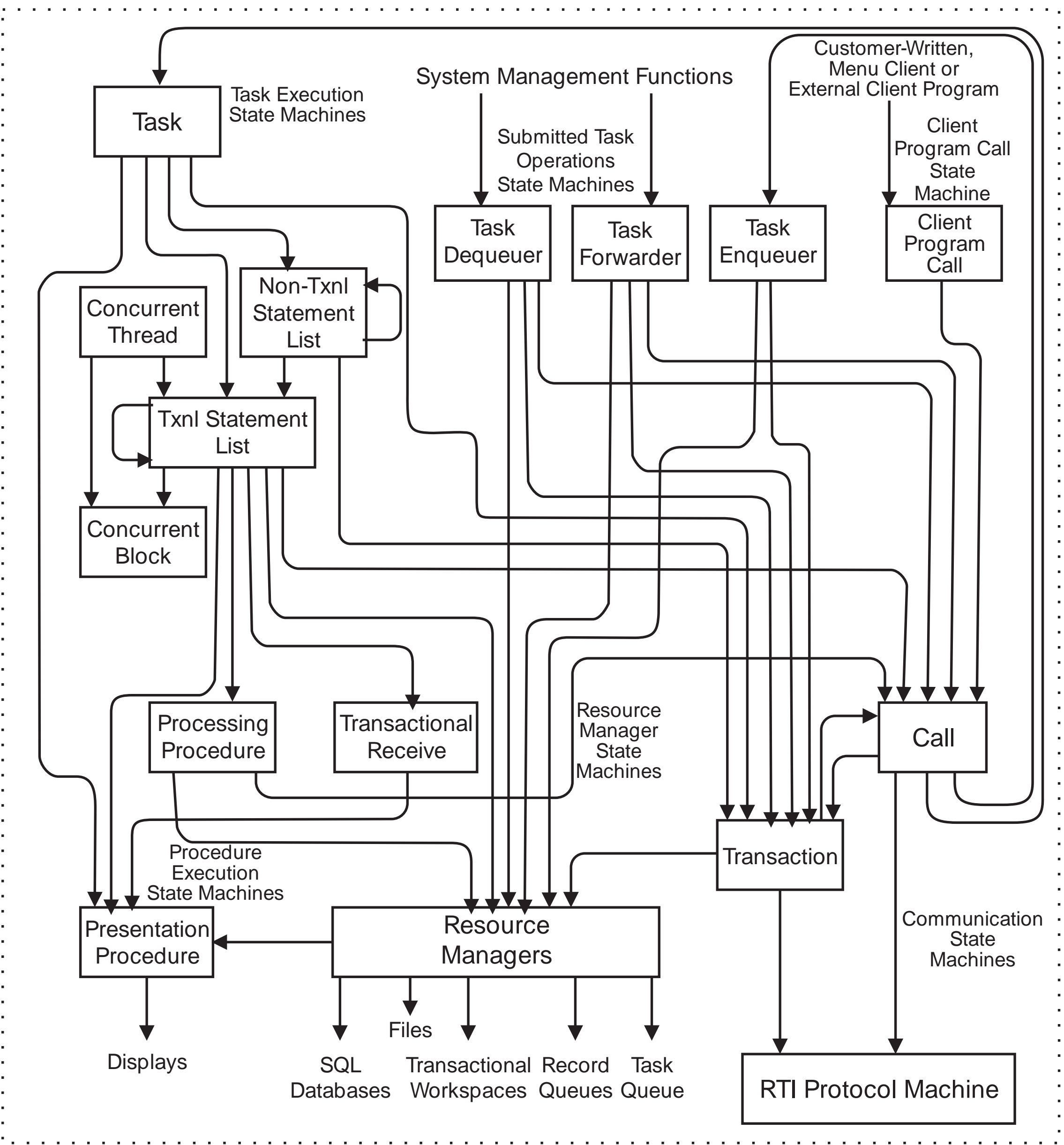


Figure 18-3 Relationships Among All State Machines

18.6 Client Program Call State Machine

The Client Program Call state machine models the actions of a customer-written client program, a menu client program, or an external client program when calling a task. One instance of the Client Program Call state machine exists for each of these client programs that is calling a task. Each Client Program Call state machine exists in a separate execution context. Note that customer-written client programs and menu client programs only call local tasks.

Service Primitives Provided

The Client Program Call state machine provides the following service primitives:

1. CP-CALL-TASK — request

   Calls a task for a client program.

   This request has the following parameters:

   a. Task-Group-UUID = from the <task-group-specification> for the server task

   b. Task-Group-Version-Major = the major version number of the server task group

   c. Task-Group-Version-Minor = the minor version number of the server task group

   d. Task-Operation-Value = the position of the <task-interface> within the <task-group-specification> as discussed in Section 18.9.1.1 on page 424.

   e. Arguments = the INPUT and INOUT arguments as defined in the <task-interface> of the <task-group-specification>

2. CP-TASK-DONE — indication

   Reports that the called task is finished.

   This indication has one parameter:

   a. Arguments = the OUTPUT and INOUT arguments as defined in the <task-interface> of the <task-group-specification>

3. CP-TASK-EXCEPTION — indication

   Reports that the called task resulted in an exception. The information in this indication is mapped into the external variable defined for customer-written client programs and external client programs to report exceptions.

   This indication has the following parameters:

   a. Class = one of the classes defined in Appendix C on page 489.

   b. Code = the exception code as defined in Section 3.7.2 on page 62.

   c. Exception-Code-Group = the exception code group as defined in Section 3.7.2 on page 62.

   d. Exception-Source = system or application

   e. Exception-Procedure = the procedure in which the exception occurred as defined in Section 3.7.2 on page 62.

   f. Exception-Procedure-Group = the procedure group in which the exception occurred as defined in Section 3.7.2 on page 62.

   Unlike other exception indications in this chapter, this indication does not include Type and Level as parameters. This information is not required by a customer-written client

program, external client program, or a menu client program. The type of exception reported by this indication is always non-transaction, because this state machine calls only noncomposable tasks.

4. CP-CANCEL-TASK — request

   Cancels the current task.

   This request has no parameters.

5. CP-EXIT — request

   Tells the Client Program Call state machine that the client program is exiting.

Service User

The service user of the Client Program Call state machine is a customer-written client program, external client program or menu client program as described in Chapter 13 on page 263. The modelling of these client programs is outside the scope of this chapter.

State Machines Used

1. Call (CALL-)

Variables

None.

Predicates

None.

Events

1. E-task-call

   A CP-CALL-TASK request received.

2. E-cancel

   A CP-CANCEL-TASK request received.

3. E-canceled

   A CALL-CANCELED indication received.

4. E-done

   A CALL-DONE indication received.

5. E-exception

   A CALL-EXCEPTION indication received.

6. E-exit

   A CP-EXIT request received.

Actions

1. A-call

   Issue a CALL-TASK request with the following parameters:

   a. Local = one of the following:

      — True if the client program is a customer-written client or a menu client

      — False if the client program is an external client

   b. RTI-API-Title = one of the following:

      — Omitted if Local is True

      — The RTI-AP-Title from a task group entry in a destination entry if Local is False. The implementation can choose to either determine the destination entry from the server task group name or use a default destination entry for the TP system. The submitter TP system chooses the task group entry within the destination by matching the server task group identification and determining whether or not the version number is compatible, as described in Section 3.3.1 on page 47 and Section 17.2.9 on page 306. An implementation can choose to use either the task group name or UUID as the task group identification.

   c. Task-Group-UUID = from the Task-Group-UUID parameter of the CP-CALL-TASK request

   d. Task-Group-Version-Major = from the Task-Group-Version-Major parameter of the CP-CALL-TASK request

   e. Task-Group-Version-Minor = from the Task-Group-Version-Minor parameter of the CP-CALL-TASK request

   f. Task-Operation-Value = from the Task-Operation-Value parameter of the CP-CALL-TASK request

   g. Transactional = false

   h. Task-Call-Info = a TASK-CALL-INFORMATION record as described in Section 18.9.1.1 on page 424 with the following fields set:

      — DISPLAY-NAME = set to null for external client programs; set to the display name for customer-written client programs and the menu client.

      — DISPLAY-TP-SYSTEM = set to null for external client programs; set to the display TP system name for customer-written client programs and the menu client.

      — LANGUAGE = set using environmental information

   i. Arguments = from the Arguments parameter of the CP-CALL-TASK request

2. A-cancel

   Issue a CALL-CANCEL request.

3. A-canceled

   Issue a CP-TASK-EXCEPTION indication with the following parameters:

   a. Class = ENV-EXECUTION-ERROR

   b. Code = implementation specific

c. Exception-Code-Group = implementation-specific
d. Exception-Source = system
e. Exception-Procedure = called task name
f. Exception-Procedure-Group = called task group name

4. A-cleanup

   Issue a CALL-RELEASE-CONTEXT request to each Call state machine instance in the execution context.

5. A-exception

   Issue a CP-TASK-EXCEPTION indication with the following parameters:

   a. Class = from the Class parameter of the CALL-EXCEPTION indication
   b. Code = from the Code parameter of the CALL-EXCEPTION indication
   c. Exception-Code-Group = from the Exception-Code-Group parameter of the CALL-EXCEPTION indication
   d. Exception-Source = from the Source parameter of the CALL-EXCEPTION indication
   e. Exception-Procedure = from the Exception-Procedure parameter of the CALL-EXCEPTION indication
   f. Exception-Procedure-Group = from the Exception-Procedure-Group parameter of the CALL-EXCEPTION indication

6. A-results

   Issue a CP-TASK-DONE indication with a single parameter:

   a. Arguments = from the Arguments parameter of the CALL-DONE indication

States

1. S-null

   The initial state.

2. S-task

   Waiting for the called task to complete.

3. S-canceled

   Waiting for the task to terminate after a cancel.

4. S-idle

   Waiting for another call task or the client program to exit.

State Transition Table

	S-null	**S-task**	**S-canceled**	**S-idle**
E-task-call	S-task A-call			S-task A-call
E-done		S-idle A-results		
E-exception		S-idle A-exception		
E-cancel		S-canceled A-cancel	S-canceled	
E-canceled			S-idle A-canceled	
E-exit				S-null A-cleanup

Table 18-1 Client Program

18.7 Task Execution State Machines

The following sections describe the state machines used in the execution of tasks.

18.7.1 Task State Machine

The Task state machine models the top-level execution of both composable and noncomposable tasks.

Multiple Task state machine instances can exist within an execution context:

1. A Task state machine instance for a noncomposable task must execute within a new execution context.
2. A Task state machine instance for a composable task called by a remote client on a new dialogue must execute within a new execution context.
3. A Task state machine instance for a composable task called by a local client task that did not specify a task group or destination on the task call must execute within the execution context of the client task.
4. A Task state machine instance for a composable task called by any other local client executes within an implementation-specific execution context, either within the client's execution context or within a new execution context.

Service Primitives Provided

1. TASK-EXECUTE — request

   Executes a task.

   This request has the following parameters:

   a. Local = a Boolean that is true if this is a local task call
   b. Transactional = a Boolean that is true if the task to be executed is a composable task.
   c. Task-Operation-Value = the operation value for the task to be executed
   d. Task-Call-Info = the TASK-CALL-INFORMATION record for this task call.
   e. Arguments = the INPUT and INOUT arguments as defined for this task in the task group specification.

2. TASK-CANCEL — request

   Cancel the current task execution.

   This request has no parameters.

3. TASK-CANCELED — indication

   Reports that the task execution has been cancelled. An implementation can use the semantics of TASK-DONE and/or TASK-CANCELED depending on implementation design and timing.

   This indication has no parameters.

4. TASK-DONE — indication

   Reports that the task execution has successfully completed and returns the output arguments.

This indication has one parameter:

a. Arguments = the OUTPUT and INOUT arguments as defined for the task to be executed in the task group specification for the task.

5. TASK-EXCEPTION — indication

   Reports that the task execution has completed with an exception.

   This indication has the following parameters:

   a. Type = Non-transaction, Transient Transaction or Permanent Transaction

   b. Class = one of the classes defined in Appendix C on page 489.

   c. Code = the exception code as defined in Section 3.7.2 on page 62.

   d. Exception-Code-Group = the exception code group as defined in Section 3.7.2 on page 62.

   e. Source = System or Application

   f. Level = Current or Propagated

   g. Exception-Procedure = the procedure in which the exception occurred as defined in Section 3.7.2 on page 62.

   h. Exception-Procedure-Group = the procedure group in which the exception occurred as defined in Section 3.7.2 on page 62.

6. TASK-RELEASE-CONTEXT — indication

   Reports that the call context is no longer needed. This is used for composable tasks with remote clients.

   This indication has no parameters.

Service User

The service user of the Task state machine is the state machine instance that issued the TASK-EXECUTE request. This service user receives all indications generated by a Task state machine instance and the TASK-CANCEL request from that service user is directed to the Task state machine instance that was created for that service user.

State Machines Used

1. Presentation Procedure (PRES-)
2. Non-transactional Statement List (NONTX-STATEMENT-)
3. Transactional Statement List (TX-STATEMENT-)
4. Transaction (TRANS-)

Variables

1. V-current-exception
   a. Type
   b. Class
   c. Code
   d. Exception-Code-Group
   e. Source
   f. Level
   g. Exception-Procedure
   h. Exception-Procedure-Group

   This variable stores data.

Predicates

1. P-f-exc

   Is true if the exception is a fatal transaction exception.

2. P-n-exc

   Is true if the exception is a non-transaction exception.

3. P-p-exc

   Is true if the exception is a permanent transaction exception.

4. P-remote

   Is true if the Local parameter of the TASK-EXECUTE request is false.

5. P-t-exc

   Is true if the exception is a transient transaction exception.

6. P-trans

   Is true if the Transactional parameter of the TASK-EXECUTE request is true.

7. P-valid

   Is true if the parameters for TASK-EXECUTE request are valid. These include:

   a. Check Transactional parameter of the EXECUTE-TASK request to verify that the called task is composable:

      — If the Transactional parameter is true, the called task must be composable

      — If the Transactional parameter is false, the called task must be noncomposable

   b. The arguments passed are valid.
   c. The authorization is valid and any access control checks pass.
   d. The task is not disabled.
   e. The high water mark has not been reached.

Events

1. E-cancel

   A TASK-CANCEL request received.

2. E-commit

   A TRANS-COMMIT-DONE indication received.

3. E-cpu-timeout

   A CPU timeout has expired.

4. E-execute

   A TASK-EXECUTE request received.

5. E-n-canceled

   A NONTX-STATEMENT-CANCELED indication received.

6. E-n-done

   A NONTX-STATEMENT-DONE indication received.

7. E-n-exception

   A NONTX-STATEMENT-EXCEPTION indication received.

8. E-n-exit-block

   A NONTX-STATEMENT-EXIT-BLOCK indication received.

9. E-n-exit-task

   A NONTX-STATEMENT-EXIT-TASK indication received.

10. E-rollb

    A TRANS-ROLLBACK-DONE indication received.

11. E-rollb-rec

    A TRANS-ROLLBACK-RECEIVED indication received.

12. E-t-canceled

    A TX-STATEMENT-CANCELED indication received.

13. E-t-done

    A TX-STATEMENT-DONE indication received.

14. E-t-exception

    A TX-STATEMENT-EXCEPTION indication received.

15. E-t-exit-block

    A TX-STATEMENT-EXIT-BLOCK indication received.

16. E-t-exit-task

    A TX-STATEMENT-EXIT-TASK indication received.

17. E-tx-timeout

    The transaction timeout has expired.

Actions

1. A-cancel-n

   Issue a NONTX-STATEMENT-CANCEL request.

2. A-cancel-t

   Issue a TX-STATEMENT-CANCEL request.

3. A-canceled

   Do one of the following:

   a. If the client is remote, issue a TASK-EXCEPTION indication with the following parameters:

      — Type = One of the following:

        — If this task is composable, Permanent Transaction

        — If this task is non-composable, Non-transaction

      — Class = ENV-EXECUTION-ERROR

      — Code = implementation-specific

      — Exception-Code-Group = implementation-specific

      — Source = System

      — Level = Current

      — Exception-Procedure = current task name

      — Exception-Procedure-Group = current task group name

   b. If the client is local, issue a TASK-CANCELED indication.

   Note that an implementation can choose to always issue the TASK-EXCEPTION indication with the listed parameters.

4. A-cleanup

   Do the following:

   a. Issue a PRES-TERMINATE request for each Presentation Procedure state machine instance in this execution.

      Note that an implementation can delay this action to a later time and reuse the Presentation Group context.

   b. Issue a CALL-RELEASE-CONTEXT request to each Call state machine instance in the execution context with V-client true.

      Note that an implementation can delay this action to a later time and reuse the non-transactional dialogue.

5. A-done

   Issue a TASK-DONE indication using the workspaces defined as OUTPUT or INOUT task arguments.

6. A-exception

   Do the following:

   a. Increment the Fault Count for the task if the exception class is one of the exception classes defined in the exception list as discussed in Section 17.2.17.1 on page 312.

   b. Issue a TASK-EXCEPTION indication with the following parameters:

      — Type = from V-current-exception, unless:

         — If the exception type is fatal and the task is composable, then permanent transaction

         — If the exception type is transient transaction and the task is composable and not restartable, then permanent transaction

         — If the task is non-composable, then non-transaction

      — Class = from V-current-exception

      — Code = from V-current-exception

      — Exception-Code-Group = from the V-current-exception

      — Source = from V-current-exception

      — Level = Propagated

      — Exception-Procedure = from V-current-exception

      — Exception-Procedure-Group = from V-current-exception

7. A-reject

   Issue a TASK-EXCEPTION indication with the following parameters:

   a. Type = from the Class parameter defined in Appendix C on page 489.

   b. Class is set as follows:

      — If task's composability is not correct = AP-EXECUTION-FAULT

      — If arguments passed are not valid = AP-INVOCATION-FAULT

      — If authorization is not valid or access control checks failed = ENV-INVOCATION-FAULT

      — The task is disabled = ENV-INVOCATION-ERROR

      — The high water mark is reached = ENV-INVOCATION-ERROR

   c. Code = implementation-specific

   d. Exception Code Group = implementation-specific

   e. Source = System

   f. Level = Current

   g. Exception-Procedure = current task name

   h. Exception-Procedure-Group = current task group name

8. A-release

   Issue a TASK-RELEASE-CONTEXT indication.

9. A-rollb

   Issue a TRANS-ROLLBACK request.

10. A-s-cpu-e

    Set V-current-exception with the following information:

    a. Type = Fatal Transaction
    b. Class = FATAL-TIMEOUT-FAULT
    c. Code = implementation-specific
    d. Exception Code Group = implementation-specific
    e. Source = System
    f. Level = Current
    g. Exception-Procedure = current task name
    h. Exception-Procedure-Group = current task group name

11. A-s-ind-e

    Set V-current-exception with the following information:

    a. Type = from the current indicator
    b. Class = from the current indicator
    c. Code = from the current indicator
    d. Exception Code Group = from the current indicator
    e. Source = from the current indicator
    f. Level = Propagated
    g. Exception-Procedure = from the current indicator
    h. Exception-Procedure-Group = from the current indicator

12. A-s-txtime-e

    Set V-current-exception with the following information:

    a. Type = Transient Transaction
    b. Class = TXN-TIMEOUT-ERROR
    c. Code = implementation-specific
    d. Exception Code Group = implementation-specific
    e. Source = System
    f. Level = Current
    g. Exception-Procedure = current task name
    h. Exception-Procedure-Group = current task group name

13. A-statements-n

    Issue a NONTX-STATEMENT-EXECUTE request.

    This generates the following internal event: E-cpu-timeout.

14. A-statements-t

    Issue a TX-STATEMENT-EXECUTE request.

    This generates one of the following internal events:

    a. E-cpu-timeout

    b. E-tx-timeout.

15. A-trans-n

    Issue a TRANS-START request with the following parameters:

    a. Type = Non-Root

States

1. S-null

   The null state

2. S-cr-statements

   Waiting for the top level statement list in a composable task from a remote client to finish.

3. S-cr-can-exc

   Waiting for the top level statement list in a composable task with a remote client to finish so that the task can terminate with an exception.

4. S-cr-can-can

   Waiting for the top level statement list in a composable task with a remote client to finish so that the task can terminate with a cancel acknowledgement.

5. S-cr-can-rollb

   Waiting for the top level statement list in a composable task with a remote client to finish after an unsolicited rollback.

6. S-cr-inactive

   Waiting for the transaction to be terminated or a new task to be called in the transaction after executing a composable task with a remote client.

7. S-cr-inactive-r

   Waiting for the transaction to be rolled back after a composable task with a remote client terminated with a transaction exception.

8. S-cr-rollb

   Waiting for the transaction to be terminated after issuing a TRANS-ROLLBACK request.

9. S-cl-statements

   Waiting for the top level statement list in a composable task with a local client to finish.

10. S-cl-can-exc

    Waiting for the top level statement list in a composable task with a local client to finish so the task can terminate with an exception.

11. S-cl-can-can

    Waiting for the top level statement list in a composable task with a local client to finish so that the task can terminate with a cancel acknowledgement.

12. S-n-statements

    Waiting for the top level statement list in non-composable task to finish.

13. S-n-can-exc

    Waiting for the top level statement list in a non-composable task to finish so the task can terminate with an exception.

14. S-n-can-can

    Waiting for the top level statement list in a non-composable task to finish so the task can terminate with a cancel acknowledgement.

State Transition Tables

	S-null
E-execute+P-valid+ P-trans+P-remote	S-cr-statements A-trans-n, A-statements-t
E-execute+P-valid+ P-trans+¬P-remote	S-cl-statements A-statements-t
E-execute+P-valid+ ¬P-trans	S-n-statements A-statements-n
E-execute+¬P-valid	S-null A-reject

Table 18-2 Task (Null State)

	S-cr-statements	**S-cr-can-exc**	**S-cr-can-can**	**S-cr-can-rollb**
E-cancel	S-cr-can-can A-cancel-t	S-cr-can-exc		
E-cpu-timeout	S-cr-can-exc A-s-cpu-e, A-cancel-t			
E-tx-timeout	S-cr-can-exc A-s-txtime-e, A-cancel-t			
E-t-done	S-cr-inactive A-done			
E-t-exit-task or E-t-exit-block	S-cr-inactive A-done			
E-t-exception+ P-n-exc	S-cr-inactive A-s-ind-e, A-exception			
E-t-exception+ (P-t-exc or P-p-exc)	S-cr-inactive-r A-s-ind-e, A-exception			
E-t-exception+ P-f-exc	S-cr-inactive-r A-s-ind-e, A-exception	S-cr-inactive-r A-s-ind-e, A-exception	S-cr-inactive-r (1)	S-cr-rollb A-rollb
E-t-canceled		S-cr-inactive-r A-exception	S-cr-inactive-r A-canceled	S-cr-rollb A-rollb
E-rollb-rec	S-cr-can-rollb A-cancel-t	S-cr-can-rollb	S-cr-can-rollb	

Table 18-3 Task (Composable, Remote Client, Statements and Cancel States)

	S-cr-inactive	**S-cr-inactive-r**	**S-cr-rollb**
E-execute+¬P-valid	S-cr-inactive-r A-reject		
E-execute+P-valid+ P-trans	S-cr-statements A-statements-t		
E-execute+P-valid+ ¬P-trans	S-cr-inactive-r A-reject		
E-commit	S-null A-cleanup, A-release		
E-rollb-rec	S-cr-rollb A-rollb	S-cr-rollb A-rollb	
E-rollb			S-null A-cleanup, A-release

Table 18-4 Task (Composable, Remote Client, Inactive States)

	S-cl-statements	**S-cl-can-exc**	**S-cl-can-can**
E-cancel	S-cl-can-can A-cancel-t	S-cl-can-exc	
E-cpu-timeout	S-cl-can-exc A-s-cpu-e, A-cancel-t		
E-tx-timeout	S-cl-can-exc A-s-txtime-e, A-cancel-t		
E-t-done	S-null A-done		
E-t-exit-task or E-t-exit-block	S-null A-done		
E-t-exception+ ¬P-f-exc	S-null A-s-ind-e, A-exception		
E-t-exception+ P-f-exc	S-null A-s-ind-e, A-exception	S-null A-s-ind-e, A-exception	S-null (1)
E-t-canceled		S-null A-exception	S-null A-canceled

Table 18-5 Task (Composable, Local Client States)

	S-n-statements	**S-n-can-exc**	**S-n-can-can**
E-cancel	S-n-can-can A-cancel-n	S-n-can-exc	
E-cpu-timeout	S-n-can-exc A-s-cpu-e, A-cancel-n		
E-n-done	S-null A-cleanup, A-done	S-null (2)	S-null (3)
E-n-exception+ ¬P-f-exc	S-null A-s-ind-e, A-cleanup, A-exception		
E-n-exception+ P-f-exc	S-null A-s-ind-e, A-cleanup, A-exception	S-null A-s-ind-e, A-cleanup, A-exception	S-null (4)
E-n-exit-task or E-n-exit-block	S-null A-cleanup, A-done		
E-n-canceled		S-null A-cleanup, A-exception	S-null A-cleanup, A-canceled

Table 18-6 Task (Non-composable Task State)

Notes:

1. An implementation can do either A-set-ind-e, A-exception or A-canceled
2. An implementation can do either A-cleanup, A-done or A-cleanup, A-exception
3. An implementation can do either A-cleanup, A-done or A-cleanup, A-canceled
4. An implementation can do either A-s-ind-e, A-cleanup, A-exception or A-cleanup, A-canceled

18.7.2 Non-transactional Statement List State Machine

The Non-transactional Statement List state machine models the processing done to execute a <statement-list> outside of a transaction. This includes all <statement-list>s that are outside of <transaction-block>s in <noncomposable-task>s.

Multiple Non-transactional Statement List state machine instances can exist within an execution context. Each NONTX-STATEMENT-EXECUTE request creates a separate Non-transactional Statement List state machine instance.

Service Primitives Provided

1. NONTX-STATEMENT-EXECUTE — request

   Executes a <statement-list> outside of a transaction.

   This service primitive has no parameters.

2. NONTX-STATEMENT-CANCEL — request

   Cancels the execution of a non-transactional statement list.

   This service primitive has no parameters.

3. NONTX-STATEMENT-CANCELED — indication

   Reports that the non-transactional statement list has been cancelled.

   This indication has no parameters.

4. NONTX-STATEMENT-DONE — indication

   Reports that the non-transactional statement list terminated with default flow control.

   This indication has no parameters.

5. NONTX-STATEMENT-EXCEPTION — indication

   Reports that the non-transactional statement list terminated with an exception.

   This indication has the following parameters:

   a. Type = Non-transaction or Fatal Transaction

   b. Class = one of the classes defined in Appendix C on page 489.

   c. Code = the exception code as defined in Section 3.7.2 on page 62.

   d. Exception Code Group = the exception code group as defined in Section 3.7.2 on page 62.

   e. Source = System or Application

   f. Level = Current or Propagated

   g. Exception-Procedure = the procedure in which the exception occurred as defined in Section 3.7.2 on page 62.

   h. Exception-Procedure-Group = the procedure group in which the exception occurred as defined in Section 3.7.2 on page 62.

6. NONTX-STATEMENT-EXIT-BLOCK — indication

   Reports that an <exit-block> has been encountered.

This indication has no parameters.

7. NONTX-STATEMENT-EXIT-TASK — indication

   Reports that an <exit-task> has been encountered.

   This indication has no parameters.

8. NONTX-STATEMENT-GOTO — indication

   Reports that a <go-to> statement has been encountered.

   This indication has one parameter:

   a. Label = the label to which control is to be transferred.

Service User

The service user of the Non-transactional Statement List state machine is the state machine instance that issued the NONTX-STATEMENT-EXECUTE request. This service user receives all indications generated by a Non-transactional Statement List state machine instance and the NONTX-STATEMENT-CANCEL request from that service user is directed to the Non-transactional Statement List state machine instance that was created for that service user.

State Machines Used

1. Non-transaction Statement List (NONTX-STATEMENT-)
2. Transaction (TRANS-)
3. Transactional Statement List (TX-STATEMENT-)
4. Transactional Receive (RR-)

Variables

1. V-current-exception

   Contains the Type, Class, Code, Exception Code Group, Source, Level, Exception-Procedure, and Exception-Procedure-Group of the current exception. This variable stores data.

2. V-label

   Contains the label of any currently executing <go-to>.

   This variable stores data.

3. V-restart-count

   This is the number of times a transaction has been restarted.

   This variable stores data.

Predicates

1. P-block

   Is true if the next <statement> to be executed is <statement-block>.

2. P-conditional

   Is true if the next <statement> to be executed is <if>, <select-first> or <control-field>.

3. P-f-exc

   Is true if the exception has a type of fatal transaction.

4. P-f-exc-cur

   Is true if V-current-exception has a type of fatal transaction.

5. P-hand

   Is true if the <statement-block> or <transaction-block> has an exception handler.

6. P-label

   Is true if the label in a NONTX-STATEMENT-GOTO indication or a <go-to> is in the <statement-list> being executed.

7. P-loop

   Is true if the next <statement> is <while>.

8. P-more

   Is true if there are more <statement>s in the <statement-list> to execute.

9. P-n-exc

   Is true if the exception has a type of non-transaction.

10. P-p-exc

    Is true if the exception has a type of permanent transaction.

11. P-rc

    Is true if V-restart-count is not greater than the Restart Limit as defined in Section 17.2.17.1 on page 312 and the task is restartable.

12. P-t-exc

    Is true if the exception has a type of transient transaction.

13. P-tx-block

    Is true if the next <statement> to be executed is <transaction-block>.

14. P-work

    Is true if the next <statement> to be executed is <assignment>, <audit>, <get-message>, <exit-block>, <exit-task>, <go-to>, <raise-exception> or <reraise-exception>.

Events

1. E-cancel

   A NONTX-STATEMENT-CANCEL request received.

2. E-canceled

   Received one of the following:

   a. NONTX-STATEMENT-CANCELED indication.

   b. TX-STATEMENT-CANCELED indication

3. E-commit

   A TRANS-COMMIT-DONE indication received.

4. E-cond-false

   One of the following:

   a. The <boolean-expression> within the <if> is false and there is no <else>

   b. None of the <boolean-expression>s within the <select-first> evaluate to true and there is not a NOMATCH

   c. None of the <control-field-value>s match the <control-field-to-match> in a <control-field> and there is not a NOMATCH

5. E-cond-true

   One of the following:

   a. The <boolean-expression> within the <if> is true or there is an <else>

   b. At least one of the <boolean-expression>s within the <select-first> evaluate to true or there is a NOMATCH

   c. At least one of the <control-field-value>s match the <control-field-to-match> in a <control-field> or there is a NOMATCH

6. E-done

   One of the following occurred:

   a. Received a NONTX-STATEMENT-DONE indication.

   b. Received a TX-STATEMENT-DONE indication

7. E-error

   An exception occurred during the execution of a statement as defined in Chapter 10 on page 167. This can occur on internally executed statements or statements that use other state machines if the error occurs in this state machine (for example an index out of range on in a <boolean-expression> of an <if>).

8. E-exception

   Received one of the following

   a. NONTX-STATEMENT-EXCEPTION indication.

   b. TX-STATEMENT-EXCEPTION indication.

9. E-execute

A NONTX-STATEMENT-EXECUTE request received.

10. E-exit-block

    Received a NONTX-STATEMENT-EXIT-BLOCK indication.

11. E-exit-block-c

    Received one of the following:

    a. TX-STATEMENT-EXIT-BLOCK indication.

    b. TX-STATEMENT-EXIT-BLOCK-COMMIT indication.

12. E-exit-block-r

    Received a TX-STATEMENT-EXIT-BLOCK-ROLLBACK indication.

13. E-exit-task

    Received a NONTX-STATEMENT-EXIT-TASK indication.

14. E-exit-task-c

    Received one of the following:

    a. TX-STATEMENT-EXIT-TASK indication

    b. TX-STATEMENT-EXIT-TASK-COMMIT indication

15. E-exit-task-r

    Received a TX-STATEMENT-EXIT-TASK-ROLLBACK indication

16. E-goto

    Received a NONTX-STATEMENT-GOTO indication

17. E-goto-c Received one of the following:

    a. TX-STATEMENT-GOTO indication

    b. TX-STATEMENT-GOTO-COMMIT indication

18. E-goto-r

    Received a TX-STATEMENT-GOTO-ROLLBACK indication

19. E-loop-false

    The <boolean-expression> in a <while> evaluated to false.

20. E-loop-true

    The <boolean-expression> in a <while> evaluated to true.

21. E-rollb

    A TRANS-ROLLBACK-DONE indication received.

22. E-rollb-rec

    A TRANS-ROLLBACK-RECEIVED indication received.

23. E-tx-timeout

    The transaction timeout has expired.

24. E-work-done

    The internal work of an <assignment>, <audit> or <get-message> statement is done.

25. E-work-exit-block

    The internal work of an <exit-block> is done.

26. E-work-exit-task

    The internal work of an <exit-task> is done.

27. E-work-goto

    The internal work of a <go-to> is done.

28. E-work-raise

    The internal work of a <reraise-exception> or <raise-exception> statement is done.

Actions

1. A-cancel-nontx-s

   Issue a NONTX-STATEMENT-CANCEL request.

2. A-cancel-tx-s

   Issue a TX-STATEMENT-CANCEL request.

3. A-canceled

   Issue a NONTX-STATEMENT-CANCELED indication.

4. A-cond-evaluate

   Evaluate the control statement to determine whether or not there is a <statement-list> within the control statement that will be executed.

   This evaluation generates one of the following events:

   a. E-cond-true

   b. E-cond-false

   c. E-error

5. A-commit

   Issue a TRANS-COMMIT request.

6. A-done

   Issue NONTX-STATEMENT-DONE indication.

7. A-exception

   Issue a NONTX-STATEMENT-EXCEPTION indication with the following parameters:

   a. Type = one of the following:

      — Fatal, if the Type of V-exception is Fatal

— Non-transaction, if the Type of V-exception is not Fatal

b. Class = from V-exception

c. Code = from V-exception

d. Exception Code Group = from V-exception

e. Source = from V-exception

f. Level = from V-exception

g. Exception-Procedure = from V-exception

h. Exception-Procedure-Group = from V-exception

8. A-exit-block

   Issue a NONTX-STATEMENT-EXIT-BLOCK indication.

9. A-exit-task

   Issue a NONTX-STATEMENT-EXIT-TASK indication.

10. A-goto

    Issue a NONTX-STATEMENT-GOTO indication using the label stored in V-label.

11. A-loop-boolean

    Evaluate the <boolean-expression> of the <while> statement.

    This evaluation generates one of the following events:

    a. E-loop-true

    b. E-loop-false

    c. E-error

12. A-rollb

    Issue a TRANS-ROLLBACK request

13. A-set-ind-exc

    Set V-current-exception with the parameters from the indication.

14. A-set-int-exc

    Set V-current-exception with the following information:

    a. Type = non-transaction

    b. Class = as defined for the exception in Chapter 10 on page 167.

    c. Code = implementation-specific

    d. Exception Code Group = implementation-specific

    e. Source = System

    f. Level = Current

    g. Exception-Procedure = current task name

    h. Exception-Procedure-Group = current task group name

15. A-set-label

    Save the label from the indication in V-label.

16. A-set-rollb-exc

    Sets V-current-exception as follows:

    a. Type = Permanent transaction
    b. Class = TXN-FAILURE-ERROR
    c. Code = implementation-specific
    d. Exception-Code-Group = implementation-specific
    e. Source = System
    f. Level = Current
    g. Exception-Procedure = current task name
    h. Exception-Procedure-Group = current task group name

17. A-set-txtime-exc

    Set V-current-exception with the following information:

    a. Type = Transient Transaction
    b. Class = TXN-TIMEOUT-ERROR
    c. Code = implementation-specific
    d. Exception-Code-Group = implementation-specific
    e. Source = System
    f. Level = Current
    g. Exception-Procedure = current task identifier
    h. Exception-Procedure-Group = current task group identifier

18. A-statements

    Issue NONTX-STATEMENT-EXECUTE request

19. A-tx-block

    Do the following:

    a. Set V-restart-count to 0
    b. Issue an RR-RESET request
    c. Issue a TRANS-START request with the following parameter:
       — Type = Root
    d. Issue TX-STATEMENT-EXECUTE request

    This generates the internal event: E-tx-timeout.

20. A-tx-block-r

    Do the following:

a. Increment V-restart-count

b. Issue a TRANS-START request with the following parameter:

   a. Type = Root

c. Issue TX-STATEMENT-EXECUTE request

This generates the internal event: E-tx-timeout.

21. A-work

Depends on the statement:

a. <assignment>, <audit> or <get-message>

   Do one of the following:

   — If an exception occurs as defined in Chapter 10 on page 167, generate an E-error event.

   — Else, generate an E-work-done event.

b. <reraise-exception>

   Generate an E-work-raise-event.

c. <raise-exception>

   Do one of the following:

   — If an exception occurs as defined in Chapter 10 on page 167, generate an E-error event.

   — Else, set V-current-exception with the information from <exception-information> and generate an E-work-raise event.

d. <exit-block>

   Generate an E-work-exit-block event.

e. <exit-task>

   Generate an E-work-exit-task event.

f. <go-to>

   Do the following:

   — Set the label in V-label.

   — Generate an E-work-goto event

States

1. S-null

   The initial state.

2. S-block

   Waiting for the <statement-list> in a <statement-block> to complete.

3. S-handler

   Waiting for the <exception-handler> on a block statement to complete.

4. S-cond-evaluate

   Waiting for the evaluation of the <if>, <control-field> or <select-first> Boolean or matching to complete.

5. S-cond-execute

   Waiting for the <statement-list> within the <if>, <control-field> or <select-first> to complete.

6. S-loop-boolean

   Waiting for the <while> Boolean to complete.

7. S-loop-execute

   Waiting for the <statement-list> in the <while> to complete.

8. S-work

   Waiting for the <assignment>, <audit>, <get-message>, <reraise-exception>, <raise-exception>, <exit-block>, <exit-task> or <go-to> statement to complete.

9. S-cancel

   Waiting for the current non-transactional <statement> to cancel.

10. S-tx-block

    Waiting for the <statement-list> in a <transaction-block> to complete.

11. S-commit

    Waiting for the transaction to finish after issuing a commit before executing the next <statement> in the current non-transactional <statement-list>.

12. S-commit-can

    Waiting for the transaction to finish after receiving a cancel in the S-commit state.

13. S-com-exit

    Waiting for the transaction to finish after issuing a commit so that the task can be terminated with an exit task indication. Any errors in the commit result in an exception being generated within the current task.

14. S-com-exit-can

    Waiting for the transaction to finish after receiving a cancel in the S-com-exit state.

15. S-com-goto

    Waiting for the transaction to finish after issuing a commit when the commit was caused by a <go-to>.

16. S-com-goto-can

    Waiting for the transaction to finish after receiving a cancel in the S-com-goto state.

17. S-rollb

    Waiting for the transaction to finish after issuing a rollback before executing the next <statement> in the current non-transactional <statement-list>.

18. S-rollb-can

Waiting for the transaction to finish after receiving a cancel in the S-rollb state.

19. S-rollb-exit

    Waiting for the transaction to finish after issuing a rollback so that the current task can be terminated normally.

20. S-rollb-exit-can

    Waiting for the transaction to finish after receiving a cancel in the S-rollb-exit state.

21. S-rollb-goto

    Waiting for the transaction to finish after issuing a rollback when the rollback was caused by a <go-to>.

22. S-rollb-goto-can

    Waiting for the transaction to finish after receiving a cancel in the S-rollb-goto state.

23. S-can-tx-restart

    Waiting for the <statement-list> in a <transaction-block> to cancel before restarting the <transaction-block>.

24. S-can-tx-hand

    Waiting for the <statement-list> in a <transaction-block> to cancel before executing an exception handler.

25. S-can-tx-can

    Waiting for the <statement-list> in a <transaction-block> to cancel before completing with a cancel acknowledgement.

26. S-rollb-tx-restart

    Waiting for the transaction to finish after issuing a rollback so that the <transaction-block> can be restarted.

27. S-rollb-tx-hand

    Waiting for the transaction to finish after issuing a rollback so that an exception handler can be executed.

28. S-rollb-tx-can

    Waiting for the transaction to finish after issuing a rollback so that the <statement-list> can be completed with a cancel acknowledgement.

29. S-can-tx-exc

    Waiting for the <statement-list> in a <transaction-block> to cancel before completing with an exception.

30. S-rollb-tx-exc

    Waiting for the transaction to finish after issuing a rollback so that the <statement-list> can be terminated with an exception.

State Transition Tables

	S-null
E-execute+ ¬P-more	S-null A-done
E-execute+ P-tx-block	S-tx-block A-tx-block
E-execute+ P-block	S-block A-statements
E-execute+ P-conditional	S-cond-evaluate A-cond-evaluate
E-execute+ P-loop	S-loop-boolean A-loop-boolean
E-execute+ P-work	S-work A-work

Table 18-7 Non-transactional Statement List (Null State)

	S-block
E-done+¬P-more or E-exit-block+¬P-more	S-null A-done
E-done+P-tx-block or E-goto+P-label+P-tx-block or E-exit-block+P-tx-block	S-tx-block A-tx-block
E-done+P-block or E-goto+P-label+P-block or E-exit-block+P-block	S-block A-statements
E-done+P-conditional or E-goto+P-label+P-conditional or E-exit-block+P-conditional	S-cond-evaluate A-cond-evaluate
E-done+P-loop or E-goto+P-label+P-loop or E-exit-block+P-loop	S-loop-boolean A-loop-boolean
E-done+P-work or E-goto+P-label+P-work or E-exit-block+P-work	S-work A-work
E-goto+¬P-label	S-null A-set-label, A-goto
E-exit-task	S-null A-exit-task
E-exception+P-n-exc +P-hand	S-handler A-set-ind-exc, A-statements
E-exception+P-n-exc +¬P-hand	S-null A-set-ind-exc, A-exception
E-exception+P-f-exc	S-null A-set-ind-exc, A-exception
E-cancel	S-cancel A-cancel-nontx-s

Table 18-8 Non-transactional Statement List (Block State)

	S-handler
E-done+¬P-more	S-null A-done
E-done+P-tx-block or E-goto+P-label+P-tx-block	S-tx-block A-tx-block
E-done+P-block or E-goto+P-label+P-block	S-block A-statements
E-done+P-conditional or E-goto+P-label+P-conditional	S-cond-evaluate A-cond-evaluate
E-done+P-loop or E-goto+P-label+P-loop	S-loop-boolean A-loop-boolean
E-done+P-work or E-goto+P-label+P-work	S-work A-work
E-goto+ ¬P-label	S-null A-set-label, A-goto
E-exception	S-null A-set-ind-exc, A-exception
E-exit-block	S-null A-exit-block
E-exit-task	S-null A-exit-task
E-cancel	S-cancel A-cancel-nontx-s

Table 18-9 Non-transactional Statement List (Handler State)

	S-cond-evaluate	**S-cond-execute**
E-cond-true	S-cond-execute A-statements	
E-cond-false+ ¬P-more	S-null A-done	
E-cond-false+ P-tx-block	S-tx-block A-tx-block	
E-cond-false+ P-block	S-block A-statements	
E-cond-false+ P-conditional	S-cond-evaluate A-cond-evaluate	
E-cond-false+ P-loop	S-loop-boolean A-loop-boolean	
E-cond-false+ P-work	S-work A-work	
E-error	S-null A-set-int-exc, A-exception	
E-done+¬P-more		S-null A-done
E-done+P-tx-block or E-goto+P-label+P-tx-block		S-tx-block A-tx-block
E-done+P-block or E-goto+P-label+P-block		S-block A-statements
E-done+P-conditional or E-goto+P-label+P-conditional		S-cond-evaluate A-cond-evaluate
E-done+P-loop or E-goto+P-label+P-loop		S-loop-boolean A-loop-boolean
E-done+P-work or E-goto+P-label+P-work		S-work A-work
E-goto+¬P-label		S-null A-set-label, A-goto
E-exception		S-null A-set-ind-exc,A-exception
E-exit-block		S-null A-exit-block
E-exit-task		S-null A-exit-task
E-cancel	S-null A-canceled	S-cancel A-cancel-nontx-s

Table 18-10 Non-transactional Statement List (Conditional States)

	S-loop-boolean	**S-loop-execute**
E-loop-true	S-loop-execute A-statements	
E-loop-false+ ¬P-more	S-null A-done	
E-loop-false+ P-tx-block	S-tx-block A-tx-block	
E-loop-false+ P-block	S-block A-statements	
E-loop-false+ P-conditional	S-cond-evaluate A-cond-evaluate	
E-loop-false+ P-loop	S-loop-boolean A-loop-boolean	
E-loop-false+ P-work	S-work A-work	
E-error	S-null A-set-int-exc, A-exception	
E-done		S-loop-boolean A-loop-boolean
E-exception		S-null A-set-ind-exc, A-exception
E-exit-block		S-null A-exit-block
E-exit-task		S-null A-exit-task
E-goto+ ¬P-label		S-null A-set-label, A-goto
E-goto+P-label+ P-tx-block		S-tx-block A-tx-block
E-goto+P-label+ P-block		S-block A-statements
E-goto+P-label+ P-conditional		S-cond-evaluate A-cond-evaluate
E-goto+P-label+ P-loop		S-loop-boolean A-loop-boolean
E-goto+P-label+ P-work		S-work A-work
E-cancel	S-null A-canceled	S-cancel A-cancel-nontx-s

Table 18-11 Non-transactional Statement List (Loop States)

	S-work
E-work-done+¬P-more	S-null A-done
E-work-done+P-tx-block or E-work-goto+P-label+P-tx-block	S-tx-block A-tx-block
E-work-done+P-block or E-work-goto+P-label+P-block	S-block A-statements
E-work-done+P-conditional or E-work-goto+P-label+P-conditional	S-cond-evaluate A-cond-evaluate
E-work-done+P-loop or E-work-goto+P-label+P-loop	S-loop-boolean A-loop-boolean
E-work-done+P-work or E-work-goto+P-label+P-work	S-work A-work
E-work-goto+¬P-label	S-null A-goto
E-work-exit-block	S-null A-exit-block
E-work-exit-task	S-null A-exit-task
E-work-raise	S-null A-exception
E-error	S-null A-set-int-exc, A-exception
E-cancel	S-null A-canceled

Table 18-12 Non-transactional Statement List (Work State)

	S-cancel
E-done+ P-more	S-null A-canceled
E-done+ ¬P-more	S-null A-done
E-exit-task	S-null A-exit-task
E-exception+ P-f-exc	S-null A-set-ind-exc, A-exception
E-canceled	S-null A-canceled

Table 18-13 Non-transactional Statement List (Cancel State)

	S-tx-block
E-done	S-commit A-commit
E-exit-block-c	S-commit A-commit
E-exit-block-r	S-rollb A-rollb
E-exit-task-c	S-com-exit A-commit
E-exit-task-r	S-rollb-exit A-rollb
E-goto-c	S-com-goto A-set-label, A-commit
E-goto-r	S-rollb-goto A-set-label, A-rollb
E-exception+(P-n-exc or P-p-exc)+P-hand	S-rollb-tx-hand A-set-ind-exc, A-rollb
E-exception+(P-n-exc or P-p-exc)+¬P-hand	S-rollb-tx-exc A-set-ind-exc, A-rollb
E-exception+P-t-exc+ P-rc	S-rollb-tx-restart A-rollb
E-exception+P-t-exc+ ¬P-rc+P-hand	S-rollb-tx-hand A-set-ind-exc, A-rollb
E-exception+P-t-exc+ ¬P-rc+¬P-hand	S-rollb-tx-exc A-set-ind-exc, A-rollb
E-exception+P-f-exc	S-rollb-tx-exc A-set-ind-exc, A-rollb
E-tx-timeout+ P-rc	S-can-tx-restart A-cancel-tx-s
E-tx-timeout+ ¬P-rc+P-hand	S-can-tx-hand A-set-txtime-exc, A-cancel-tx-s
E-tx-timeout+ ¬P-rc+¬P-hand	S-can-tx-exc A-set-txtime-exc, A-cancel-tx-s
E-cancel	S-can-tx-can A-cancel-tx-s
E-rollb-rec+ P-hand	S-can-tx-hand A-set-rollb-exc, A-cancel-tx-s
E-rollb-rec+ ¬P-hand	S-can-tx-exc A-set-rollb-exc, A-cancel-tx-s

Table 18-14 Non-transactional Statement List (Transaction Block State)

	S-commit	**S-commit-can**
E-commit + ¬P-more	S-null A-done	S-null A-done
E-commit + P-tx-block	S-tx-block A-tx-block	S-null A-canceled
E-commit + P-block	S-block A-statements	S-null A-canceled
E-commit + P-conditional	S-cond-evaluate A-cond-evaluate	S-null A-canceled
E-commit + P-loop	S-loop-boolean A-loop-boolean	S-null A-canceled
E-commit + P-work	S-work A-work	S-null A-canceled
E-rollb+ P-hand	S-handler A-set-rollb-exc, A-statements	S-null A-canceled
E-rollb+ ¬P-hand	S-null A-set-rollb-exc, A-exception	S-null A-canceled
E-cancel	S-commit-can	

Table 18-15 Non-transaction Statement List (Transaction Block Standard Commit States)

	S-com-exit	**S-com-exit-can**
E-commit	S-null A-exit-task	S-null A-exit-task
E-rollb+ P-hand	S-handler A-set-rollb-exc, A-statements	S-null A-canceled
E-rollb+ ¬P-hand	S-null A-set-rollb-exc, A-exception	S-null A-canceled
E-cancel	S-com-exit-can	

Table 18-16 Non-transaction Statement List (Transaction Block Task Exit Commit States)

	S-com-goto	**S-com-goto-can**
E-commit+ ¬P-label	S-null A-goto	S-null A-canceled
E-commit+P-label+ P-tx-block	S-tx-block A-tx-block	S-null A-canceled
E-commit+P-label+ P-block	S-block A-statements	S-null A-canceled
E-commit+P-label+ P-conditional	S-cond-evaluate A-cond-evaluate	S-null A-canceled
E-commit+P-label+ P-loop	S-loop-boolean A-loop-boolean	S-null A-canceled
E-commit+P-label+ P-work	S-work A-work	S-null A-canceled
E-rollb+ P-hand	S-handler A-set-rollb-exc, A-statements	S-null A-canceled
E-rollb+ ¬P-hand	S-null A-set-rollb-exc, A-exception	S-null A-canceled
E-cancel	S-com-goto-can	

Table 18-17 Non-transaction Statement List (Transaction Block Go To Commit State)

	S-rollb	**S-rollb-can**
E-rollb + ¬P-more	S-null A-done	S-null A-done
E-rollb+ P-tx-block	S-tx-block A-tx-block	S-null A-canceled
E-rollb+ P-block	S-block A-statements	S-null A-canceled
E-rollb+ P-conditional	S-cond-evaluate A-cond-evaluate	S-null A-canceled
E-rollb+ P-loop	S-loop-boolean A-loop-boolean	S-null A-canceled
E-rollb+ P-work	S-work A-work	S-null A-canceled
E-cancel	S-rollb-can	

Table 18-18 Non-transaction Statement List (Transaction Block Standard Rollback States)

	S-rollb-exit	**S-rollb-exit-can**
E-rollb	S-null A-exit-task	S-null A-exit-task
E-cancel	S-rollb-exit-can	

Table 18-19 Non-transaction Statement List (Transaction Block Task Exit Rollback States)

	S-rollb-goto	**S-rollb-goto-can**
E-rollb+ ¬P-label	S-null A-goto	S-null A-canceled
E-rollb+P-label+ P-tx-block	S-tx-block A-tx-block	S-null A-canceled
E-rollb+P-label+ P-block	S-block A-statements	S-null A-canceled
E-rollb+P-label+ P-conditional	S-cond-evaluate A-cond-evaluate	S-null A-canceled
E-rollb+P-label+ P-loop	S-loop-boolean A-loop-boolean	S-null A-canceled
E-rollb+P-label+ P-work	S-work A-work	S-null A-canceled
E-cancel	S-rollb-goto-can	

Table 18-20 Non-transaction Statement List (Transaction Block Go To Rollback States)

	S-can-tx-restart	**S-can-tx-hand**	**S-can-tx-can**
E-rollb-rec	S-can-tx-restart	S-can-tx-hand	S-can-tx-can
E-exception+ P-f-exc	S-rollb-tx-exc A-set-ind-exc, A-rollb	S-rollb-tx-exc A-set-ind-exc, A-rollb	S-rollb-tx-exc A-set-ind-exc, A-rollb
E-canceled	S-rollb-tx-restart A-rollb	S-rollb-tx-hand A-rollb	S-rollb-tx-can A-rollb
E-cancel	S-can-tx-can	S-can-tx-can	

Table 18-21 Non-transaction Statement List (Transaction Block Non-exception Cancel States)

	S-rollb-tx-restart	**S-rollb-tx-hand**	**S-rollb-tx-can**
E-rollb	S-tx-block A-tx-block-r	S-handler A-statements	S-null A-canceled
E-cancel	S-rollb-tx-can	S-rollb-tx-can	

Table 18-22 Non-trans. Stmt. List (Trans. Block Rollback for Restart, Handler, Exception, Cancel States)

	S-can-tx-exc	**S-rollb-tx-exc**
E-rollb-rec	S-can-tx-exc	
E-exception+ P-f-exc	S-rollb-tx-exc A-set-ind-exc, A-rollb	
E-canceled	S-rollb-tx-exc A-rollb	
E-cancel+ ¬P-f-exc-cur	S-can-tx-can	S-rollb-tx-can
E-cancel+ P-f-exc-cur	S-can-tx-exc	S-rollb-tx-exc
E-rollb		S-null A-exception

Table 18-23 Non-transaction Statement List (Transaction Block Exception States)

18.7.3 Transactional Statement List State Machine

The Transactional Statement List state machine models the processing done to execute a <statement-list> within a transaction. This includes all <statement-list>s that are in (or nested in) <transaction-block>s and <composable-task>s.

Multiple Transactional Statement List state machine instances can exist within an execution context. Each TX-STATEMENT-EXECUTE request creates a separate Transactional Statement List state machine instance.

Service Primitives Provided

1. TX-STATEMENT-EXECUTE — request

   Executes a <statement-list> inside a transaction.

   This service primitive has no parameters.

2. TX-STATEMENT-CANCEL — request

   Cancels the execution of a transactional statement list.

   This service primitive has no parameters.

3. TX-STATEMENT-CANCELED — indication

   Reports that the transactional statement list has been canceled.

   This indication has no parameters.

4. TX-STATEMENT-DONE — indication

   Reports that the transactional statement list terminated with default flow control.

   This indication has no parameters.

5. TX-STATEMENT-EXCEPTION — indication

   Reports that the transactional statement list terminated with an exception.

   This indication has the following parameters:

   a. Type = Non-transaction, Transient Transaction, Permanent Transaction or Fatal Transaction

   b. Class = one of the classes defined in Appendix C on page 489.

   c. Code = the exception code as defined in Section 3.7.2 on page 62.

   d. Exception-Code-Group = the exception code group as defined in Section 3.7.2 on page 62.

   e. Source = System or Application

   f. Level = Current or Propagated

   g. Exception-Procedure = the procedure in which the exception occurred as defined in Section 3.7.2 on page 62.

   h. Exception-Procedure-Group = the procedure group in which the exception occurred as defined in Section 3.7.2 on page 62.

6. TX-STATEMENT-EXIT-BLOCK — indication

   Reports that an <exit-block> without <transaction-control> was encountered.

This indication has no parameters.

7. TX-STATEMENT-EXIT-BLOCK-COMMIT — indication

   Reports that an EXIT BLOCK WITH COMMIT statement has been encountered.

   This indication has no parameters.

8. TX-STATEMENT-EXIT-BLOCK-ROLLBACK — indication

   Reports that an EXIT BLOCK WITH ROLLBACK statement has been encountered.

   This indication has no parameters.

9. TX-STATEMENT-EXIT-TASK — indication

   Reports that an <exit-task> without <transaction-control> has been encountered.

   This indication has no parameters.

10. TX-STATEMENT-EXIT-TASK-COMMIT — indication

    Reports that an EXIT TASK WITH COMMIT statement has been encountered.

    This indication has no parameters.

11. TX-STATEMENT-EXIT-TASK-ROLLBACK — indication

    Reports that an EXIT TASK WITH ROLLBACK statement has been encountered.

    This indication has no parameters.

12. TX-STATEMENT-GOTO — indication

    Reports that a <go-to> statement without <transaction-control> has been encountered.

    This indication has one parameter:

    a. Label = the label to which control is to be transferred.

13. TX-STATEMENT-GOTO-COMMIT — indication

    Reports that a GO TO <label-identifier> WITH COMMIT statement has been encountered.

    This indication has one parameter:

    a. Label = the label to which control is to be transferred.

14. TX-STATEMENT-GOTO-ROLLBACK — indication

    Reports that a GO TO <label-identifier> WITH ROLLBACK statement has been encountered.

    This indication has one parameter:

    a. Label = the label to which control is to be transferred.

Service User

The service user of the Transactional Statement List state machine is the state machine instance that issued the TX-STATEMENT-EXECUTE request. This service user receives all indications generated by a Transactional Statement List state machine instance and the TX-STATEMENT-CANCEL request from that service user is directed to the Transactional Statement List state machine instance that was created for that service user.

State Machines Used

1. Call (CALL-)
2. Concurrent Block (CONC-)
3. Presentation Procedure (PRES-)
4. Processing Procedure (PROC-)
5. Transactional Receive (RR-)
6. Resource Manager (RM-)
7. Transactional Statement List (TX-STATEMENT-)

Variables

1. V-work-type

   The state machine executing the external work. This can take one of the following values:

   a. C-call = the Call state machine
   b. C-presentation-call = the Presentation Procedure state machine
   c. C-processing-call = the Processing Procedure state machine
   d. C-transactional-receive = the Transactional Receive state machine
   e. C-RM = the Resource Manager state machine

   This variable stores data.

2. V-current-exception

   Contains the Type, Class, Code, Exception-Code-Group, Source, Level, Exception-Procedure, and Exception-Procedure-Group of the current exception.

   This variable stores data.

3. V-label

   Contains the label of any currently executing <go-to>.

   This variable stores data.

4. V-tx-control

   Contains the type of <transaction-control> for the currently executing statement (if any). This can take one of the following values:

   a. C-tx-none = the statement has no <transaction-control>
   b. C-tx-commit = the statement has COMMIT <transaction-control>
   c. C-tx-rollback = the statement has ROLLBACK <transaction-control>

   This variable stores data.

Predicates

1. P-block

   Is true if the next <statement> to be executed is <statement-block>.

2. P-conc-block

   Is true if the next <statement> to be executed is <concurrent-block>.

3. P-conditional

   Is true if the next <statement> to be executed is <if>, <select-first> or <control-field>.

4. P-fatal

   Is true if the exception has a type of fatal transaction.

5. P-hand

   Is true if the <statement-block> or <concurrent-block> has an exception handler.

6. P-label

   Is true if the label in a TX-STATEMENT-GOTO indication or a <go-to> is in the <statement-list> being executed.

7. P-loop

   Is true if the next <statement> is <while>.

8. P-more

   Is true if there are more <statement>s in the <statement-list> to execute.

9. P-non-trans

   Is true if the exception has a type of non-transaction.

10. P-tx-none

    Is true if V-tx-control is C-tx-none.

11. P-tx-commit

    Is true if V-tx-control is C-tx-commit.

12. P-tx-rollback

    Is true if V-tx-control is C-tx-rollback.

13. P-work-external

    Is true if the next <statement> to be executed is <call-procedure>, <call-task>, <submit-task>, <cancel-submit>, <enqueue-record>, <dequeue-record>, <read-queue-record> or <call-presentation-procedure>.

14. P-work-internal

    Is true if the next <statement> to be executed is <assignment>, <audit>, <get-message>, <reraise-exception>, <restart-transaction>, <raise-exception>, <exit-block>, <exit-task> or <go-to>.

Events

1. E-cancel

   A TX-STATEMENT-CANCEL request received.

2. E-canceled

   Received one of the following:

   a. CALL-CANCELED indication
   b. CONC-CANCELED indication
   c. PRES-CANCELED indication
   d. PROC-CANCELED indication
   e. RM-CANCELED indication
   f. RR-CANCELED indication
   g. TX-STATEMENT-CANCELED indication

3. E-cond-false

   One of the following:

   a. The <boolean-expression> within the <if> is false and there is no <else>
   b. None of the <boolean-expression>s within the <select-first> evaluate to true and there is not a NOMATCH
   c. None of the <control-field-value>s match the <control-field-to-match> in a <control-field> and there is not a NOMATCH

4. E-cond-true

   One of the following:

   a. The <boolean-expression> within the <if> is true or there is an <else>
   b. At least one of the <boolean-expression>s within the <select-first> evaluate to true or there is a NOMATCH
   c. At least one of the <control-field-value>s match the <control-field-to-match> in a <control-field> or there is a NOMATCH

5. E-done

   One of the following occurred:

   a. Received a CALL-DONE indication
   b. Received a CONC-DONE indication
   c. Received a PRES-DONE indication
   d. Received a PROC-DONE indication
   e. Received an RM-ENQUEUE-TASK indication
   f. Received an RM-ENQUEUE-RECORD indication
   g. Received an RM-DEQUEUE-RECORD indication
   h. Received an RM-READ-QUEUE-RECORD indication

   i. Received an RM-DONE indication
   j. Received an RR-DONE indication
   k. Received a TX-STATEMENT-DONE indication

6. E-error

   An exception occurred during the execution of a statement. This can occur on internally executed statements or statements that use other state machines if the error occurs in this state machine (for example an index out of range on a parameter to <call-task>).

7. E-exception

   Received one of the following:

   a. CALL-EXCEPTION indication
   b. CONC-EXCEPTION indication
   c. PRES-EXCEPTION indication
   d. PROC-EXCEPTION indication
   e. RM-EXCEPTION indication
   f. RR-EXCEPTION indication
   g. TX-STATEMENT-EXCEPTION indication

8. E-execute

   A TX-STATEMENT-EXECUTE request received.

9. E-exit-block

   Received a TX-STATEMENT-EXIT-BLOCK indication.

10. E-exit-block-c

    Received a TX-STATEMENT-EXIT-BLOCK-COMMIT indication.

11. E-exit-block-r

    Received a TX-STATEMENT-EXIT-BLOCK-ROLLBACK indication.

12. E-exit-task

    Received one of the following:

    a. TX-STATEMENT-EXIT-TASK indication
    b. CONC-EXIT-TASK indication

13. E-exit-task-c

    Received one of the following:

    a. TX-STATEMENT-EXIT-TASK-COMMIT indication
    b. CONC-EXIT-TASK-COMMIT indication

14. E-exit-task-r

    Received one of the following:

    a. TX-STATEMENT-EXIT-TASK-ROLLBACK indication

   b. CONC-EXIT-TASK-ROLLBACK indication

15. E-goto

    A TX-STATEMENT-GOTO indication received.

16. E-goto-c

    A TX-STATEMENT-GOTO-COMMIT indication received.

17. E-goto-r

    A TX-STATEMENT-GOTO-ROLLBACK indication received.

18. E-loop-false

    The <boolean-expression> in a <while> evaluated to false.

19. E-loop-true

    The <boolean-expression> in a <while> evaluated to true.

20. E-work-done

    The internal work of an <assignment>, <audit> or <get-message> statement is done.

21. E-work-exit-block

    The internal work of an <exit-block> statement is done.

22. E-work-exit-task

    The internal work of an <exit-task> statement is done.

23. E-work-goto

    The internal work of a <go-to> statement is done.

24. E-work-raise

    The internal work of a <reraise-exception>, <restart-transaction> or <raise-exception> statement is done.

Actions

1. A-cancel-conc

   Issue a CONC-CANCEL request.

2. A-cancel-work

   Issue one of the following requests depending on the value of V-work-type:

   a. C-call = CALL-CANCEL request

   b. C-presentation-call = PRES-CANCEL request

   c. C-processing-call = PROC-CANCEL request

   d. C-transactional-receive = RR-CANCEL request

   e. C-RM = RM-CANCEL request

3. A-cancel-s

   Issue a TX-STATEMENT-CANCEL request.

4. A-canceled

   Issue a TX-STATEMENT-CANCELED indication.

5. A-conc-block

   Issue CONC-EXECUTE request

6. A-cond-evaluate

   Evaluate the control statement to determine whether or not there is a <statement-list> within the control statement to be executed.

   This evaluation generates one of the following events:

   a. E-cond-true

   b. E-cond-false

   c. E-error

7. A-done

   Issue TX-STATEMENT-DONE indication.

8. A-exception

   Issue TX-STATEMENT-EXCEPTION indication with the parameters from V-current-exception.

9. A-exit-block

   Issue a TX-STATEMENT-EXIT-BLOCK indication.

10. A-exit-block-c

    Issue a TX-STATEMENT-EXIT-BLOCK-COMMIT indication.

11. A-exit-block-r

    Issue a TX-STATEMENT-EXIT-BLOCK-ROLLBACK indication.

12. A-exit-task

    Issue a TX-STATEMENT-EXIT-TASK indication.

13. A-exit-task-c

    Issue a TX-STATEMENT-EXIT-TASK-COMMIT indication.

14. A-exit-task-r

    Issue a TX-STATEMENT-EXIT-TASK-ROLLBACK indication.

15. A-loop-boolean

    Evaluate the <boolean-expression> of the <while> statement.

    This evaluation generates one of the following events:

    a. E-loop-true

    b. E-loop-false

    c. E-error

16. A-goto

    Issue a NONTX-STATEMENT-GOTO indication using the label stored in V-label.

17. A-goto-c

    Issue a NONTX-STATEMENT-GOTO-COMMIT indication using the label stored in V-label.

18. A-goto-r

    Issue a NONTX-STATEMENT-GOTO-ROLLBACK indication using the label stored in V-label.

19. A-set-ind-exc

    Set V-current-exception with the parameters from the indication.

20. A-set-int-exc

    Set V-current-exception with the following information:

    a. Type = as defined for the exception in Chapter 10 on page 167.
    b. Class = as defined for the exception in Chapter 10 on page 167.
    c. Code = implementation-specific
    d. Exception-Code-Group = implementation-specific
    e. Source = System
    f. Level = Current
    g. Exception-Procedure = current task name
    h. Exception-Procedure-Group = current task group name

21. A-set-label

    Save the label from the indication in V-label.

22. A-statements

    Issue TX-STATEMENT-EXECUTE request

23. A-work-external

    Can generate an E-error event if an exception occurs as defined in Chapter 10 on page 167 or do one of the following depending on the statement:

    a. <call-procedure>

       Do the following:

       — Set V-work-type to C-processing-call

       — Issue a PROC-EXECUTE request

    b. <call-task> without <call-task-in> or <call-task-at>

       Do the following:

       — Set V-work-type to C-call

       — Issue a CALL-TASK request with the following parameters:

— Local = true

— RTI-AP-Title = omitted

— Task-Group-UUID = from the server <task-group-specification>

— Task-Group-Version-Major = from the server <task-group-specification>

— Task-Group-Version-Minor = from the server <task-group-specification>

— Task-Operation-Value = the position of the server task's <task-interface> within the server task group's <task-group-specification> as described in Section 18.9.1.1 on page 424.

— Transactional = from the server <task-group-specification>

— Task-Call-Info = the information listed in Section 18.9.1.1 on page 424.

— Arguments = the INPUT and INOUT arguments as defined in the server task <task-group-specification>

c. <call-task> with <call-task-in> and without <call-task-at>

Do the following:

— Set V-work-type to C-call

— Issue a CALL-TASK request with the following parameters:

— Local = false

An implementation can make the local parameter true if the TP-system for the <task-group-identifier> is the same as the caller's TP-system.

— RTI-AP-Title = the RTI-AP-Title from a task group entry in a destination entry. The implementation can choose to either determine the destination entry from the server task group name or use a default destination entry for the TP system. The client TP system chooses the task group entry within the destination by matching the server task group identification and determining whether or not the version number is compatible, as described in Section 3.3.1 on page 47 and Section 17.2.9 on page 306. An implementation can choose to use either the task group name or UUID as the task group identification.

— Task-Group-UUID = from the server <task-group-specification>

— Task-Group-Version-Major = from the server <task-group-specification>

— Task-Group-Version-Minor = from the server <task-group-specification>

— Task-Operation-Value = the position of the server task's <task-interface> within the server task group's <task-group-specification> as described in Section 18.9.1.1 on page 424.

— Transactional = from the server <task-group-specification>

— Task-Call-Info = the information listed in Section 18.9.1.1 on page 424.

— Arguments = the INPUT and INOUT arguments as defined in the server task <task-group-specification>

d. <call-task> with <call-task-in> and with <call-task-at>

Do the following:

— Set V-work-type to C-call

— Issue a CALL-TASK request with the following parameters:

- — Local = false
- — RTI-AP-Title = the RTI-AP-Title in the destination entry for the destination name, as described in Section 17.2.9 on page 306.
- — Task-Group-UUID = from the server <task-group-specification>
- — Task-Group-Version-Major = from the server <task-group-specification>
- — Task-Group-Version-Minor = from the server <task-group-specification>
- — Task-Operation-Value = the position of the server task's <task-interface> within the server task group's <task-group-specification> as described in Section 18.9.1.1 on page 424.
- — Transactional = from the server <task-group-specification>
- — Task-Call-Info = the information listed in Section 18.9.1.1 on page 424.
- — Arguments = the INPUT and INOUT arguments as defined in the server task <task-group-specification>

e. <submit-task> without <submit-in> or <submit-at>

Do the following:

— Set V-work-type to C-RM

— Combine the input arguments for the task with the TASK-CALL-INFORMATION record as described in Section 18.9.1.1 on page 424 to produce the RPC procedure argument list for the server task, and marshal the argument list.

— Issue an RM-ENQUEUE-TASK request with the following parameters:

- — Local-Client = True
- — Local-Server = True
- — Client-RTI-AP-Title = omitted
- — Submission-Information-Record = the information listed in Section 18.9.1.2 on page 426. The DESTINATION-NAME field is implementation-specific, because the server TP system is the same as the submitter TP system and the client TP system for this submitted task request.
- — Marshaled-Arguments = the arguments for the call to the server task. Since the server task executes on the submitter TP system, the value of this parameter is implementation-specific.

f. <submit-task> with <submit-in> and without <submit-at>

Do the following:

— Set V-work-type to C-RM

— Combine the input arguments for the task with the TASK-CALL-INFORMATION record as described in Section 18.9.1.1 on page 424 to produce the RPC procedure

argument list for the server task, and marshal the argument list.

— Issue an RM-ENQUEUE-TASK request with the following parameters:

  — Local-Client = depends on whether the task enqueuer listed in the destination entry for the client TP system for the <task-group-identifier> as described in Section 17.2.9 on page 306 is on the submitter TP system.

  — Local-Server = depends on whether the RTI-AP-Title as listed in the destination entry for the server TP system for the <task-group-identifier> as described in Section 17.2.9 on page 306 is on the submitter TP system.

  — Client-RTI-AP-Title = if Local-Client is false, then the RTI-AP-Title of the task enqueuer at the client TP system as defined for the destination entry for the <task-group-identifier> as described in Section 17.2.9 on page 306. If Local-Client is true, then this parameter is omitted.

  — Submission-Information-Record = the information listed in Section 18.9.1.2 on page 426. Specifically the DESTINATION-NAME field is the <task-group-identifier> if Local-Server is false; otherwise, the value of the DESTINATION-NAME is implementation-specific.

  — Marshaled-Arguments = the marshaled arguments for the server task.

g. <submit-task> with <submit-in> and with <submit-at> using a destination name

   Do the following:

   — Set V-work-type to C-RM

   — Combine the input arguments for the task with the TASK-CALL-INFORMATION record as described in Section 18.9.1.1 on page 424 to produce the RPC procedure argument list for the server task, and marshal the argument list.

   — Issue an RM-ENQUEUE-TASK request with the following parameters:

     — Local-Client = depends on whether the task enqueuer listed in the destination entry for the client TP system for the destination name, as described in Section 17.2.9 on page 306, is on the submitter TP system.

     — Local-Server = depends on whether the RTI-AP-Title as listed in the destination entry for the server TP system for the destination name, as described in Section 17.2.9 on page 306, is on the submitter TP system.

     — Client-RTI-AP-Title = if Local-Client is false, then the RTI-AP-Title of the task enqueuer at the client TP system as defined for the destination entry for the destination name, as described in Section 17.2.9 on page 306. If Local-Client is true, then this parameter is omitted.

     — Submission-Information-Record = the information listed in Section 18.9.1.2 on page 426. Specifically the DESTINATION-NAME field is the <destination-name> if Local-Server is false; otherwise, the value of the DESTINATION-NAME is implementation-specific.

     — Marshaled-Arguments = the marshaled arguments for the server task.

h. <submit-task> with <submit-in> and with <submit-at> using a distribution list

   Do the following:

   — Set V-work-type to C-RM

— Combine the input arguments for the task with the TASK-CALL-INFORMATION record as described in Section 18.9.1.1 on page 424 to produce the RPC procedure argument list for the server task, and marshal the argument list.

— For each destination name in the distribution list, issue an RM-ENQUEUE-TASK request with the following parameters:

— Local-Client = depends on whether the task enqueuer listed in the destination entry for the client TP system for the destination name as described in Section 17.2.9 on page 306 is on the submitter TP system.

— Local-Server = depends on whether the RTI-AP-Title as listed in the destination entry for the server TP system for the destination name as described in Section 17.2.9 on page 306 is on the submitter TP system.

— Client-RTI-AP-Title = if Local-Client is false, then the RTI-AP-Title of the task enqueuer at the client TP system as defined for the destination entry for the destination name as described in Section 17.2.9 on page 306. If Local-Client is true, then this parameter is omitted.

— Submission-Information-Record = the information listed in Section 18.9.1.2 on page 426. Specifically the DESTINATION-NAME field is the <destination-name> if Local-Server is false; otherwise, the value of the DESTINATION-NAME field is implementation-specific.

— Marshaled-Arguments = the arguments for the call to the server task. These arguments must be marshaled as described in Section 18.9.1.1 on page 424.

i. <cancel-submit>

Do the following:

— Set V-work-type to C-RM

— Issue an RM-REMOVE-TASK request with the following parameter:

— Entry-Id = the ID of the entry from the <cancel-submit>

j. <enqueue-record>

Do the following:

— Set V-work-type to C-RM

— Issue an RM-ENQUEUE-RECORD request with the following parameter:

— Data = from the <enqueue-record>

k. <dequeue-record> using <dequeue-first>

Do the following:

— Set V-work-type to C-RM

— Issue an RM-DEQUEUE-RECORD-FIRST request

l. <dequeue-record> using <dequeue-key>

Do the following:

— Set V-work-type to C-RM

— Issue an RM-DEQUEUE-RECORD-KEY request with the following parameter:

— Key = the key specified in <dequeue-key-value>

m. <dequeue-record> using <dequeue-id>

Do the following:

— Set V-work-type to C-RM

— Issue an RM-DEQUEUE-RECORD-ID request with the following parameter:

— Entry-ID = the ID of the data record from <id-workspace-field-1>

n. <read-queue-record> using <read-queue-first>

Do the following:

— Set V-work-type to C-RM

— Issue an RM-READ-QUEUE-RECORD-FIRST request

o. <read-queue-record> using <read-queue-next>

Do the following:

— Set V-work-type to C-RM

— Issue an RM-DEQUEUE-RECORD-NEXT request

p. <read-queue-record> using <read-queue-id>

Do the following:

— Set V-work-type to C-RM

— Issue an RM-READ-QUEUE-RECORD-ID request with the following parameter:

— Entry-ID = the ID of the data record from <id-workspace-field-1>

q. <read-queue-record> using <read-queue-key-first>

Do the following:

— Set V-work-type to C-RM

— Issue an RM-READ-QUEUE-RECORD-KEY request with the following parameter:

— Key = the key specified in <read-queue-key-value>

r. <read-queue-record> using <read-queue-key-next>

Do the following:

— Set V-work-type to C-RM

— Issue an RM-READ-QUEUE-RECORD-KEY-NEXT request

s. <call-presentation-procedure> with transactional work and <call-presentation-receive-list>

Do the following:

— Set V-work-type to C-transactional-receive

— Issue an RR-RECEIVE request

t. <call-presentation-procedure> with transaction work and <call-presentation-send-list>

Do the following:

— Set V-work-type to C-RM

— Issue an RM-TRANSACTIONAL-SEND request with the following parameters:

  — Data = data from the <call-presentation-send-list>

u. <call-presentation-procedure> without transaction work

Do the following:

— Set V-work-type to C-presentation-call

— Issue PRES-EXECUTE request with the following parameter:

  — Data = from any SENDING clause

24. A-work-internal

Depends on the statement:

a. <assignment>, <audit> or <get-message>

Do one of the following:

— If an exception occurs as defined in Chapter 10 on page 167, generate an E-error event.

— Else, generate an E-work-done event.

b. <reraise-exception>

Generate a E-work-raise event.

c. <restart-transaction>

Do one of the following:

— If an exception occurs as defined in Chapter 10 on page 167, generate an E-error event.

— Else, set V-current-exception with the information from <exception-information>, setting type to transient transaction, and generate a E-work-raise event.

d. <raise-exception>

Do one of the following:

— If an exception occurs as defined in Chapter 10 on page 167, generate an E-error event

— Else, set V-current-exception with the information from <exception-information> setting the type to permanent transaction if ROLLBACK is specified and generate a E-work-raise event.

e. <exit-block>

Do the following:

— Do one of the following:

  — If no <transaction-control> is specified, set V-tx-control to C-tx-none.

  — If the <transaction-control> specifies COMMIT, set V-tx-control to C-tx-commit.

— If the <transaction-control> specifies ROLLBACK, set V-tx-control to C-tx-rollback

— Generate an E-work-exit-block event.

f. <exit-task>

Do the following:

a. Do one of the following:

a. If no <transaction-control> is specified and the task is composable, set V-tx-control to C-tx-none.

b. If no <transaction-control> is specified and the task is noncomposable, set V-tx-control to C-tx-commit.

c. If the <transaction-control> specifies COMMIT, set V-tx-control to C-tx-commit.

d. If the <transaction-control> specifies ROLLBACK, set V-tx-control to C-tx-rollback

b. Generate an E-work-exit-task event.

g. <go-to>

Do the following:

a. Set the label in V-label.

b. Do one of the following:

a. If no <transaction-control> is specified, set V-tx-control to C-tx-none.

b. If the <transaction-control> specifies COMMIT, set V-tx-control to C-tx-commit.

c. If the <transaction-control> specifies ROLLBACK, set V-tx-control to C-tx-rollback

c. Generate a E-work-goto event

States

1. S-null

   The initial state.

2. S-block

   Waiting for the <statement-block> statement to complete.

3. S-conc-block

   Waiting for the <concurrent-block> statement to complete.

4. S-handler

   Waiting for the <exception-handler> on a block statement to complete.

5. S-cond-evaluate

   Waiting for the evaluation of the <if>, <control-field> or <select-first> Boolean or matching to complete.

6. S-cond-execute

   Waiting for the <statement-list> within the <if>, <control-field> or <select-first> to complete.

7. S-loop-boolean

   Waiting for the <while> Boolean to complete.

8. S-loop-execute

   Waiting for the <statement-list> in the <while> to complete.

9. S-work-internal

   Waiting for the <assignment>, <audit>, <get-message>, <reraise-exception>, <restart-transaction>, <raise-exception>, <exit-block>, <exit-task> or <go-to> statement to complete.

10. S-work-external

    Waiting for the <call-procedure>, <call-task>, <submit-task>, <cancel-submit>, <enqueue-record>, <dequeue-record>, <read-queue-record> or <call-presentation-procedure> statement to complete.

11. S-cancel

    Waiting for the current <statement> to cancel.

State Transition Tables

	S-null
E-execute+ ¬P-more	S-null A-done
E-execute+ P-block	S-block A-statements
E-execute+ P-conc-block	S-conc-block A-conc-block
E-execute+ P-conditional	S-cond-evaluate A-cond-evaluate
E-execute+ P-loop	S-loop-boolean A-loop-boolean
E-execute+ P-work-internal	S-work-internal A-work-internal
E-execute+ P-work-external	S-work-external A-work-external

Table 18-24 Transactional Statement List (Null State)

	S-block
E-done+¬P-more or E-exit-block+¬P-more	S-null A-done
E-done+P-block or E-goto+P-label+P-block or E-exit-block+P-block	S-block A-statements
E-done+P-conc-block or E-goto+P-label+P-conc-block or E-exit-block+P-conc-block	S-conc-block A-conc-block
E-done+P-conditional or E-goto+P-label+P-conditional or E-exit-block+P-conditional	S-cond-evaluate A-cond-evaluate
E-done+P-loop or E-goto+P-label+P-loop or E-exit-block+P-loop	S-loop-boolean A-loop-boolean
E-done+P-work-internal or E-goto+P-label+P-work-internal or E-exit-block+P-work-internal	S-work-internal A-work-internal
E-done+P-work-external or E-goto+P-label+P-work-external or E-exit-block+P-work-external	S-work-external A-work-external
E-goto+¬P-label	S-null A-set-label, A-goto
E-goto-c	S-null A-set-label, A-goto-c
E-goto-r	S-null A-set-label, A-goto-r
E-exit-task	S-null A-exit-task
E-exit-task-c	S-null A-exit-task-c
E-exit-task-r	S-null A-exit-task-r
E-exception+P-non-trans+P-hand	S-handler A-set-ind-exc, A-statements
E-exception+P-non-trans+ ¬P-hand	S-null A-set-ind-exc, A-exception
E-exception+¬P-non-trans	S-null A-set-ind-exc, A-exception
E-cancel	S-cancel A-cancel-s

Table 18-25 Transactional Statement List (Block State)

	S-conc-block
E-done+¬P-more	S-null A-done
E-done+P-block	S-block A-statements
E-done+P-conc-block	S-conc-block A-conc-block
E-done+P-conditional	S-cond-evaluate A-cond-evaluate
E-done+P-loop	S-loop-boolean A-loop-boolean
E-done+P-work-internal	S-work-internal A-work-internal
E-done+P-work-external	S-work-external A-work-external
E-exit-task	S-null A-exit-task
E-exit-task-c	S-null A-exit-task-c
E-exit-task-r	S-null A-exit-task-r
E-exception+ P-non-trans+P-hand	S-handler A-set-ind-exc, A-statements
E-exception+ P-non-trans+¬P-hand	S-null A-set-ind-exc, A-exception
E-exception+ ¬P-non-trans	S-null A-set-ind-exc, A-exception
E-cancel	S-cancel A-cancel-conc

Table 18-26 Transactional Statement List (Concurrent Block State)

	S-handler
E-done+¬P-more	S-null A-done
E-done+P-block or E-goto+P-label+P-block	S-block A-statements
E-done+P-conc-block or E-goto+P-label+P-conc-block	S-conc-block A-conc-block
E-done+P-conditional or E-goto+P-label+P-conditional	S-cond-evaluate A-cond-evaluate
E-done+P-loop or E-goto+P-label+P-loop	S-loop-boolean A-loop-boolean
E-done+P-work-internal or E-goto+P-label+P-work-internal	S-work-internal A-work-internal
E-done+P-work-external or E-goto+P-label+P-work-external	S-work-external A-work-external
E-goto+¬P-label	S-null A-set-label, A-goto
E-goto-c	S-null A-set-label, A-goto-c
E-goto-r	S-null A-set-label, A-goto-r
E-exit-block	S-null A-exit-block
E-exit-block-c	S-null A-exit-block-c
E-exit-block-r	S-null A-exit-block-r
E-exit-task	S-null A-exit-task
E-exit-task-c	S-null A-exit-task-c
E-exit-task-r	S-null A-exit-task-r
E-exception	S-null A-set-ind-exc, A-exception
E-cancel	S-cancel A-cancel-s

Table 18-27 Transactional Statement List (Exception Handler State)

	S-cond-evaluate
E-cond-true	S-cond-execute A-statements
E-cond-false+ ¬P-more	S-null A-done
E-cond-false+ P-block	S-block A-statements
E-cond-false+ P-conc-block	S-conc-block A-conc-block
E-cond-false+ P-conditional	S-cond-evaluate A-cond-evaluate
E-cond-false+ P-loop	S-loop-boolean A-loop-boolean
E-cond-false+ P-work-internal	S-work-internal A-work-internal
E-cond-false+ P-work-external	S-work-external A-work-external
E-error	S-null A-set-int-exc, A-exception
E-cancel	S-null A-canceled

Table 18-28 Transactional Statement List (Conditional Evaluation State)

	S-cond-execute
E-done+¬P-more	S-null A-done
E-done+P-block or E-goto+P-label+P-block	S-block A-statements
E-done+P-conc-block or E-goto+P-label+P-conc-block	S-conc-block A-conc-block
E-done+P-conditional or E-goto+P-label+P-conditional	S-cond-evaluate A-cond-evaluate
E-done+P-loop or E-goto+P-label+P-loop	S-loop-boolean A-loop-boolean
E-done+P-work-internal or E-goto+P-label+P-work-internal	S-work-internal A-work-internal
E-done+P-work-external or E-goto+P-label+P-work-external	S-work-external A-work-external
E-goto+¬P-label	S-null A-set-label, A-goto
E-goto-c	S-null A-set-label, A-goto-c
E-goto-r	S-null A-set-label, A-goto-r
E-exit-block	S-null A-exit-block
E-exit-block-c	S-null A-exit-block-c
E-exit-block-r	S-null A-exit-block-r
E-exit-task	S-null A-exit-task
E-exit-task-c	S-null A-exit-task-c
E-exit-task-r	S-null A-exit-task-r
E-exception	S-null A-set-ind-exc, A-exception
E-cancel	S-cancel A-cancel-s

Table 18-29 Transactional Statement List (Conditional Execute State)

	S-loop-boolean
E-loop-true	S-loop-execute A-statements
E-loop-false+ ¬P-more	S-null A-done
E-loop-false+ P-block	S-block A-statements
E-loop-false+ P-conc-block	S-conc-block A-conc-block
E-loop-false+ P-conditional	S-cond-evaluate A-cond-evaluate
E-loop-false+ P-loop	S-loop-boolean A-loop-boolean
E-loop-false+ P-work-internal	S-work-internal A-work-internal
E-loop-false+ P-work-external	S-work-external A-work-external
E-error	S-null A-set-int-exc, A-exception
E-cancel	S-null A-canceled

Table 18-30 Transactional Statement List (Loop Boolean State)

	S-loop-execute
E-done	S-loop-boolean A-loop-boolean
E-goto+P-label+ P-block	S-block A-statements
E-goto+P-label+ P-conc-block	S-conc-block A-conc-block
E-goto+P-label+ P-conditional	S-cond-evaluate A-cond-evaluate
E-goto+P-label+ P-loop	S-loop-boolean A-loop-boolean
E-goto+P-label+ P-work-internal	S-work-internal A-work-internal
E-goto+P-label+ P-work-external	S-work-external A-work-external
E-goto+¬P-label	S-null A-set-label, A-goto
E-goto-c	S-null A-set-label, A-goto-c
E-goto-r	S-null A-set-label, A-goto-r
E-exit-block	S-null A-exit-block
E-exit-block-c	S-null A-exit-block-c
E-exit-block-r	S-null A-exit-block-r
E-exit-task	S-null A-exit-task
E-exit-task-c	S-null A-exit-task-c
E-exit-task-r	S-null A-exit-task-r
E-exception	S-null A-set-ind-exc, A-exception
E-cancel	S-cancel A-cancel-s

Table 18-31 Transactional Statement List (Loop Execute State)

	S-work-internal
E-work-done+¬P-more	S-null A-done
E-work-done+P-block or E-work-goto+P-label+P-block	S-block A-statements
E-work-done+P-conc-block or E-work-goto+P-label+P-conc-block	S-conc-block A-conc-block
E-work-done+P-conditional or E-work-goto+P-label+P-conditional	S-cond-evaluate A-cond-evaluate
E-work-done+P-loop or E-work-goto+P-label+P-loop	S-loop-boolean A-loop-boolean
E-work-done+P-work-internal or E-work-goto+P-label+P-work-internal	S-work-internal A-work-internal
E-work-done+P-work-external or E-work-goto+P-label+P-work-external	S-work-external A-work-external
E-work-goto+¬P-label	S-null A-goto
E-work-raise	S-null A-exception
E-work-exit-block+P-tx-none	S-null A-exit-block
E-work-exit-block+P-tx-commit	S-null A-exit-block-c
E-work-exit-block+P-tx-rollback	S-null A-exit-block-r
E-work-exit-task+P-tx-none	S-null A-exit-task
E-work-exit-task+P-tx-commit	S-null A-exit-task-c
E-work-exit-task+P-tx-rollback	S-null A-exit-task-r
E-error	S-null A-set-int-exc, A-exception
E-cancel	S-null A-canceled

Table 18-32 Transactional Statement List (Internal Work State)

	S-work-external
E-done +¬P-more	S-null A-done
E-done+P-block	S-block A-statements
E-done+P-conc-block	S-conc-block A-conc-block
E-done+P-conditional	S-cond-evaluate A-cond-evaluate
E-done+P-loop	S-loop-boolean A-loop-boolean
E-done+P-work-internal	S-work-internal A-work-internal
E-done+P-work-external	S-work-external A-work-external
E-error	S-null A-set-int-exc, A-exception
E-exception	S-null A-set-ind-exc, A-exception
E-cancel	S-cancel A-cancel-work

Table 18-33 Transactional Statement List (External Work State)

	S-cancel
E-exception+ P-fatal	S-null A-set-ind-exc, A-exception
E-canceled	S-null A-canceled

Table 18-34 Transactional Statement List (Cancel State)

18.7.4 Concurrent Block State Machine

The Concurrent Block state machine models the processing done to execute a <concurrent-block>.

Multiple Concurrent Block state machines can exist within an execution context. Each CONC-EXECUTE request creates a separate Concurrent Block state machine instance.

The Concurrent Block state machine models a <concurrent-block> by issuing indications which create a Concurrent Thread state machine instance for each concurrent <statement> in a <concurrent-block>. Since the Concurrent Thread state machine instances are created using indications, they execute in new thread contexts. The thread context in which the Concurrent Block state machine instance executes is called the parent thread. The new thread contexts are called child threads.

Service Primitives Provided

The service primitives for the Concurrent Block state machine are divided into two sets: those used by the parent thread and those used by the child threads.

1. CONC-EXECUTE — request, parent thread

   Executes a concurrent block.

   This request has no parameters.

2. CONC-CANCEL — request, parent thread

   Cancels the execution of a concurrent block.

   This request has no parameters.

3. CONC-CANCELED — indication, parent thread

   Reports that the concurrent block has been cancelled.

   This indication has no parameters.

4. CONC-DONE — indication, parent thread

   Reports that the concurrent block execution finished with default flow control.

   This indication has no parameters.

5. CONC-EXIT-TASK — indication, parent thread

   Reports that a thread executed an <exit-task> without transaction control.

   This indication has no parameters.

6. CONC-EXIT-TASK-COMMIT — indication, parent thread

   Reports that a thread executed an <exit-task> with commit.

   This indication has no parameters.

7. CONC-EXIT-TASK-ROLLBACK — indication, parent thread

   Reports that a thread executed an <exit-task> with rollback.

   This indication has no parameters.

8. CONC-EXCEPTION — indication, parent thread

   Reports that the concurrent block execution terminated with an exception.

This indication has the following parameters:

a. Type = Non-transaction, Transient Transaction, Permanent Transaction or Fatal Transaction

b. Class = one of the classes defined in Appendix C on page 489.

c. Code = the exception code for the exception as defined in Section 3.7.2 on page 62.

d. Exception-Code-Group = the exception code group for the exception as defined in Section 3.7.2 on page 62.

e. Source = System or Application

f. Level = Current or Propagated

g. Exception-Procedure = the procedure in which the exception occurred as defined in Section 3.7.2 on page 62.

h. Exception-Procedure-Group = the procedure group in which the exception occurred as defined in Section 3.7.2 on page 62.

9. CONC-THREAD-EXECUTE — indication, child thread

   Indicates that a new child thread is needed to execute a concurrent statement.

   This indication has no parameters.

10. CONC-THREAD-EXIT-TASK — request, child thread

    Reports that a child thread executed an <exit-task> without transaction control.

    This request has no parameters

11. CONC-THREAD-EXIT-TASK-COMMIT — request, child thread

    Reports that a child thread executed an <exit-task> with commit.

    This request has no parameters

12. CONC-THREAD-EXIT-TASK-ROLLBACK — request, child thread

    Reports that a child thread executed an <exit-task> with rollback.

    This request has no parameters.

13. CONC-THREAD-CANCEL — indication, child thread

    Reports that a the child thread is to be cancelled.

    This indication has no parameters.

14. CONC-THREAD-CANCELED — request, child thread

    Reports that the child thread has been cancelled.

    This indication has no parameters.

15. CONC-THREAD-DONE — request, child thread

    Reports that a child thread has finished execution with default flow control.

    This indication has no parameters.

16. CONC-THREAD-EXCEPTION — request, child thread

    Reports that a child thread terminated with an exception.

This indication has the following parameters:

a. Type = Non-transaction, Transient Transaction, Permanent Transaction or Fatal Transaction

b. Class = one of the classes defined in Appendix C on page 489.

c. Code = the exception code for the exception as defined in Section 3.7.2 on page 62.

d. Exception-Code-Group = the exception code group for the exception as defined in Section 3.7.2 on page 62.

e. Source = System or Application

f. Level = Current or Propagated

g. Exception-Procedure = the procedure in which the exception occurred as defined in Section 3.7.2 on page 62.

h. Exception-Procedure-Group = the procedure group in which the exception occurred as defined in Section 3.7.2 on page 62.

Service User

The service user of the Concurrent Block state machine is different for the parent thread services and the child thread services.

1. For the parent thread services:

   The service user is the state machine instance that issued the CONC-EXECUTE request. This service user receives all parent thread indications generated by a Concurrent Block state machine instance and the CONC-CANCEL request from that service user is directed to the Concurrent Block state machine instance that was created for that service user.

2. For the child thread services:

   The service users are the new instances of the Concurrent Thread state machine created by the CONC-THREAD-EXECUTE indications generated by a Concurrent Block state machine instance. When a Concurrent Block state machine issues a CONC-THREAD-CANCEL indication, it is sent to all Concurrent Thread state machine instances created by that Concurrent Block state machine instance.

State Machines Used

None.

Variables

1. V-exit-type

   Contains the type of exit task request received. This can take one of the following values:

   a. C-exit-task

      <exit-task> executed without <transaction-control>.

   b. C-exit-task-commit

      <exit-task> executed with COMMIT.

   c. C-exit-task-rollback

<exit-task> executed with ROLLBACK.

This variable stores data.

2. V-saved-exception

   Contains the Type, Class, Code, Exception-Code-Group, Source, Level, Exception-Procedure, and Exception-Procedure-Group of a saved exception.

   This variable stores data.

Predicates

1. P-all

   Is true if all the child threads have terminated.

2. P-fatal

   Is true if the Type parameter of the CONC-THREAD-EXCEPTION request contains Fatal.

3. P-non-txn

   Is true if the Type parameter of the CONC-THREAD-EXCEPTION request contains Non-transaction.

4. P-trans

   Is true if the Type parameter of the CONC-THREAD-EXCEPTION request contains Permanent Transaction or Transient Transaction.

Events

1. E-cancel

   A CONC-CANCEL request received.

2. E-canceled

   A CONC-THREAD-CANCELED request received.

3. E-done

   A CONC-THREAD-DONE request received.

4. E-execute

   A CONC-EXECUTE request received.

5. E-exception

   A CONC-THREAD-EXCEPTION request received.

6. E-exit

   A CONC-THREAD-EXIT-TASK request received.

7. E-exit-c

   A CONC-THREAD-EXIT-TASK-COMMIT request received.

8. E-exit-r

   A CONC-THREAD-EXIT-TASK-ROLLBACK request received.

Actions

1. A-cancel

   Issue a CONC-THREAD-CANCEL indication to each child thread that has not terminated.

2. A-canceled

   Issue a CONC-CANCELED indication.

3. A-cancel-o

   An implementation can issue a CONC-THREAD-CANCEL indication to each child thread that has not terminated. This is not required.

4. A-cancel-p

   Issue a CONC-THREAD-CANCEL indication to each child thread that has not terminated if an implementation did not implement A-cancel-o.

5. A-done

   Issue a CONC-DONE indication.

6. A-exception-r

   Issue a CONC-EXCEPTION indication with the exception information from the CONC-THREAD-EXCEPTION request.

7. A-exception-s

   Issue a CONC-EXCEPTION indication with the exception information from V-saved-exception.

8. A-exit

   Depending on the contents of V-exit-type, issue a CONC-EXIT-TASK, CONC-EXIT-TASK-COMMIT or CONC-EXIT-TASK-ROLLBACK indication.

9. A-save-exc

   Save the parameters from the CONC-THREAD-EXCEPTION request in V-saved-exception.

10. A-save-exit

    Save in V-exit-type whether a CONC-THREAD-EXIT, CONC-THREAD-EXIT-COMMIT or CONC-THREAD-EXIT-ROLLBACK request was received.

11. A-threads

    Issue a CONC-THREAD-EXECUTE indication for each concurrent <statement> in the <concurrent-block>.

States

1. S-null

   The initial state.

2. S-execute

   Waiting until all threads finish executing with default flow control.

3. S-exit-c

Waiting until all threads finish executing after a thread completes with an <exit-task> that did not specify ROLLBACK.

4. S-exc-n

   Waiting until all threads finish after a thread generates a non-transaction exception.

5. S-exit-r

   Waiting until all threads finish after a thread completes with an <exit-task> flow and that <exit-task> specified ROLLBACK.

6. S-exc-t

   Waiting until all threads finish after a thread generates a transient transaction or permanent transaction exception.

7. S-cancel

   Waiting until all threads finish after receiving a cancel.

8. S-exc-f

   Waiting until all threads finish after a thread generates a fatal transaction exception.

State Transition Tables

	S-null
E-execute	S-execute A-threads

Table 18-35 Concurrent Block (Null State)

	S-execute	**S-exit-c**	**S-exc-n**	**S-exit-r**
E-done+ P-all	S-null A-done	S-null A-exit	S-null A-exception-s	S-null A-exit
E-done+ ¬P-all	S-execute	S-exit-c	S-exc-n	S-exit-r
E-exit+P-all or E-exit-c+P-all	S-null A-save-exit A-exit	S-null A-exit	S-null A-exception-s	S-null A-exit
E-exit+¬P-all or E-exit-c+¬P-all	S-exit-c A-save-exit	S-exit-c	S-exc-n	S-exit-r
E-exception+P-all+ P-non-txn	S-null A-exception-r	S-null A-exception-r	S-null A-exception-s	S-null A-exit
E-exception+¬P-all+ P-non-txn	S-exc-n A-save-exc	S-exc-n A-save-exc	S-exc-n	S-exit-r
E-exit-r+P-all	S-null A-save-exit A-exit	S-null A-save-exit A-exit	S-null A-save-exit A-exit	S-null A-exit
E-exit-r+¬P-all	S-exit-r A-save-exit	S-exit-r A-save-exit	S-exit-r A-save-exit	S-exit-r
E-exception+P-all+ P-trans	S-null A-exception-r	S-null A-exception-r	S-null A-exception-r	S-null A-exception-r
E-exception+¬P-all+ P-trans	S-exc-t A-save-exc A-cancel-o	S-exc-t A-save-exc A-cancel-o	S-exc-t A-save-exc A-cancel-o	S-exc-t A-save-exc A-cancel-o
E-cancel	S-cancel A-cancel	S-cancel A-cancel	S-cancel A-cancel	S-cancel A-cancel
E-exception+P-all+ P-fatal	S-null A-exception-r	S-null A-exception-r	S-null A-exception-r	S-null A-exception-r
E-exception+¬P-all+ P-fatal	S-exc-f A-save-exc A-cancel-o	S-exc-f A-save-exc A-cancel-o	S-exc-f A-save-exc A-cancel-o	S-exc-f A-save-exc A-cancel-o

Table 18-36 Concurrent Block (Execute, Exit and Non-transaction Exception States)

	S-exc-t	**S-cancel**	**S-exc-f**
E-done+ P-all	S-null A-exception-s		S-null A-exception-s
E-done+ ¬P-all	S-exc-t		S-exc-f
E-exit+P-all or E-exit-c+P-all	S-null A-exception-s		S-null A-exception-s
E-exit+¬P-all or E-exit-c+¬P-all	S-exc-t		S-exc-f
E-exception+P-all+ P-non-txn	S-null A-exception-s		S-null A-exception-s
E-exception+¬P-all+ P-non-txn	S-exc-t		S-exc-f
E-exit-r+P-all	S-null A-exception-s		S-null A-exception-s
E-exit-r+¬P-all	S-exc-t		S-exc-f
E-exception+P-all+ P-trans	S-null A-exception-s		S-null A-exception-s
E-exception+¬P-all+ P-trans	S-exc-t		S-exc-f
E-cancel	S-cancel A-cancel-p		S-exc-f A-cancel-p
E-canceled+ P-all	S-null A-exception-s	S-null A-canceled	S-null A-exception-s
E-canceled+ ¬P-all	S-exc-t	S-cancel	S-exc-f
E-exception+P-all+ P-fatal	S-null A-exception-r	S-null A-exception-r	S-null A-exception-s
E-exception+¬P-all+ P-fatal	S-exc-f A-save-exc	S-exc-f A-save-exc	S-exc-f

Table 18-37 Concurrent Block (Transaction Exception and Cancel States)

18.7.5 Concurrent Thread State Machine

The Concurrent Thread state machine models the processing done to execute a <statement> in a concurrent block.

Only one Concurrent Thread state machine can exist within a thread context. Each CONC-EXECUTE-THREAD indication creates a separate Concurrent Thread state machine instance. Since each Concurrent Thread state machine instance is created by an indication, each Concurrent Thread state machine instance creation also creates a thread context.

Service Primitives Provided

None.

Service User

None.

State Machines Used

1. Concurrent Block (CONC-)
2. Transaction Statement List (TX-STATEMENT-)

Variables

None.

Predicates

1. P-fatal

   Is true if the Type parameter of the TX-STATEMENT-EXCEPTION indication is Fatal.

Events

1. E-cancel

   A CONC-THREAD-CANCEL indication received.
2. E-canceled

   A TX-STATEMENT-CANCELED indication received.
3. E-done

   A TX-STATEMENT-DONE indication received.
4. E-exception

   A TX-STATEMENT-EXCEPTION indication received.
5. E-execute

   A CONC-THREAD-EXECUTE indication received.
6. E-exit-block

   A TX-STATEMENT-EXIT-BLOCK indication received.
7. E-exit-task

   A TX-STATEMENT-EXIT-TASK indication received.

8. E-exit-task-c

   A TX-STATEMENT-EXIT-TASK-COMMIT indication received.

9. E-exit-task-r

   A TX-STATEMENT-EXIT-TASK-ROLLBACK indication received.

Actions

1. A-cancel

   Issue a TX-STATEMENT-CANCEL request.

2. A-canceled

   Issue a CONC-THREAD-CANCELED request.

3. A-done

   Issue a CONC-THREAD-DONE request.

4. A-exc

   Issue a CONC-THREAD-EXCEPTION request with the exception information from the TX-STATEMENT-EXCEPTION indication.

5. A-execute

   Issue a TX-STATEMENT-EXECUTE request.

6. A-exit-task

   Issue a CONC-THREAD-EXIT-TASK request.

7. A-exit-task-c

   Issue a CONC-THREAD-EXIT-TASK-COMMIT request.

8. A-exit-task-r

   Issue a CONC-THREAD-EXIT-TASK-ROLLBACK request.

States

1. S-null

   The initial state.

2. S-execute

   Waiting for the statement sequence to finish executing.

3. S-cancel

   Waiting for the statement sequence to cancel.

State Transition Tables

	S-null	**S-execute**	**S-cancel**
E-execute	S-execute A-execute		
E-cancel		S-cancel A-cancel	
E-done or E-exit-block		S-null A-done	
E-exit-task		S-null A-exit-task	
E-exit-task-c		S-null A-exit-task-c	
E-exit-task-r		S-null A-exit-task-r	
E-exception+ P-fatal		S-null A-exc	S-null A-exc
E-exception+ ¬P-fatal		S-null A-exc	
E-canceled			S-null A-canceled

Table 18-38 Concurrent Thread

18.8 Submitted Task Operations State Machines

18.8.1 Task Dequeuer State Machine

The Task Dequeuer state machine models the processing done to dequeue submitted task requests stored on a client TP system and to call those tasks.

No more than one Task Dequeuer state machine instance can exist within an execution context. Each DEQUEUE-START request creates a separate Task Dequeuer state machine instance.

Service Primitives Provided

The Task Dequeuer state machine provides the following service primitives:

1. DEQUEUE-START — request

   Starts processing submitted task requests. This request is made using an implementation-specific system management function.

   This request has no parameters.

2. DEQUEUE-STOP — request

   Stops processing submitted task requests. This request is made using an implementation-specific system management function.

   This request has no parameters.

Service User

The service user of the Task Dequeuer state machine is the TP system management entity. The modelling of the TP system management entity is outside the scope of this chapter.

State Machines Used

1. Call (CALL-)
2. Resource Manager (RM-)
3. Transaction (TRANS-)

Variables

1. V-local-server

   The value of the Local-Server parameter of the last RM-DEQUEUE-TASK indication.

   This is a Boolean variable.

2. V-submission-record

   The value of the Submission-Record parameter of the last RM-DEQUEUE-TASK indication.

   This variable stores data.

3. V-marshaled-arguments

   The value of the Marshaled-Arguments parameter of the last RM-DEQUEUE-TASK indication.

   This variable stores data.

Predicates

1. P-rec-exc

   Is true if the exception is one of the following classes:

   a. ENV-INVOCATION-ERROR

   b. ENV-EXECUTION-ERROR

   c. TXN-FAILURE-ERROR

   d. TXN-TIMEOUT-ERROR

   e. TXN-INCOMPLETE-ERROR

   f. REQUEST-TIMEOUT-ERROR

2. P-repeat

   Is true if the REPEAT-TIME field of the V-submission-record is not null.

3. P-retry

   Is true if the retry count for the submitted task entry is not at the maximum. See Section 17.2.18 on page 315 for more information about retry counts.

Events

1. E-commit-done

   Received a TRANS-COMMIT-DONE indication

2. E-done

   A CALL-DONE indication received.

3. E-element

   An RM-DEQUEUE-TASK indication received.

4. E-exception

   A CALL-EXCEPTION indication received.

5. E-rollb-done

   An TRANS-ROLLBACK-DONE indication received.

6. E-rollb-rec

   A TRANS-ROLLBACK-RECEIVED indication received.

7. E-start

   A DEQUEUE-START request received.

8. E-stop

   A DEQUEUE-STOP request received.

Actions

1. A-audit-failure

   Do the following:

   a. Audit the failure

   b. Remove the task queue entry for the submitted task

2. A-call-task

   Do the following:

   a. Save the following from the RM-DEQUEUE-TASK indication:

      — Local-Server parameter in V-local-server

      — Submission-Information-Record parameter in V-submission-record

      — Marshaled-Arguments parameter in V-marshaled-arguments

   b. Issue a CALL-TASK-MARSHALED request with the following parameters:

      — Local = from the Local-Server parameter of the RM-DEQUEUE-TASK indication.

      — RTI-AP-Title = omitted if Local is true, else from the destination entry for the DESTINATION-NAME in V-submission-record as defined in Section 17.2.9 on page 306.

      — Task-Group-UUID = from the TASK-GROUP-UUID field of V-submission-record.

      — Task-Group-Version-Major = from the VERSION-MAJOR field of V-submission-record.

      — Task-Group-Version-Minor = from the VERSION-MINOR field of V-submission-record.

      — Task-Operational-Value = from the TASK-OPERATION-ID field of V-submission-record.

      — Transactional = from the COMPOSABLE-FLAG of V-submission-record.

      — Marshalled-Arguments = from the Marshaled-Arguments parameter of V-submission-record.

3. A-cleanup

   Issue a CALL-RELEASE-CONTEXT request to each Call state machine instance with V-client true in the execution context (optional).

4. A-commit

   Issue a TRANS-COMMIT request.

5. A-dequeue

   Do the following:

   a. Issue a TRANS-START request with the following parameter:

      — Transaction type = Root

   b. Issue an RM-DEQUEUE-TASK request with the following parameters:

      — Submission client = True

6. A-repeat

   Issue an RM-ENQUEUE-TASK request with the following parameters:

   a. Local-Client = true

   b. Local-Server = from V-local-server

   c. Client-RTI-AP-Title = omitted

   d. Submission-Information-Record = from V-submission-record with the following changes:

      — TRIGGER-TYPE set to 2 or 3 (implementation-specific)

      — TRIGGER-TIME field set to be the current time plus the value of REPEAT-TIME

   e. Marshaled-Arguments = from V-marshaled-arguments

7. A-requeue

   Do the following:

   a. Increment the retry count maintained for the submitted task entry. See Section 17.2.18.1 on page 316 for more information on retry counts.

   b. Change the trigger on the submitted task entry so that some time passes until the task call is retried (optional).

8. A-rollback

   Issue a TRANS-ROLLBACK request.

States

1. S-null

   The initial state.

2. S-deque-w

   Waiting for the dequeue operation to complete.

3. S-task-w

   Waiting for task execution to complete.

4. S-commit

   Waiting for normal commit.

5. S-rollb-r

   Waiting for transaction rollback before retrying the task call.

6. S-rollb-n

   Waiting for transaction rollback when task call will not be retried.

7. S-shutdwn

   Waiting for transaction rollback before shutting down.

State Transition Tables

	S-null	**S-deque-w**	**S-task-w**
E-start	S-deque-w A-dequeue		
E-element		S-task-w A-call-task	
E-done+ P-repeat			S-commit A-repeat, A-commit
E-done+ ¬P-repeat			S-commit A-commit
E-exception+ P-retry+P-rec-exc			S-rollb-r A-rollback
E-exception+ (¬P-retry or¬P-rec-exc)			S-rollb-n A-rollback
E-rollb-rec+ P-retry			S-rollb-r A-rollback
E-rollb-rec+ ¬P-retry			S-rollb-n A-rollback
E-stop		S-shutdwn A-rollback	

Table 18-39 Task Dequeuer (Null, Dequeue and Execute States)

	S-commit	**S-rollb-r**	**S-rollb-n**	**S-shutdwn**
E-commit-done	S-deque-w A-dequeue			
E-rollb-done+ P-retry	S-deque-w A-requeue A-dequeue	S-deque-w A-requeue A-dequeue	S-deque-w A-audit-failure A-dequeue	S-null A-cleanup
E-rollb-done+ ¬P-retry	S-deque-w A-audit-failure A-dequeue		S-deque-w A-audit-failure A-dequeue	S-null A-cleanup

Table 18-40 Task Dequeuer (Commit, Rollback and Shutdown States)

18.8.2 Task Forwarder State Machine

The Task Forwarder state machine models the processing done to dequeue submitted task requests stored on a submitter TP system and forward those requests to a client TP system.

No more than one Task Forwarder state machine instance can exist within an execution context. Each FORWARD-START request creates a separate Task Forwarder state machine instance.

Service Primitives Provided

The Task Forwarder state machine provides the following service primitives:

1. FORWARD-START — request

   Starts processing submitted task requests. This request is made using an implementation-specific system management function.

   This request has no parameters.

2. FORWARD-STOP — request

   Stops processing submitted task requests. This request is made using an implementation-specific system management function.

   This request has no parameters.

Service User

The service user of the Task Forwarder state machine is the TP system management entity. The modelling of the TP system management entity is outside the scope of this chapter.

State Machines Used

1. Call (CALL-)
2. Resource Manager (RM-)
3. Transaction (TRANS-)

Variables

None.

Predicates

1. P-rec-exc

   Is true if the exception is one of the following classes:

   a. ENV-INVOCATION-ERROR
   b. ENV-EXECUTION-ERROR
   c. TXN-FAILURE-ERROR
   d. TXN-TIMEOUT-ERROR

2. P-retry

   Is true if the retry count for the submitted task entry is not at the maximum. See Section 17.2.18 on page 315 for more information about retry counts.

Events

1. E-commit-done

   A TRANS-COMMIT-DONE indication received.

2. E-done

   A CALL-DONE indication received.

3. E-element

   An RM-DEQUEUE-TASK indication received.

4. E-exception

   A CALL-EXCEPTION indication received.

5. E-rollb-done

   A TRANS-ROLLBACK-DONE indication received.

6. E-rollb-rec

   A TRANS-ROLLBACK-RECEIVED indication received.

7. E-start

   An FORWARD-START request received.

8. E-stop

   An FORWARD-STOP request received.

Actions

1. A-audit-failure

   Do the following:

   a. Audit the failure

   b. Remove the task queue entry for the submitted task

2. A-cleanup

   Issue CALL-RELEASE-CONTEXT request to each Call state machine instance with V-client true in the execution context (optional).

3. A-commit

   Issue a TRANS-COMMIT request.

4. A-dequeue

   Do the following:

   a. Issue a TRANS-START request with the following parameter:

      — Transaction type = Root

   b. Issue an RM-DEQUEUE-TASK request with the following parameters:

      — Submission client = False

5. A-forward

   Issue a CALL-FORWARD-TASK request with the following parameters:

   a. RTI-AP-Title = the Client-RTI-AP-Title parameter of the RM-DEQUEUE-TASK indication.

   b. Transactional = from the Transactional Forwarding attribute for the client TP system as set in the Destination attribute in the environmental execution information as discussed in Section 3.3.2 on page 50 and Section 17.2.9 on page 306.

   c. Task-Forward-Info = from the Submission-Information-Record parameter of the RM-DEQUEUE-TASK indication.

   d. Arguments = from the Marshaled-Arguments parameter of the RM-DEQUEUE-TASK indication.

6. A-requeue

   Increment the retry count maintained for the submitted task entry. See Section 17.2.18 on page 315 for more information about retry counts.

7. A-rollback

   Issue a TRANS-ROLLBACK request.

States

1. S-null

   The null state.

2. S-deque-w

   Waiting for dequeue operation to complete.

3. S-forward-w

   Waiting for task forwarding operation to complete.

4. S-commit

   Waiting for normal commit.

5. S-rollb-r

   Waiting for transaction rollback before retrying the task forward.

6. S-rollb-n

   Waiting for transaction rollback when task forward will not be retried.

7. S-shutdwn

   Waiting for transaction rollback before shutting down.

State Transition Tables

	S-null	**S-deque-w**	**S-forward-w**
E-start	S-deque-w A-dequeue		
E-element		S-forward-w A-forward	
E-done			S-commit A-commit
E-exception+ P-retry+P-rec-exc			S-rollb-r A-rollback
E-exception+ (¬P-retry or¬P-rec-exc)			S-rollb-n A-rollback
E-rollb-rec+ P-retry			S-rollb-r A-rollback
E-rollb-rec+ ¬P-retry			S-rollb-n A-rollback
E-stop		S-shutdwn A-rollback	

Table 18-41 Task Forwarder (Null, Dequeue and Forward States)

	S-commit	**S-rollb-r**	**S-rollb-n**	**S-shutdwn**
E-commit-done	S-deque-w A-dequeue			
E-rollb-done+ P-retry	S-deque-w A-requeue A-dequeue	S-deque-w A-requeue A-dequeue	S-deque-w A-audit-failure A-dequeue	S-null A-cleanup
E-rollb-done+ ¬P-retry	S-deque-w A-audit-failure A-dequeue		S-deque-w A-audit-failure A-dequeue	S-null A-cleanup

Table 18-42 Task Forwarder (Commit, Rollback and Shutdown States)

18.8.3 Task Enqueuer State Machine

The Task Enqueuer state machine models the processing done to accept forwarded submitted task requests from a submitter TP system and store them on a client TP system.

No more than one Task Enqueuer state machine instance can exist within an execution context. How a Task Enqueuer state machine instance is created depends on the following:

1. If the task forwarding operation is non-transactional, each ENQUEUE-EXECUTE request creates a separate Task Enqueuer state machine instance.
2. If the task forwarding operation is transactional, the first ENQUEUE-EXECUTE request for a dialogue creates a separate Task Enqueuer state machine instance. Any additional ENQUEUE-EXECUTE requests on that dialogue are directed to the Task Enqueuer state machine that was created.

Service Primitives Provided

1. ENQUEUE-EXECUTE — request

   Executes a task forwarding request.

   This request has the following parameters:

   a. Transactional = a Boolean that is true if the forward is to be executed transactionally.
   b. Task-Forward-Info = the TASK-FORWARD-INFORMATION record for this forward as as described in Section 18.9.1.2 on page 426.
   c. Arguments = the information for this forward as described in Section 18.9.1.2 on page 426.

2. ENQUEUE-CANCEL — request

   Cancel the current task forward operation.

   This request has no parameters.

3. ENQUEUE-DONE — indication

   Reports that the task forward operation has successfully completed.

   This indication has no parameters.

4. ENQUEUE-EXCEPTION — indication

   Reports that the task forward operation has completed with an exception.

   This indication has the following parameters:

   a. Type = Non-transaction, Transient Transaction or Permanent Transaction
   b. Class = one of the classes defined in Appendix C on page 489.
   c. Code = implementation-specific
   d. Exception-Code-Group = implementation-specific
   e. Source = System
   f. Level = Current
   g. Exception-Procedure = implementation-specific string
   h. Exception-Procedure-Group = implementation-specific string

5. ENQUEUE-RELEASE-CONTEXT — indication

   Reports that the call context is no longer needed. This is used for transactional task forward operations.

   This indication has no parameters.

Service User

The service user of the Task Enqueuer state machine is the state machine instance that issued the ENQUEUE-EXECUTE request. This service user receives all indications generated by a Task Enqueuer state machine instance and the ENQUEUE-CANCEL request from that service user is directed to the Task Enqueuer state machine instance that was created for that service user.

State Machines Used

1. Resource Manager (RM-)
2. Transaction (TRANS-)

Variables

None.

Predicates

1. P-t-exc

   Is true if the exception is a transaction exception.

2. P-trans

   Is true if the Transactional parameter of the ENQUEUE-EXECUTE request is true.

3. P-valid

   Is true if the parameters for ENQUEUE-EXECUTE request are valid. This includes:

   a. Check Transactional parameter of the ENQUEUE-EXECUTE request to verify that the client TP system supports transactions
   b. The arguments passed are valid
   c. The authorization is valid and any access control checks pass
   d. The task queue is enabled

Events

1. E-commit

   A TRANS-COMMIT-DONE indication received.

2. E-cancel

   An ENQUEUE-CANCEL indication received.

3. E-canceled

   A RM-CANCELED indication received.

4. E-done

An RM-DONE indication received.

5. E-exception

   An RM-EXCEPTION indication received.

6. E-forward

   An ENQUEUE-EXECUTE request received.

7. E-rollb

   A TRANS-ROLLBACK-DONE indication received.

8. E-rollb-rec

   A TRANS-ROLLBACK-RECEIVED indication received.

Actions

1. A-cancel

   Issue an RM-CANCEL request.

2. A-canceled

   Issue an ENQUEUE-EXCEPTION indication with the following parameters:

   a. Type = one of the following:

      — Non-transactional if the forward is non-transactional

      — Permanent Transaction if the forward is transactional

   b. Class = ENV-EXECUTION-ERROR

   c. Code = implementation-specific

   d. Exception-Code-Group = implementation-specific

   e. Source = System

   f. Level = Current

   g. Exception-Procedure = implementation-specific string

   h. Exception-Procedure-Group = implementation-specific string

3. A-commit

   Issue a TRANS-COMMIT request.

4. A-done

   Issue an ENQUEUE-DONE indication.

5. A-enqueue

   Issue an RM-ENQUEUE-TASK request with the following parameters:

   a. Local-Client = true.

   b. Local-Server = is true if the DESTINATION-NAME field of the Task-Forward-Info parameter of the ENQUEUE-EXECUTE request is on the client TP system.

   c. Client-RTI-AP-Title = omitted.

   d. Submission-Information-Record = from the Task-Forward-Info parameter of the ENQUEUE-EXECUTE request.

   e. Marshaled arguments = from the Argument parameter of the ENQUEUE-EXECUTE REQUEST.

6. A-exception

   Issue an ENQUEUE-EXCEPTION indication with the following parameters:

   a. Type = one of the following:

      — Non-transactional if the forward is non-transactional

      — From the Class parameter as defined in Appendix C on page 489 if the forward is transactional

   b. Class = one of the following:

      — FATAL-TIMEOUT-FAULT = if an implementation-specific CPU timeout expired (this is not a required feature)

      — TXN-FAILURE-ERROR = if the transaction could not commit due to a resource manager failing to commit — non-transaction forwarding only

      — TXN-TIMEOUT-ERROR = if the transaction could not complete due to a transaction timeout — non-transaction forwarding only (this is not a required feature)

      — ENV-EXECUTION-FAULT = if the client TP system cannot store the task submission request due to a permanent error

      — ENV-EXECUTION-ERROR = if the client TP system cannot store the task submission request due to a transient error

   c. Code = implementation-specific

   d. Exception-Code-Group = implementation-specific

   e. Source = System

   f. Level = Current

   g. Exception-Procedure = implementation-specific string

   h. Exception-Procedure-Group = implementation-specific string

7. A-exception-r

   Issue an ENQUEUE-EXCEPTION indication with the following parameters:

   a. Type = Non-transaction

   b. Class = TXN-FAILURE-ERROR

   c. Code = implementation-specific

   d. Exception-Code-Group = implementation-specific

   e. Source = System

   f. Level = Propagated

   g. Exception-Procedure = implementation-specific string

h. Exception-Procedure-Group = implementation-specific string

8. A-reject

   Issue an ENQUEUE-EXCEPTION indication with the following parameters:

   a. Type = from the Class parameter defined in Appendix C on page 489.

   b. Class is set as follows:

      — If task was forwarded using transactions and the client TP system does not support distributed transactions = ENV-INVOCATION-FAULT

      — If arguments passed are not valid = AP-INVOCATION-FAULT

      — If authorization is not valid or access control checks failed = ENV-INVOCATION-FAULT

      — The task queue is disabled = ENV-INVOCATION-ERROR

   c. Code = implementation-specific

   d. Exception-Code-Group = implementation-specific

   e. Source = System

   f. Level = Current

   g. Exception-Procedure = implementation-specific string

   h. Exception-Procedure-Group = implementation-specific string

9. A-release

   Issue an ENQUEUE-RELEASE-CONTEXT indication.

10. A-rollb

    Issue a TRANS-ROLLBACK request.

11. A-trans-n

    Issue a TRANS-START request with the following parameters:

    a. Type = Non-Root

12. A-trans-r

    Issue a TRANS-START request with the following parameters:

    a. Type = Root

States

1. S-null

   The initial state.

2. S-nt-enque

   Waiting for the task to be queued in a non-transactional task forwarding operation.

3. S-nt-cancel

   Waiting for the enqueue operation to cancel during a non-transactional task forwarding operation.

4. S-nt-commit

   Waiting for the local transaction to commit in a non-transactional task forwarding operation.

5. S-nt-rollb

   Waiting for the local transaction to rollback in a non-transactional task forwarding operation.

6. S-nt-rollb-can

   Waiting for the local transaction to rollback in a non-transactional task forwarding operation after a cancel.

7. S-t-enque

   Waiting for the task to be queued in a transactional task forwarding operation.

8. S-t-inactive

   Waiting for the transaction to be terminated or another task to be forwarded in the transaction.

9. S-t-inactive-r

   Waiting for the transaction to be rolled back.

10. S-t-can-can

    Waiting for the enqueue operation to cancel in a transactional task forwarding operation so that a cancel acknowledgement can be returned.

11. S-t-can-rollb

    Waiting for the enqueue operation to cancel after a rollback was received in a transactional task forwarding operation.

12. S-t-rollb

    Waiting for the distributed transaction to rollback in a transactional task forwarding operation.

State Transition Tables

	S-null
E-forward+ ¬P-valid	S-null A-reject
E-forward+ P-valid+P-trans	S-t-enque A-trans-n, A-enqueue
E-forward+ P-valid+¬P-trans	S-nt-enque A-trans-r, A-enqueue

Table 18-43 Task Enqueuer (Null State)

	S-nt-enque	**S-nt-cancel**	**S-nt-commit**	**S-nt-rollb**	**S-nt-rollb-can**
E-cancel	S-nt-cancel A-cancel		S-nt-commit	S-nt-rollb	
E-done	S-nt-commit A-commit				
E-exception	S-nt-rollb A-rollb				
E-canceled		S-nt-rollb-can A-rollb			
E-commit			S-null A-done		
E-rollb			S-null A-exception-r	S-null A-exception	S-null A-canceled

Table 18-44 Task Enqueuer (Non-transactional Forwards)

	S-t-enque	**S-t-can-can**	**S-t-can-rollb**	**S-t-inactive**	**S-t-inactive-r**	**S-t-rollb**
E-cancel	S-t-can-can A-cancel					
E-forward+ P-trans+P-valid				S-t-enque A-enqueue		
E-forward+ P-trans+¬P-valid				S-t-inactive-r A-reject		
E-forward+ ¬P-trans				S-t-inactive-r A-reject		
E-done	S-t-inactive A-done					
E-exception+ P-t-exc	S-t-inactive-r A-exception					
E-exception+ ¬P-t-exc	S-t-inactive A-exception					
E-canceled		S-t-inactive-r A-canceled	S-t-rollb A-rollb			
E-commit				S-null A-release		
E-rollb						S-null A-release
E-rollb-rec	S-t-can-rollb A-cancel	S-t-can-rollb		S-t-rollb A-rollb	S-t-rollb A-rollb	

Table 18-45 Task Enqueuer (Transactional Forwards)

18.9 Communication State Machines

18.9.1 RPC Interfaces Used

TP systems supporting STDL applications communicate using the remote procedure call (RPC) protocol described in the referenced X/Open TxRPC specification and in Appendix H on page 533. The interaction can be either a task invocation or a task forwarding operation.

For receiving a task invocation from an external client program, the protocol is defined in Appendix H on page 533. For any other task invocation or a task forwarding operation, the protocol is defined in the referenced X/Open TxRPC specification.

In order to use the RPC protocol, the interface to the dialogue server TP system must be well defined. This is done by the use of an interface definition. An *interface definition* is the definition of:

- Interface UUID

  The interface UUID uniquely identifies the interface.

- Interface Version Number

  The interface version number identifies the version of the interface. The interface version number follows the rules outlined for <version> in the <task-group-specification> in Section 9.2.2 on page 160.

- Ordered List of Procedures in the Interface

  For each procedure the following is defined:

  — An operation value

    The operation value identifies which RPC procedure is being called. Each procedure in the RPC interface has a different operation value. The operation values are assigned to procedures from 0 to n according to the order of the procedures within the RPC interface definition.

  — Whether the procedure is transactional

    This is the transaction_mandatory attribute specified in the referenced X/Open TxRPC specification.

  — Ordered list of arguments

    For each argument the following is defined:

    — Argument data type = this is one of the STDL data types defined in Section 5.2 on page 123.

    — Whether the argument is input to the remote procedure, output from the remote procedure or both input and output.

When a remote procedure call is performed, data from the dialogue client must be sent to the dialogue server to start the call and data from the dialogue server must be sent to the dialogue client to finish the call. When data is sent from one system to another, the data must be transformed from the form it has within a system into a standard format that can be transmitted. *Marshalling* is the process of transforming data from internal system representation into a standard format that can be transmitted. When the data is received, the data must be transformed from the standard format into a form that can be used on the local system. *Unmarshalling* is the process of transforming data from the standard format that was transmitted into the internal system representation.

The data for the Arguments parameter on the RTI-CALL-TASK and RTI-CALL-RESULTS requests must be marshaled before invoking the request. The data for the Arguments parameter on the RTI-CALL-TASK and RTI-CALL-RESULTS indications must be unmarshaled before the data can be used.

On the RTI-CALL-TASK service primitive, the Arguments parameter consists of the marshaled argument values for the arguments in the RPC procedure definition that are either input or both input and output. Those argument values are marshaled and unmarshaled in the order that the arguments appear in the RPC definition.

On the RTI-CALL-RESULTS service primitive, the Arguments parameter consists of the marshaled argument values for the arguments in the RPC procedure definition that are either output or both input and output. These argument values are marshaled and unmarshaled in the order that the arguments appear in the RPC definition.

The standard format that is used in transmitting data between systems is defined in Appendix G on page 529.

The interface definition used by the dialogue client system must be the same as the interface definition used by the dialogue server TP system. The following sections describe how this is done for task calls and for task forwarding operations.

18.9.1.1 Interfaces for Calling Tasks

For calling tasks, the RPC interface definition is based on the <task-group-specification>. By sharing the <task-group-specification> and applying the following rules, the dialogue client system and the dialogue server TP system share the same RPC interface definition.

The rules for transforming a <task-group-specification> are:

1. Use the UUID in the <task-group-specification> for the RPC interface UUID.
2. Use the version number in the <task-group-specification> for the RPC interface version number.
3. Use the list of tasks in the <task-group-specification> for the list of procedures in the RPC interface.

   For each task do the following:

   a. Assign the operation value according to the position of the <task-interface> within the <task-group-specification> starting with 0.
   b. If the task is composable, the procedure is transactional.
   c. Construct the argument list for the RPC procedure according to the following rules:
      - The first RPC procedure argument has the data type TASK-CALL-INFORMATION record as defined below and is input to the RPC procedure.
      - The second RPC procedure argument has the data type EXCEPTION-INFORMATION record as defined in Section 18.9.1.3 on page 430 and is output from the RPC procedure.
      - The rest of the RPC procedure's arguments are the arguments for the task. The task arguments are offset by 2 in the RPC procedure argument list. Task argument 1 is RPC procedure argument 3, task argument 2 is RPC procedure argument 4, and so on.

        The data type and access for these arguments are as defined in the <task-interface> in the <task-group-specification> for the server task.

The TASK-CALL-INFORMATION record contains information for the server task to use as passed from the client.

The data type definition for the TASK-CALL-INFORMATION record type is never seen by the application programmer. The data type definition for the TASK-CALL-INFORMATION record type is:

```
TYPE TASK-CALL-INFORMATION IS RECORD
    FORMAT-UUID UUID;
    RECORD-VERSION RECORD
        RECORD-VERSION-MAJOR INTEGER;
        RECORD-VERSION-MINOR INTEGER;
    END RECORD;
    DISPLAY-NAME TEXT CHARACTER SET SIMPLE-LATIN SIZE 256;
    DISPLAY-TP-SYSTEM TEXT CHARACTER SET SIMPLE-LATIN SIZE 256;
    LANGUAGE TEXT CHARACTER SET SIMPLE-LATIN SIZE 16;
END RECORD;
```

1. FORMAT-UUID

   This field identifies the format of the record. The UUID must be: 53E67AC0-9D3A-11CA-80AB-08002B14B188.

2. RECORD-VERSION

   This record contains the version number of the format of the TASK-CALL-INFORMATION record. This field consists of two subfields:

   a. RECORD-VERSION-MAJOR

      This field contains the major version number for the TASK-CALL-INFORMATION record. This number is changed when there is an incompatible change to the TASK-CALL-INFORMATION record.

      Currently, this field must contain a 1.

   b. RECORD-VERSION-MINOR

      This field contains the minor version number for the TASK-CALL-INFORMATION record. This number is changed when there is a compatible change to the TASK-CALL-INFORMATION record.

      Currently, this field must contain a 0.

3. DISPLAY-NAME

   This field contains the symbolic name for the display as passed from the client or submitter task. This name is meaningful on the TP system as identified in DISPLAY-TP-SYSTEM.

   This field can be null. The null value is all spaces. A null value means that there is no meaningful display for the client or submitter task. If this field is not null, then the DISPLAY-TP-SYSTEM field must not be null. If this field is null, then the DISPLAY-TP-SYSTEM field must be null.

4. DISPLAY-TP-SYSTEM

   This field contains the TP system name on which the display named in the DISPLAY-NAME field is meaningful. This field (if not null) must be usable as a destination name in a <call-task> on the server TP system.

This field can be null. The null value is all spaces. A null value means that there is no display for the client (or submitter) task. If this field is not null, then the DISPLAY-NAME field must not be null. If this field is null, then the DISPLAY-NAME field must be null.

5. LANGUAGE

   This field contains the human language as passed from the client or submitter task. It must contain one of the strings listed in Section 4.5.6 on page 82.

   This field must not be null.

18.9.1.2 Interface for the Task Enqueuer

For task forwarding, a fixed RPC interface is used. This RPC interface definition is:

1. UUID is 40FE0BA0-B3A1-11C9-990D-08002B102989.
2. Version number is 1.0.
3. Procedure list is:
   a. Non-transactional task forwarding procedure
      — Operation value = 0
      — Argument list as defined below.
   b. Transactional task forwarding procedure
      — Operation value = 1
      — The procedure as transactional.
      — Argument list is defined below.

The RPC procedure argument list for both procedures is:

1. Forward-Info
   a. Data type = TASK-FORWARD-INFORMATION record
   b. Access = INPUT
2. Exception-Info
   a. Data type = EXCEPTION-INFORMATION record
   b. Access = OUTPUT
3. Submitted-task-arguments
   a. Data type = SUBMITTED-TASK-ARGUMENT record
   b. Access = INPUT

      The data type definition of SUBMITTED-TASK-ARGUMENT record type is:

```
TYPE SUBMITTED-TASK-ARGUMENT IS RECORD
    DATA-SIZE INTEGER;
    MARSHALED-DATA ARRAY SIZE x
        DEPENDING ON DATA-SIZE OF OCTET;
END RECORD;
```

      The required minimum limit of SIZE x that an implementation must support is defined in Appendix A on page 481.

The TASK-FORWARD-INFORMATION record type contains information for use by the client TP system about the task submission.

The data type definition for the TASK-FORWARD-INFORMATION record type is never seen by the application programmer. The TASK-FORWARD-INFORMATION type definition is:

```
TYPE TASK-FORWARD-INFORMATION IS RECORD
    FORMAT-UUID UUID;
    RECORD-VERSION RECORD
        RECORD-VERSION-MAJOR INTEGER;
        RECORD-VERSION-MINOR INTEGER;
    END RECORD;
    SERVER-TASK-INFORMATION RECORD
        TASK-GROUP-UUID UUID;
        TASK-OPERATION-ID INTEGER;
        DESTINATION-NAME TEXT CHARACTER SET SIMPLE-LATIN SIZE 256;
        VERSION-NUMBER RECORD
            VERSION-MAJOR INTEGER;
            VERSION-MINOR INTEGER;
        END RECORD;
        COMPOSABLE-FLAG INTEGER;
    END RECORD;
    DISPLAY-NAME TEXT CHARACTER SET SIMPLE-LATIN SIZE 256;
    DISPLAY-TP-SYSTEM TEXT CHARACTER SET SIMPLE-LATIN SIZE 256;
    LANGUAGE TEXT CHARACTER SET SIMPLE-LATIN SIZE 16;
    TRIGGER-INFORMATION RECORD
        TRIGGER-TYPE INTEGER;
        TRIGGER-TIME TEXT CHARACTER SET SIMPLE-LATIN SIZE 14;
    END RECORD;
    REPEAT-TIME TEXT CHARACTER SET SIMPLE-LATIN SIZE 10;
    SUBMISSION-INFORMATION RECORD
        SUBMIT-TIME TEXT CHARACTER SET SIMPLE-LATIN SIZE 14;
        SUBMIT-TP-SYSTEM TEXT CHARACTER SET SIMPLE-LATIN SIZE 256;
        SUBMIT-TASK-GROUP TEXT CHARACTER SET SIMPLE-LATIN SIZE 32;
        SUBMIT-TASK TEXT CHARACTER SET SIMPLE-LATIN SIZE 32;
        SERVER-TP-SYSTEM TEXT CHARACTER SET SIMPLE-LATIN SIZE 256;
        SERVER-TASK-GROUP TEXT CHARACTER SET SIMPLE-LATIN SIZE 32;
        SERVER-TASK TEXT CHARACTER SET SIMPLE-LATIN SIZE 32;
    END RECORD;
END RECORD;
```

1. FORMAT-UUID

   This field identifies the format of the record. The UUID must be: 59BCC51D-9D3A-11CA-B614-08002B14B188.

2. RECORD-VERSION

   This record contains the version number of the format of the TASK-FORWARD-INFORMATION record. This field consists of two subfields:

a. RECORD-VERSION-MAJOR

This field contains the major version number for the TASK-FORWARD-INFORMATION record. This number is changed when there is an incompatible change to the TASK-FORWARD-INFORMATION record.

Currently, this field must contain a 1.

b. RECORD-VERSION-MINOR

This field contains the minor version number for the TASK-FORWARD-INFORMATION record. This number is changed when there is a compatible change to the TASK-FORWARD-INFORMATION record.

Currently, this field must contain a 0.

3. SERVER-TASK-INFORMATION

This record contains information required by the client TP system to issue the remote procedure call to the server TP system.

a. TASK-GROUP-UUID

This field contains the UUID of the server task group.

This field must not be null.

b. TASK-OPERATION-ID

This field contains the operation ID of the server task.

This field must not be null.

c. DESTINATION-NAME

This field contains the destination name for the task call to the server TP system

This field must not be null.

d. VERSION-NUMBER

— VERSION-MAJOR

This field contains the major version number of the server task group.

This field must not be null.

— VERSION-MINOR

This field contains the minor version number of the server task group.

This field must not be null.

e. COMPOSABLE-FLAG

This field indicates whether or not the server task is composable. This field can contain one of two values:

1 The server task is noncomposable.

2 The server task is composable.

This field must not be null.

4. DISPLAY-NAME

   This field contains the value of the DISPLAY-NAME field of the TASK-CALL-INFORMATION record in the server task's marshaled arguments. It follows the same rules as that field.

5. DISPLAY-TP-SYSTEM

   This field contains the value of the DISPLAY-TP-SYSTEM field of the TASK-CALL-INFORMATION record in the server task's marshaled arguments. It follows the same rules as that field.

6. LANGUAGE

   This field contains the value of the LANGUAGE field of the TASK-CALL-INFORMATION record in the server task's marshaled arguments. It follows the same rules as that field.

7. TRIGGER-INFORMATION

   a. TRIGGER-TYPE

      This field contains the type of trigger information for the server task. This field can contain one of the following values:

      0 No trigger (this is the null value).

      1 Trigger is an operator hold.

      2 Trigger is specified as absolute universal time. TRIGGER-TIME contains the time when the server task is first called

      3 Trigger is specified as absolute client time. TRIGGER-TIME contains the time when the server task is first called.

   b. TRIGGER-TIME

      If TRIGGER-TYPE field contains a value of 2, this field contains the absolute universal time at which the first server task for this submission request is to be called.

      If TRIGGER-TYPE field contains a value of 3, this field contains the absolute client time, as interpreted by the client, at which the first server task for this submission request is to be called.

      This field is formatted as described in Section 4.17.1 on page 117.

      This field must not be null if the field TRIGGER-TYPE contains a value of 2 or 3. Otherwise, this field must be null. The null value is all spaces.

8. REPEAT-TIME

   This field contains the repeat time. If this field is not null, then the client TP system repeatedly calls the server task, with the calls separated by the time specified in this field. If this field is null, then the client TP system calls the server task once. This field is formatted as described in Section 4.17.2 on page 118.

   This field can be null. The null value is all spaces.

9. SUBMISSION-INFORMATION

   This record contains information that identifies the submission request. This information is for use by the client TP system in operator commands that display the contents of task queues and manipulates the entries of the task queue as described in Section 17.2.18.1 on page 316.

a. SUBMIT-TIME

This field contains the universal, absolute time when the submitter task executed the <submit-task> . The field is formatted as described in Section 4.17.1 on page 117.

This field must not be null.

b. SUBMIT-TP-SYSTEM

This field contains the destination name of the submitter TP system. Destination names are defined in Section 4.5.3 on page 81.

This field must not be null.

c. SUBMIT-TASK-GROUP

This field contains the name of the task group of the submitter task.

This field must not be null.

d. SUBMIT-TASK

This field contains the name of the submitter task.

This field must not be null.

e. SERVER-TP-SYSTEM

This field contains the destination name of the server TP system. Destination names are defined in Section 4.5.3 on page 81.

This field must not be null.

f. SERVER-TASK-GROUP

This field contains the name of the task group of the server task.

This field must not be null.

g. SERVER-TASK

This field contains the name of the server task.

This field must not be null.

18.9.1.3 Exceptions

The EXCEPTION-INFORMATION record returns exceptions from a server task or a task enqueuer.

The data type definition for the EXCEPTION-INFORMATION record type is never seen by the application programmer. The EXCEPTION-INFORMATION record type is:

```
TYPE EXCEPTION-INFORMATION IS RECORD
    FORMAT-UUID UUID;
    RECORD-VERSION RECORD
        RECORD-VERSION-MAJOR INTEGER;
        RECORD-VERSION-MINOR INTEGER;
    END RECORD;
    EXCEPTION-TYPE INTEGER;
    EXCEPTION-CLASS INTEGER;
    EXCEPTION-CODE INTEGER;
    EXCEPTION-CODE-GROUP UUID;
    EXCEPTION-LEVEL INTEGER;
    EXCEPTION-SOURCE INTEGER;
    EXCEPTION-PROCEDURE
        TEXT CHARACTER SET SIMPLE-LATIN SIZE 32;
    EXCEPTION-PROCEDURE-GROUP
        TEXT CHARACTER SET SIMPLE-LATIN SIZE 32;
END RECORD;
```

1. FORMAT-UUID

   This field identifies the format of the record. The UUID must be: 5B7EC918-9D3A-11CA-897F-08002B14B188.

2. RECORD-VERSION

   This record contains the version number of the format of the EXCEPTION-INFORMATION record. This field consists of two subfields:

   a. RECORD-VERSION-MAJOR

      This field contains the major version number for the EXCEPTION-INFORMATION record. This number is changed when there is an incompatible change to the EXCEPTION-INFORMATION record.

      Currently, this field must contain a 1.

   b. RECORD-VERSION-MINOR

      This field contains the minor version number for the EXCEPTION-INFORMATION record. This number is changed when there is a compatible change to the EXCEPTION-INFORMATION record.

      Currently, this field must contain a 0.

3. EXCEPTION-TYPE

   This field determines whether or not there is an exception and the type of exception. This field can contain the following values:

   0 No exception (this is the null value).

   1 Non-transaction exception.

   2 Transient transaction exception.

   3 Permanent transaction exception.

   4 Fatal transaction exception.

4. EXCEPTION-CLASS

   This field contains the exception class for an exception. This field can contain one of the values listed in Section A.2 on page 486.

   This field cannot be null if the field EXCEPTION-TYPE is not null. If the EXCEPTION-TYPE field is null, the value of this field is undefined.

5. EXCEPTION-CODE

   This field contains the exception code for an exception. If the EXCEPTION-TYPE field is not null and EXCEPTION-SOURCE is 1 (application), then this field contains a message number as defined in a <message-group-definition>. If the EXCEPTION-TYPE field is not null and EXCEPTION-SOURCE is 0 (system), then this field contains an implementation-specific value.

   If the EXCEPTION-TYPE field is null, the value of this field is undefined.

6. EXCEPTION-CODE-GROUP

   This field contains the exception group UUID for exceptions. If the EXCEPTION-TYPE field is not null and EXCEPTION-SOURCE is 1 (application), then this field is either all zeros or contains the value of the specified <message-group-definition> UUID, if any. If the EXCEPTION-TYPE field is not null and EXCEPTION-SOURCE is 0 (system), then this field contains an implementation-specific UUID.

7. EXCEPTION-LEVEL

   This field contains the exception level for an exception. This field can contain one of the following values:

   0 Current.

   1 Propagated.

   If the EXCEPTION-TYPE field is null, the value of this field is undefined.

8. EXCEPTION-SOURCE

   This field contains the exception source for an exception. This field can contain one of the following values:

   0 System.

   1 Application.

   If the EXCEPTION-TYPE field is null, the value of this field is undefined.

9. EXCEPTION-PROCEDURE

   This field contains the exception procedure in which the exception was raised.

   If the EXCEPTION-TYPE field is null, the value of this field is undefined.

10. EXCEPTION-PROCEDURE-GROUP

    This field contains the exception procedure group in which the exception was raised.

    If the EXCEPTION-TYPE field is null, the value of this field is undefined.

18.9.2 Call State Machine

The Call state machine models the communication between execution contexts both within a single TP system and between TP systems. This includes the construction and interpretation of the RPC argument list as discussed in Section 18.9.1 on page 423 for communications between TP systems.

Call state machines operate in one of three modes:

1. local call
2. client to remote TP system
3. server from remote system.

A separate state table is provided for each mode.

One Call state machine instance exists within an execution context for:

1. Each active task that an execution context is invoking on the local TP system.

   If the called task is a noncomposable task, these Call state machine instances are actually in both the client execution context (either task, task dequeuer or client program) and the called, noncomposable task.

2. Each transactional dialogue that an execution context is using as a client to a TP system.

   A separate transactional dialogue is used for each unique combination of the following:

   a. RTI-AP-Title
   b. RPC interface UUID
   c. RPC interface version.

   (Dialogue is called context in the referenced X/Open TxRPC specification and Appendix H on page 533.)

3. Each non-transactional dialogue that an execution context is using as a client to a TP system.

   A separate non-transactional dialogue is used for each unique combination of the following:

   a. RTI-AP-Title
   b. RPC interface UUID
   c. RPC interface version.

   (Dialogue is called context in the referenced X/Open TxRPC specification and Appendix H on page 533.)

4. Each task being invoked by a remote system.

Service Primitives Provided

1. CALL-TASK — request, task invocation

   Initiates a task call when the input and input/output arguments for the task have not been combined with the TASK-CALL-INFORMATION record to form the RPC procedure argument list and that argument list is not already marshaled. The request causes the Call state machine to combine the TASK-CALL-INFORMATION record and Arguments parameters into a single RPC argument list as defined in Section 18.9.1.1 on page 424 and then marshal that argument list.

   This service primitive has the following parameters:

   a. Local = a Boolean that is true if this is a local task call

      This Boolean is true if the client task omits the "IN" and "AT" phrases on <call-task>. However, an implementation can make this true whenever the client and the server are on the same TP system.

   b. RTI-AP-Title = of the server, used only if Local is false

   c. Task-Group-UUID = the UUID for the server task group

   d. Task-Group-Version-Major = the major version number of the server task group

   e. Task-Group-Version-Minor = the minor version number of the server task group

   f. Task-Operation-Value = the operation value of the server task

   g. Transactional = a Boolean that is true if the server task is composable

   h. Task-Call-Info = the TASK-CALL-INFORMATION record for this task call

   i. Arguments = the INPUT and INOUT arguments as defined for the server task in the server task group specification

2. CALL-TASK-MARSHALED — request, task invocation

   Initiates a task call when the input and input/output arguments for the RPC procedure for the task have already been combined with the TASK-CALL-INFORMATION record to form an RPC argument list as described in Section 18.9.1.1 on page 424 and that argument list has already been marshaled. The request causes the Call state machine to map the Arguments parameter of the CALL-TASK request directly to the Arguments parameter of the RTI-CALL-TASK request.

   This service primitive has the following parameters:

   a. Local = a Boolean that is true if this is a local task call

   b. RTI-AP-Title = of the server, used only if Local is false

   c. Task-Group-UUID = the UUID for the server task group

   d. Task-Group-Version-Major = the major version number of the server task group

   e. Task-Group-Version-Minor = the minor version number of the server task group

   f. Task-Operation-Value = the operation value of the server task

   g. Transactional = a Boolean that is true if the server task is composable

   h. Marshaled-Arguments = the marshaled INPUT and INOUT arguments for the task call combined with the TASK-CALL-INFORMATION record

3. CALL-FORWARD-TASK — request, task forwarding

   Initiates a task forwarding operation. The request causes the Call state machine to combine the TASK-FORWARD-INFORMATION record with the Argument parameter into a single RPC argument list as defined in Section 18.9.1.2 on page 426.

   This service primitive has the following parameters:

   a. RTI-AP-Title = of the client TP system's task enqueuer

   b. Transactional = indicates whether the forward operation is to be done using transactions

   c. Task-Forward-Info = the TASK-FORWARD-INFORMATION record for this task forward

   d. Arguments = the marshaled data to be given to the client TP system. The client TP system uses the marshaled data in the RTI-CALL-TASK request. The marshaled data must consist of the input arguments for the server task combined with the TASK-CALL-INFORMATION record as described in Section 18.9.1.2 on page 426.

4. CALL-CANCEL — request, task invocation

   Cancel the current task call.

   This service primitive has no parameters.

5. CALL-CANCELED — indication, task invocation

   Reports that a task call has been cancelled. An implementation can use the semantics of CALL-DONE and/or CALL-EXCEPTION depending on implementation design and timing.

   The service primitive has no parameters.

6. CALL-DONE — indication, both task invocation and task forwarding

   Reports that a call completed successfully and returns the output arguments.

   This service primitive has the following parameters:

   a. Arguments = the OUTPUT and INOUT arguments as defined for the server task in the server task group specification (this is null for task forward operations).

7. CALL-EXCEPTION — indication, both task invocation and task forwarding

   Reports that an exception has occurred. This service primitive has the following parameters:

   a. Type = Non-transaction, Transient Transaction or Permanent Transaction

   b. Class = one of the classes defined in Appendix C on page 489.

   c. Code = the exception code as defined in Section 3.7.2 on page 62.

   d. Exception-Code-Group = the exception code group as defined in Section 3.7.2 on page 62.

   e. Source = System or Application

   f. Level = Current or Propagated

   g. Exception-Procedure = the procedure in which the exception occurred as defined in Section 3.7.2 on page 62.

   h. Exception-Procedure-Group = the procedure group in which the exception occurred as defined in Section 3.7.2 on page 62.

8. CALL-RELEASE-CONTEXT — request, both task invocation and task forwarding

   Terminates Call context machine instances with V-local false and V-client true.

   This request has no parameters.

Service User

When operating as a server for a remote client, there is no service user.

In all other cases, the service user is the state machine instance that issues the CALL-TASK or CALL-FORWARD-TASK request. If the CALL-TASK request is used and Local is false, if there is a Call state machine instance in the same execution context with the V-call set true and the correct RTI-AP-Title, Task-Group-UUID, Task-Group-Version, and Transactional values in V-call-context, then that state machine instance is used. If there is no such Call state machine instance or the Local parameter of the CALL-TASK request is true, a new one is created.

If the CALL-FORWARD-TASK request is used, if there is a Call state machine instance in the same execution context with V-call set false and the correct RTI-AP-Title and Transactional values in V-forward-context, then that Call state machine instance is used. If there is no such Call state machine instance, a new one is created.

The CALL-DONE, CALL-EXCEPTION and CALL-CANCELED indications are directed to the state machine instance that issued the CALL-TASK or CALL-FORWARD-TASK request. A CALL-CANCEL request is directed to the Call state machine instance that the service user is currently using.

The CALL-RELEASE-CONTEXT request is always sent to all Call state machine instances within the execution context.

State Machines Used

1. Remote Task Invocation Protocol Machine (RTI-)
2. Task Enqueuer (ENQUEUE-)
3. Task (TASK-)
4. Transaction (TRANS-)

Variables

1. V-call

   Is true if this state machine instance is being used for task calls and false if it is being used for task forwards.

   This is a Boolean variable.

2. V-call-context

   Is the following information about a call context together with V-call and V-trans makes a Call state machine instance unique for clients calling remote tasks:

   a. RTI-AP-Title

   b. Task-Group-UUID

   c. Task-Group-Version-Major
   d. Task-Group-Version-Minor.

   This variable stores data.

3. V-client

   Is true if the Call state machine instance is for clients calling tasks on remote server TP systems.

   This is a Boolean variable.

4. V-forward-context

   Is the RTI-AP-Title, which together with the values of V-call and V-trans makes a Call state machine instance unique for clients doing task forwards.

   This variable stores data.

5. V-local

   Is true if the client is local to the server.

   This is a Boolean variable.

6. V-trans

   Is true if the call or forward is transactional.

   This is a Boolean variable.

Predicates

1. P-call

   Is true if V-call is true.

2. P-exception

   Is true if the EXCEPTION-TYPE field of the EXCEPTION-INFORMATION record in the Arguments parameter of the RTI-CALL-RESULT indication is not null as defined in Section 18.9.1.3 on page 430.

3. P-failure-fatal

   Is true when the Reason parameter of the RTI-CALL-FAILURE indication is one of the following:

   a. RTI-SERVICE-UNKNOWN
   b. PROTOCOL-VERSION-NOT-SUPPORTED
   c. CONTEXT-TYPE-NOT-SUPPORTED
   d. PERMANENT-COMMUNICATION-FAILURE
   e. TRANSIENT-COMMUNICATION-FAILURE
   f. PROTOCOL-MACHINE-FAILURE
   g. INTERFACE-UNKNOWN
   h. INTERFACE-PERMANENTLY-UNAVAILABLE
   i. INTERFACE-TEMPORARILY-UNAVAILABLE

   j. ROLLBACK-IN-PROGRESS

   k. REASON-NOT-SPECIFIED.

4. P-local

   Is true if the Local parameter of CALL-TASK or CALL-TASK-MARSHALLED request is true.

5. P-trans

   Is true if the Transactional parameter of the CALL-TASK, CALL-TASK-MARSHALLED or CALL-FORWARD-TASK request is true or Context-Type parameter of the RTI-CALL-TASK indication is Transactional.

Events

1. E-call

   A CALL-TASK request or a CALL-TASK-MARSHALED request received.

2. E-call-release

   A CALL-RELEASE-CONTEXT request received.

3. E-cancel

   A CALL-CANCEL request received.

4. E-canceled

   A TASK-CANCELED indication received.

5. E-done

   A TASK-DONE or ENQUEUE-DONE indication received.

6. E-exception

   A TASK-EXCEPTION or ENQUEUE-EXCEPTION indication received.

7. E-forward

   A CALL-FORWARD-TASK request received.

8. E-RTI-call

   An RTI-CALL-TASK indication received.

9. E-RTI-cancel

   An RTI-CANCEL-CALL indication received.

10. E-RTI-failure

    An RTI-CALL-FAILURE indication received.

11. E-RTI-release

    An RTI-RELEASE-CONTEXT indication received.

12. E-RTI-result

    An RTI-CALL-RESULT indication received.

13. E-task-release

A TASK-RELEASE-CONTEXT or ENQUEUE-RELEASE-CONTEXT indication received.

Actions

1. A-call

   Issue a TASK-EXECUTE request with the following parameters:

   a. Local = value of V-local

   b. Trans = value of V-trans

   c. Task-Operation-Value = depends on the value of V-local

      — If V-local is true, then from the Task-Operation-Value of the CALL-TASK request or CALL-TASK-MARSHALED request

      — If V-local is not true, then the Operation-Number from the RTI-CALL-TASK indication

   d. Task-Call-Info = depends on the value of V-local

      — If V-local is true, then from the Task-Call-Info parameter of the CALL-TASK request or unmarshaled from the Marshaled-Arguments parameter of the CALL-TASK-MARSHALED request as described in Section 18.9.1.1 on page 424.

      — If V-local is false, then unmarshaled from the Arguments parameter of the RTI-CALL-TASK indication as described in Section 18.9.1.1 on page 424.

   e. Arguments = depends on the value of V-local

      — If V-local is true, then from Arguments parameter of the CALL-TASK request or unmarshaled from the Marshaled-Arguments parameter of the CALL-TASK-MARSHALED request as described in Section 18.9.1.1 on page 424.

      — If V-local is false, then unmarshaled from the Arguments parameter of the RTI-CALL-TASK indication as described in Section 18.9.1.1 on page 424.

2. A-cancel

   Do one of the following:

   a. If V-call is true, issue a TASK-CANCEL request

   b. If V-call is false, issue a ENQUEUE-CANCEL request

3. A-canceled

   Issue a CALL-CANCELED indication.

4. A-dist

   Issue a TRANS-DISTRIBUTED request.

5. A-done

   Issue a CALL-DONE indication with the following parameters:

   a. Arguments = depends on

      — If V-call is true, depends on

         — If local, then derived from Arguments parameter of the TASK-DONE indication

    — If remote, then derived from the RTI-CALL-RESULT indication as described in Section 18.9.1.1 on page 424.

  — If V-call is false, then null

6. A-exception

   Issue a CALL-EXCEPTION indication with the following parameters derived from the TASK-EXCEPTION indication if local or RTI-CALL-RESULT indication as described in Section 18.9.1.3 on page 430 if a server for a remote client:

   a. Type
   b. Class
   c. Code
   d. Exception-Code-Group
   e. Source
   f. Level
   g. Exception-Procedure
   h. Exception-Procedure-Group

7. A-failure-exc

   Issue a CALL-EXCEPTION indication with the following parameters:

   a. Type = Derived from the Class as defined in Appendix C on page 489.
   b. Class = Map the Reason-Code on the RTI-CALL-FAILURE indication to an exception class as follows:
      — RTI-SERVICE-UNKNOWN = ENV-INVOCATION-FAULT
      — PROTOCOL-VERSION-NOT-SUPPORTED = ENV-INVOCATION-FAULT
      — CONTEXT-TYPE-NOT-SUPPORTED = ENV-INVOCATION-FAULT
      — PERMANENT-COMMUNICATION-FAILURE = ENV-EXECUTION-FAULT
      — TRANSIENT-COMMUNICATION-FAILURE = ENV-EXECUTION-ERROR
      — PROTOCOL-MACHINE-FAILURE = ENV-EXECUTION-FAULT
      — INTERFACE-UNKNOWN = ENV-INVOCATION-FAULT
      — INTERFACE-PERMANENTLY-UNAVAILABLE = ENV-INVOCATION-FAULT
      — INTERFACE-TEMPORARILY-UNAVAILABLE = ENV-INVOCATION-ERROR
      — ROLLBACK-IN-PROGRESS = ENV-EXECUTION-ERROR
      — REASON-NOT-SPECIFIED = ENV-UNSPECIFIED-FAULT
      — RPC-ACCESS-VIOLATION = AP-EXECUTION-FAULT
      — RPC-CANCEL = ENV-EXECUTION-ERROR
      — RPC-FLOATING-DIVIDE-BY-ZERO = AP-EXECUTION-FAULT
      — RPC-FLOATING-ERROR = AP-EXECUTION-FAULT
      — RPC-FLOATING-OVERFLOW = AP-EXECUTION-FAULT

— RPC-FLOATING-UNDERFLOW = AP-EXECUTION-FAULT

— RPC-INSUFFICIENT-RESOURCES = ENV-INVOCATION-ERROR

— RPC-INTEGER-DIVIDE-BY-ZERO = AP-EXECUTION-FAULT

— RPC-INTEGER-OVERFLOW = AP-EXECUTION-FAULT

— RPC-INVALID-OPERATION-NUMBER = AP-INVOCATION-FAULT

— RPC-INVOCATION-FAILURE = ENV-INVOCATION-FAULT

— RPC-MARSHALING-ERROR = AP-INVOCATION-FAULT

— RPC-PROTOCOL-ERROR = ENV-EXECUTION-FAULT

— RPC-REASON-NOT-SPECIFIED = ENV-UNSPECIFIED-FAULT

c. Code = implementation-specific

d. Exception-Code-Group = implementation-specific

e. Source = System

f. Level = Current

g. Exception-Procedure = current task name

h. Exception-Procedure-Group = current task group name

8. A-forward

Issue a ENQUEUE-EXECUTE request with the following parameters:

a. Transactional = the value of V-trans.

b. Task-Forward-Info = derived from the Arguments parameter of the RTI-CALL-TASK indication as described in Section 18.9.1.2 on page 426.

c. Arguments = derived from the Arguments parameter of the RTI-CALL-TASK indication as described in Section 18.9.1.2 on page 426.

9. A-RTI-call

Issue an RTI-CALL-TASK request with the following parameters:

a. Interface-UUID = the value of Task-Group-UUID of V-call-context.

b. Interface-Version-Major = the value of Task-Group-Version-Major of V-call-context.

c. Interface-Version-Minor = the value of Task-Group-Version-Minor of V-call context.

d. Transaction-Attribute = one of the following:

— If V-trans is true, TRANSACTION-MANDATORY.

— If V-trans is false, TRANSACTION-NONE.

e. Operation-Number = from the Task-Operation-Number of the CALL-TASK request or the CALL-TASK-MARSHALED request.

f. Arguments = depends on the request:

— If the request is CALL-TASK, combine the Task-Call-Info parameter with the Arguments parameter to produce the RPC argument list for the server task as described in Section 18.9.1 on page 423 and marshal that argument list.

— If the the request is CALL-TASK-MARSHALED, then the value of the Marshaled-Arguments parameter of the CALL-TASK-MARSHALED request.

10. A-RTI-cancel

    Issue an RTI-CANCEL-CALL request.

11. A-RTI-done

    Issue an RTI-CALL-RESULT request with the following parameters:

    a. Arguments = if V-call is true then derived from the arguments parameter of the TASK-DONE indication as described in Section 18.9.1.1 on page 424, otherwise as described in Section 18.9.1.2 on page 426. In both cases set the EXCEPTION-TYPE field of the EXCEPTION-INFORMATION parameter to 0 (no exception).

12. A-RTI-est-ctx

    Do the following:

    a. Set V-client to true

    b. Set the parts of V-call-context from the parameters of the CALL-TASK, CALL-TASK-MARSHALED or CALL-FORWARD-TASK request as follows:

       — RTI-AP-Title = the value of the RTI-AP-Title parameter.

       — Task-Group-UUID = if V-call is true then from the value of the Task-Group-UUID parameter, otherwise the UUID for the standard task enqueuer RPC interface as described in Section 18.9.1.2 on page 426.

       — Task-Group-Version-Major = if V-call is true then from the value of the Task-Group-Version-Major parameter, otherwise the major version number for the standard task enqueuer RPC interface as described in Section 18.9.1.2 on page 426.

       — Task-Group-Version-Minor = if V-call is true then from the value of the Task-Group-Version-Minor parameter, otherwise the minor version number for the standard task enqueuer RPC interface as described in Section 18.9.1.2 on page 426.

    c. Issue an RTI-ESTABLISH-CONTEXT request with the following parameters:

       — RTI-AP-Title = from the CALL-TASK, CALL-TASK-MARSHALED, or the CALL-FORWARD-TASK request RTI-AP-Title parameter

       — RTI-AE-Qualifier = the string "STDL" (without the quotes).

       — Object-UUID = Null.

       — Client-Name = the principal as described in Section 3.4.1 on page 56.

       — Client-Authenticator-Type = the authentication mechanism type as described in Section 3.4.3 on page 57. 1=default, 2=customer-written.

       — Client-Authenticator = the authentication data as described in Section 3.4.3 on page 57.

       — Interface-UUID = if V-call is true then from the Task-Group-UUID parameter of the CALL-TASK or CALL-TASK-MARSHALED request, otherwise the UUID for the standard task enqueuer RPC interface as described in Section 18.9.1.2 on page 426.

       — Interface-Version-Major = if V-call is true then from the Task-Group-Version-Major parameter of the CALL-TASK or CALL-TASK-MARSHALED request,

otherwise the major version number for the standard task enqueuer RPC interface as described in Section 18.9.1.2 on page 426.

— Interface-Version-Minor = if V-call is true then from the Task-Group-Version-Minor parameter of the CALL-TASK or CALL-TASK-MARSHALED request, otherwise the minor version number for the standard task enqueuer RPC interface as described in Section 18.9.1.2 on page 426.

— Context-Type = one of the following:

— If V-trans is true, TRANSACTION-ENABLED.

— If V-trans is false, NOT-TRANSACTION-ENABLED.

13. A-RTI-exception

    Issue an RTI-CALL-RESULT request with the following parameter:

    a. Arguments = as described in Section 18.9.1.3 on page 430 for reporting exceptions from servers using the exception information from the TASK-EXCEPTION or ENQUEUE-EXCEPTION indication. Exception information and all OUTPUT and INOUT parameters are returned, even though the content of any OUTPUT and INOUT parameters is undefined.

14. A-RTI-forward

    Issue an RTI-CALL-TASK request with the following parameters:

    a. Interface-UUID = the value of Task-Group-UUID of V-call-context.
    b. Interface-Version-Major = the value of Task-Group-Version-Major of V-call-context.
    c. Interface-Version-Minor = the value of Task-Group-Version-Minor of V-call-context.
    d. Transaction-Attribute = one of the following:

       — If V-trans is true, TRANSACTION-MANDATORY.

       — If V-trans is false, TRANSACTION-NONE.

    e. Operation-Number = if V-trans is true then 1, else 0.
    f. Arguments = the arguments as described in Section 18.9.1.2 on page 426.

15. A-RTI-release

    Issue an RTI-RELEASE-CONTEXT request.

16. A-set-call

    Set V-call to true.

17. A-set-local

    Set V-local to true.

18. A-set-trans

    Set V-trans to true.

States

1. S-null

   The initial state.

2. S-local

   A local call is in progress.

3. S-local-cancel

   Waiting for a local call to terminate after a cancel.

4. S-client-call

   A client call to a remote TP system is in progress.

5. S-client-cancel

   Waiting for the server task to terminate after a cancel.

6. S-client-inactive

   The client dialogue to the remote server TP system is inactive.

7. S-n-t-server-call

   A non-transactional server call from a remote system is in progress.

8. S-t-server-call

   A transactional server call from a remote TP system is in progress.

9. S-t-svr-inactive

   A transactional server dialogue from a remote TP system is inactive.

State Transition Tables

	S-null
E-call+P-local+P-trans	S-local A-set-call, A-set-trans, A-set-local, A-call
E-call+P-local+¬P-trans	S-local A-set-call, A-set-local, A-call
E-call+¬P-local+P-trans	S-client-call A-set-call, A-set-trans, A-dist, A-RTI-est-ctx, A-RTI-call
E-call+¬P-local+¬P-trans	S-client-call A-set-call, A-RTI-est-ctx, A-RTI-call
E-forward+¬P-local+P-trans	S-client-call A-set-trans, A-dist, A-RTI-est-ctx, A-RTI-forward
E-forward+¬P-local+¬P-trans	S-client-call A-RTI-est-ctx, A-RTI-forward
E-RTI-call+P-call+P-trans	S-t-server-call A-set-call, A-set-trans, A-call
E-RTI-call+P-call+¬P-trans	S-n-t-server-call A-set-call, A-call
E-RTI-call+¬P-call+P-trans	S-t-server-call A-set-trans, A-forward
E-RTI-call+¬P-call+¬P-trans	S-n-t-server-call A-forward

Table 18-46 Call (Null State)

	S-local	**S-local-cancel**
E-done	S-null A-done	S-null A-canceled
E-exception	S-null A-exception	S-null A-canceled
E-cancel	S-local-cancel A-cancel	
E-canceled		S-null A-canceled

Table 18-47 Call (Local Call States)

	S-client-call	**S-client-cancel**	**S-client-inactive**
E-call			S-client-call A-RTI-call
E-forward			S-client-call A-RTI-forward
E-RTI-result+ P-exception	S-client-inactive A-exception	S-client-inactive A-canceled	
E-RTI-result+ ¬P-exception	S-client-inactive A-done	S-client-inactive A-canceled	
E-RTI-failure+P-trans+ P-failure-fatal	S-null A-failure-exc	S-null A-canceled	
E-RTI-failure+¬P-trans+ P-failure-fatal	S-null A-RTI-release A-failure-exc	S-null A-RTI-release A-canceled	
E-RTI-failure+ ¬P-failure-fatal	S-client-inactive A-failure-exc	S-client-inactive A-canceled	
E-cancel	S-client-cancel A-RTI-cancel		
E-call-release+P-trans			S-null
E-call-release+¬P-trans			S-null A-RTI-release
E-RTI-release			S-null

Table 18-48 Call State Transitions (Client to Remote Server)

	S-n-t-server-call	**S-t-server-call**	**S-t-svr-inactive**
E-RTI-call+ P-call+P-trans			S-t-server-call A-call
E-RTI-call+ ¬P-call+P-trans			S-t-server-call A-forward
E-done	S-null A-RTI-done	S-t-svr-inactive A-RTI-done	
E-exception	S-null A-RTI-exception	S-t-svr-inactive A-RTI-exception	
E-RTI-cancel	S-n-t-server-call A-cancel	S-t-server-call A-cancel	
E-task-release		S-null	S-null

Table 18-49 Call (Server from Remote Client)

18.9.3 Transaction State Machine

The Transaction state machine models the management of transactions within a TP system. This includes the coordination of commitment or rollback of transactions in resource managers in a TP system.

No more than one Transaction state machine instance can exist within an execution context.

Service Primitives Provided

1. TRANS-START — request

   Starts a transaction in the TP system.

   This request has the following parameter:

   a. Type = either Root or Non-Root

2. TRANS-DISTRIBUTED — request

   Tells the Transaction state machine that a transactional client call has been performed in this transaction.

   This request has no parameters.

3. TRANS-COMMIT — request

   Commits a root transaction.

   This request has no parameters.

4. TRANS-ROLLBACK — request

   Rolls back a transaction.

   This request has no parameters.

5. TRANS-COMMIT-DONE — indication

   Reports that the transaction has been committed = only issued for root transactions.

   This indication has no parameters.

6. TRANS-ROLLBACK-RECEIVED — indication

   Reports that a rollback request from the RTI protocol machine has been received.

   This indication has no parameters.

7. TRANS-ROLLBACK-DONE — indication

   Reports that the transaction has been rolled back.

   This indication has no parameters.

Service User

The service user of the Transaction state machine for most service primitives is the state machine instance that issued the TRANS-START request. All indications from the Transaction state machine instance created by the TRANS-START request are directed to that service user. The TRANS-COMMIT and TRANS-ROLLBACK requests from that service user are directed to the Transaction state machine instance created for that service user.

The TRANS-DISTRIBUTED request is always issued by the Call state machine. However, the Call state machine never receives any indications from the Transaction state machine.

State Machines Used

1. Call (CALL-)
2. Remote Task Invocation Protocol Machine (RTI-)
3. Resource Manager (RM-)

Variables

None.

Predicates

None.

Events

1. E-commit

   A TRANS-COMMIT request received.

2. E-dist-trans

   A TRANS-DISTRIBUTED request received.

3. E-RM-P1-Fail

   An RM-P1-FAIL indication received from any Resource Manager state machine instance and all other Resource Manager state machine instances in this execution context have sent either RM-P1-FAIL or RM-P1-DONE indications.

4. E-RM-P1-OK

   An RM-P1-DONE indication received from all local Resource Manager state machine instances in this execution context.

5. E-RM-P2

   An RM-P2-DONE indication received from all local Resource Manager state machine instances in this execution context.

6. E-RM-rollback

   An RM-ROLLBACK-DONE indication received from all local Resource Manager state machine instances in this execution context.

7. E-rollback

   A TRANS-ROLLBACK request received.

8. E-RTI-commit

   An RTI-COMMIT-TRANS indication received.

9. E-RTI-complete

   An RTI-TRANS-COMPLETE indication received.

10. E-RTI-heuristic

    An RTI-HEURISTIC-REPORT indication received.

11. E-RTI-prepare

    An RTI-PREPARE-TRANS indication received.

12. E-RTI-ready

    An RTI-TRANS-READY indication received.

13. E-RTI-rollback

    An RTI-ROLLBACK-TRANS indication received.

14. E-start-root

    A TRANS-START request received with Type = Root.

15. E-start-non-root

    A TRANS-START request received with Type = Non-Root.

Actions

1. A-done-c

   Issue a TRANS-COMMIT-DONE indication.

2. A-done-r

   Issue a TRANS-ROLLBACK-DONE indication.

3. A-heuristic

   Report the heuristic report in an implementation-specific manner.

4. A-report-rollback

   Issue a TRANS-ROLLBACK-RECEIVED indication.

5. A-release

   Issue a CALL-RELEASE-CONTEXT request to each Call state machine instance in the execution context with V-trans true and V-client-true.

6. A-RM-P1

   Issue an RM-DO-P1 request to each Resource Manager state machine instance in the execution context.

7. A-RM-P2

   Issue an RM-DO-P2 request to each Resource Manager state machine instance in the execution context.

8. A-RM-rollback

   Issue an RM-ROLLBACK request to each Resource Manager state machine instance in the execution context.

9. A-RTI-commit

   Issue an RTI-COMMIT-TRANS request.

10. A-RTI-done

    Issue an RTI-TRANS-DONE request.

11. A-RTI-end

    Issue an RTI-END-TRANS request.

12. A-RTI-ready

    Issue an RTI-TRANS-READY request.

13. A-RTI-rollback

    Issue an RTI-ROLLBACK-TRANS request.

States

1. S-null

   The null state.

2. S-local

   A local-only transaction is active.

3. S-root

   A distributed transaction is active and this execution context is the root of the transaction tree.

4. S-branch

   A distributed transaction is active and this execution context is not the root of the transaction tree.

5. S-dist-rollb-pending

   Waiting for the task to finish processing before rolling the transaction back.

6. S-local-P1

   Resource managers are performing phase 1 commit work for a local transaction.

7. S-local-P2

   Resource managers are performing phase 2 commit work for a local transaction.

8. S-local-rollb

   Resource managers are performing rollback work for a local transaction.

9. S-branch-dist-P1

   Resource and communication managers are performing phase 1 commit work for a distributed transaction, and this execution context is not the root of the transaction tree.

10. S-root-dist-P1

    Resource and communication managers are performing phase 1 commit work for a distributed transaction, and this execution context is the root of the transaction tree.

11. S-root-dist-wait-cm

    The communication resource manager is performing phase 1 commit work for a distributed transaction, and this execution context is the root of the transaction tree.

12. S-root-dist-wait-rm

    Local resource managers are performing phase 1 commit work for a distributed transaction, and this execution context is the root of the transaction tree.

13. S-dist-wait-P2

    Waiting for RTI to indicate whether to do phase 2 commit work or to rollback the transaction.

14. S-dist-P2

    Resource managers are performing phase 2 commit work for a distributed transaction.

15. S-dist-forget-c

    Waiting for RTI to indicate that the transaction is done for a committed transaction.

16. S-dist-rollb

    Resource managers are performing rollback work for a distributed transaction.

17. S-dist-forget-r

    Waiting for RTI to indicate that the transaction is done for a rolled back transaction.

State Transition Tables

	S-null
E-start-root	S-local
E-start-non-root	S-branch

Table 18-50 Transaction (Null State)

	S-local	S-root	S-branch	S-dist-rollb-pending
E-dist-trans	S-root	S-root		
E-commit	S-local-P1 A-RM-P1	S-root-dist-P1 A-RTI-end, A-RM-P1		
E-RTI-prepare			S-branch-dist-P1 A-RM-P1	
E-rollback	S-local-rollb A-RM-rollback	S-dist-rollb A-RTI-rollback A-RM-rollback		S-dist-rollb A-RM-rollback
E-RTI-rollback		S-dist-rollb-pending A-report-rollback	S-dist-rollb-pending A-report-rollback	

Table 18-51 Transaction (Active States)

	S-local-P1	S-local-P2	S-local-rollb
E-RM-P1-OK	S-local-P2 A-RM-P2		
E-RM-P1-Fail	S-local-rollb A-RM-rollback		
E-RM-P2		S-null A-release, A-done-c	
E-RM-rollback			S-null A-release, A-done-r

Table 18-52 Transaction (Local Termination States)

	S-root-dist-P1	S-root-dist-wait-cm	S-root-dist-wait-rm	S-branch-dist-P1
E-RM-P1-OK	S-root-dist-wait-cm		S-dist-wait-P2 A-RTI-commit	S-dist-wait-P2 A-RTI-ready
E-RM-P1-Fail	S-dist-rollb A-RTI-rollback A-RM-rollback		S-dist-rollb A-RTI-rollback A-RM-rollback	S-dist-rollb A-RTI-rollback A-RM-rollback
E-RTI-ready	S-root-dist-wait-rm	S-dist-wait-P2 A-RTI-commit		
E-RTI-rollback	S-dist-rollb A-RM-rollback	S-dist-rollb A-RM-rollback	S-dist-rollb A-RM-rollback	S-dist-rollb A-RM-rollback

Table 18-53 Transaction (Distributed P1 Commit States)

	S-dist-wait-P2	**S-dist-P2**	**S-dist-forget-c**
E-RTI-commit	S-dist-P2 A-RM-P2		
E-RM-P2		S-dist-forget-c A-RTI-done	
E-RTI-rollback	S-dist-rollb A-RM-rollback		
E-RTI-complete			S-null A-release, A-done-c
E-RTI-heuristic		S-dist-P2 A-heuristic	S-dist-forget-c A-heuristic

Table 18-54 Transaction (Distributed P2 Commit States)

	S-dist-rollb	**S-dist-forget-r**
E-RM-rollback	S-dist-forget-r A-RTI-done	
E-RTI-complete		S-null A-release, A-done-r
E-RTI-heuristic	S-dist-rollb A-heuristic	S-dist-forget-r A-heuristic

Table 18-55 Transaction (Distributed Rollback States)

18.10 Resource Manager State Machines

18.10.1 Transactional Receive State Machine

The Transactional Receive state machine models the execution of transactional receives.

No more than one Transactional Receive state machine instance can exist within an execution context.

Service Primitives Provided

1. RR-RECEIVE — request

   Executes a transactional receive.

   This request has no parameters.

2. RR-DONE — indication

   Reports that the the transactional receive successfully completed.

   This indication has one parameter:

   a. Data

3. RR-EXCEPTION — indication

   Reports that the transactional receive resulted in an exception.

   This indication has the following parameters:

   a. Type = Non-transaction, Transient Transaction, Permanent Transaction or Fatal Transaction

   b. Class = one of the classes defined in Appendix C on page 489.

   c. Code = the exception code as defined in Section 3.7.2 on page 62.

   d. Exception-Code-Group = the exception code group as defined in Section 3.7.2 on page 62.

   e. Source = System or Application

   f. Level = Current or Propagated

   g. Exception-Procedure = the procedure in which the exception occurred as defined in Section 3.7.2 on page 62.

   h. Exception-Procedure-Group = the procedure group in which the exception occurred as defined in Section 3.7.2 on page 62.

4. RR-RESET — request

   Resets the transactional receive state machine instance so that the next RR-RECEIVE request calls a presentation procedure to get new data.

   This request has no parameters.

5. RR-CANCEL — request

   Cancels the execution of the transactional receive.

   This request has no parameters.

6. RR-CANCELED — indication

Indicates that the transactional receive execution has been canceled.

This indication has no parameters.

Service User

The service user of a Transactional Receive state machine is the state machine instance that issued the RR-RECEIVE request. The RR-DONE and RR-EXCEPTION indications are directed to that service user.

State Machines Used

1. Presentation Procedure (PRES-)

Variables

1. V-current-data

   This is the data for the transactional receive in the current transaction, if any.

   This variable stores data.

Predicates

None.

Events

1. E-cancel

   An RR-CANCEL request received.

2. E-canceled

   A PRES-CANCELED indication received.

3. E-done

   A PRES-DONE indication received.

4. E-execute

   An RR-RECEIVE request received.

5. E-exception

   A PRES-EXCEPTION indication received.

6. E-reset

   An RR-RESET request received.

Actions

1. A-cancel

   Issue a PRES-CANCEL request.

2. A-canceled

   Issue an RR-CANCELED indication.

3. A-done

Issue an RR-DONE indication using V-current-data.

4. A-exception

   Return an RR-EXCEPTION indication with the following parameters:

   a. Type = from indication
   b. Class = from indication
   c. Code = from indication
   d. Exception-Code-Group = from indication
   e. Source = from indication
   f. Level = from indication
   g. Exception-Procedure = from indication
   h. Exception-Procedure-Group = from indication

5. A-execute

   Issue a PRES-EXECUTE request.

6. A-save

   Save the results of the PRES-DONE indication in V-current-data.

States

1. S-null

   The initial state.

2. S-execute

   Waiting for a presentation procedure to complete.

3. S-inactive

   Have saved receive data, waiting for either a reset or the next execution.

4. S-cancel

   Waiting for the presentation procedure to return PRES-CANCELED.

State Transition Tables

	S-null	**S-execute**	**S-inactive**	**S-cancel**
E-execute	S-execute A-execute		S-inactive A-done	
E-done		S-inactive A-save A-done		
E-exception		S-null A-exception		
E-cancel		S-cancel A-cancel		
E-canceled				S-null A-canceled
E-reset	S-null		S-null	

Table 18-56 Transactional Receive

18.10.2 Resource Manager State Machine

The Resource Manager state machine models the processing done to access a transactional resource manager.

Multiple Resource Manager state machine instances can exist within an execution context. Each request (other than those issued by the Transaction state machine) from any state machine instance within an execution context for a separate resource type creates a separate Resource Manager state machine instance. The indications generated by a Resource Manager state machine instance are returned to the issuer of the last request. The RM-CANCEL request from a service user are directed to the Resource Manager state machine instance being used by that service user.

Service Primitives Provided

The service primitives for the Resource Manager state machine are divided into three categories:

1. Resource manager specific
2. Resource manager general
3. Transaction service primitives

These are listed below.

1. Resource manager specific service primitives:
   a. Queued Task:
      - RM-ENQUEUE-TASK — request

        Enqueues a submitted task request.

        This request has the following parameters:

        - Local-Client = is a Boolean that is true if the TP system on which the request is issued is the client TP system.
        - Local-Server = is a Boolean that is true if the server TP system is the same as the TP system on which this request is issued.
        - Client-RTI-AP-Title = is the RTI-AP-Title of the Task Enqueuer on the Client TP system. This is used only if the Local-Client parameter is false.
        - Submission-Information-Record = is the TASK-FORWARD-INFORMATION record for the submitted task request.
        - Marshaled-Arguments = is the marshaled arguments as described in Section 18.9.1.1 on page 424 for the submitted task request. These arguments are marshaled on the submitter TP system.
      - RM-ENQUEUE-TASK — indication

        Reports that the task enqueue request successfully completed.

        This indication has the following parameter:

        - Entry-ID = the ID of the task submission request
      - RM-DEQUEUE-TASK — request

        Dequeues a submitted task request

This request has one parameter:

— Submission Client = a Boolean value

True The client TP system for the submitted task request is the current TP system. This also indicates that only those submitted task requests with a trigger that allows the task to be called are to be returned.

False The client TP system is a TP system other than the current TP system.

— RM-DEQUEUE-TASK — indication

Reports that the Dequeue a submitted task request successfully completed

This indication has the following parameters:

— Local-Client = is a Boolean that is true if the submitter TP system is the client TP system for the submitted task request.

— Local-Server = is a Boolean that is true if the server TP system is the same as the TP system on which this request is issued.

— Client-RTI-AP-Title = is the RTI-AP-Title of the Task Enqueuer on the Client TP system. This is used only if the Local-Client parameter is false.

— Submission-Information-Record = is the TASK-FORWARD-INFORMATION record for the submitted task request.

— Marshaled-Arguments = is the marshaled arguments as described in Section 18.9.1.1 on page 424 for the submitted task request. These arguments are marshaled on the submitter TP system.

— RM-REMOVE-TASK — request

Removes a submitted task request.

This request has one parameter:

— Entry-Id = the ID of the task submission request entry to remove

b. Queued record:

— RM-ENQUEUE-RECORD — request

Enqueues a data record.

This request has a single parameter:

— Data

— RM-ENQUEUE-RECORD — indication

Reports that the data record enqueue has successfully completed.

This indication has the following parameter:

— Entry-ID = the ID of the record queued

— RM-DEQUEUE-RECORD-FIRST — request

Dequeues the first data record available in the record queue.

This request has no parameters.

— RM-DEQUEUE-RECORD-KEY — request

Dequeues the first data record available in the record queue with a specified key.

This request has the following parameter:

— Key = the key of the record to dequeue

— RM-DEQUEUE-RECORD-ID — request

Dequeues a specific data record.

This request has the following parameter:

— Entry-ID = the ID of the data record

— RM-DEQUEUE-RECORD — indication

Reports that a record dequeue is successful.

This indication has the following parameter:

— Data

— RM-READ-QUEUE-RECORD-FIRST — request

Reads the first data record available in the record queue.

This request has no parameters.

— RM-READ-QUEUE-RECORD-NEXT — request

Reads the next data record available in the record queue.

This request has no parameters.

— RM-READ-QUEUE-RECORD-ID — request

Reads a specific data record.

This request has the following parameter:

— Entry-ID = the ID of the data record

— RM-READ-QUEUE-RECORD-KEY-FIRST — request

Reads the first available data record in the queue with a specified key.

This request has the following parameter:

— Key = the key specified in the <dequeue-key-value> of the <dequeue-record>.

— RM-READ-QUEUE-RECORD-KEY-NEXT — request

Reads the next available data record in the queue with a previously specified key.

This request has no parameters.

— RM-READ-QUEUE-RECORD — indication

Reports that a record read is successful.

This indication has the following parameter:

— Data

c. Transactional send:

— RM-TRANSACTIONAL-SEND — request

Sends a transactional message.

This request has the following parameter:

— Data

d. SQL and file access:

— RM-DO-IO — request

Accesses file or an SQL database.

This request has no parameters (for this model).

2. General resource manager service primitives:

a. RM-CANCEL — request

Cancels a resource manager operation (other than commit processing).

This request has no parameters.

b. RM-CANCELED — indication

Indicates that the resource manager operation has been canceled.

This indication has no parameters.

c. RM-DONE — indication

Reports that the operation is done. This indication is used when the resource manager operation does not return data.

This indication has no parameters.

d. RM-EXCEPTION — indication

Reports the the operation resulted in an exception. This indication has the following parameters:

— Type = Non-transaction, Transient Transaction, Permanent Transaction or Fatal Transaction

— Class = one of the classes defined in Appendix C on page 489.

— Code = implementation-specific

— Exception-Code-Group = implementation-specific

— Source = System or Application

— Level = Current or Propagated

3. Service primitives used by the Transaction state machine:

a. RM-DO-P1 — request

Performs phase 1 of commit process.

This request has no parameters.

b. RM-DO-P2 — request

Performs phase 2 of commit process.

This request has no parameters.

c. RM-ROLLBACK — request

Undoes the transactional operations done by this resource manager in this transaction.

This request has no parameters.

d. RM-ROLLBACK-DONE — indication

Reports that a rollback of the transactional operations are done.

This indication has no parameters.

e. RM-P1-DONE — indication

Reports that phase 1 of commit successfully finished.

This indication has no parameters.

f. RM-P1-FAIL — indication

Reports that phase 1 of commit failed.

This indication has no parameters.

g. RM-P2-DONE — indication

Reports that phase 2 of commit has successfully finished.

This indication has no parameters.

Service User

The service user for the Resource Manager state machine depends on the type of service primitive:

1. Resource manager specific and general resource manager service primitives:

   For these service primitives, the service user for a Resource Manager state machine instance is the state machine instance that last issued a request to the Resource Manager state machine. The indications generated by a Resource Manager state machine instance are directed to this service user.

2. Transactional service primitives:

   For these service primitives, the service user for a Resource Manager state machine instance is the Transaction state machine instance in the execution context.

State Machines Used

1. Presentation Procedure (PRES-)

   The Transactional Send resource manager uses the Presentation Procedure state machine to perform transactional sends sometime after the transactional send operation has been committed. However, this detailed operation of the Transactional Send resource manager is not shown in the Resource Manager state machine presented in this section.

Variables

None.

Predicates

1. P-trans

   Is true if the exception is transactional.

Events

1. E-cancel

   An RM-CANCEL request received.

2. E-do-P1

   An RM-DO-P1 request received.

3. E-do-P2

   An RM-DO-P2 request received.

4. E-done

   The transactional operation successfully finished.

5. E-exception

   The transactional operation produced an error.

6. E-execute

   Received one of the following:

   a. an RM-ENQUEUE-TASK request
   b. an RM-DEQUEUE-TASK request
   c. an RM-REMOVE-TASK request
   d. an RM-ENQUEUE-RECORD request
   e. an RM-DEQUEUE-RECORD-FIRST request
   f. an RM-DEQUEUE-RECORD-KEY request
   g. an RM-DEQUEUE-RECORD-ID request
   h. an RM-READ-QUEUE-RECORD-FIRST request
   i. an RM-READ-QUEUE-RECORD-NEXT request
   j. an RM-READ-QUEUE-RECORD-ID request
   k. an RM-READ-QUEUE-RECORD-KEY-FIRST request
   l. an RM-READ-QUEUE-RECORD-KEY-NEXT request
   m. an RM-DEQUEUE-RECORD request
   n. an RM-READ-QUEUE-RECORD request
   o. an RM-TRANSACTIONAL-SEND request
   p. an RM-DO-IO request.

7. E-P1-done

   The resource manager successfully finished phase 1 commit processing. The transactional resource can now either be committed or rolled back.

8. E-P1-fail

   The resource manager encountered an error in phase 1 commit processing. The transactional resource cannot be committed. However, the transactional resource can be rolled back.

9. E-P2-done

   The resource manager finished phase 2 commit processing. The transactional resource is now in a stable state with the transactional operations performed in this transaction made permanent.

10. E-rollback

    An RM-ROLLBACK request received.

11. E-rollback-done

    The resource manager has finished rolling back the transactional operations performed in this transaction.

Actions

1. A-cancel

   Cancel the current operation and issue an RM-CANCELED indication.

2. A-done

   Issue one of the following indications depending on V-operation:

   a. S-enqueue-task

      RM-DONE indication.

   b. S-dequeue-task

      RM-DEQUEUE-TASK indication.

   c. S-remove-task

      RM-DONE indication.

   d. S-enqueue-record

      RM-DONE indication.

   e. S-dequeue-record

      RM-DEQUEUE-RECORD indication.

   f. S-read-queue-record

      RM-READ-QUEUE-RECORD indication.

   g. S-transactional-send

      RM-DONE indication.

   h. S-i-o

RM-DONE indication.

3. A-do-P1

   Perform phase 1 commit processing for the resource manager. The resource manager attempts to put the transactional resource in a state where the transactional operations can be either committed or rolled back.

   This generates one of the following internal events:

   a. E-P1-done

   b. E-P1-fail

4. A-do-P2

   Perform phase 2 commit processing for the resource manager. The resource manager makes the transactional operations done during the transaction being committed permanent.

   This generates a E-P2-done internal event.

5. A-exception

   Issue an RM-EXCEPTION indication with the following parameters:

   a. Type = Derived from the Class field as defined in Appendix C on page 489.

   b. Class = One of the following:

      — AP-INVOCATION-FAULT (transactional sends only)

      — ENV-INVOCATION-FAULT

      — ENV-INVOCATION-ERROR (transactional sends only)

      — AP-EXECUTION-FAULT

      — ENV-EXECUTION-FAULT

      — REQUEST-TIMEOUT-ERROR (dequeue record and read queue record only)

      — SYSTEM-SHUTDOWN-FAULT (dequeue record, read queue record and dequeue task only)

      — NO-OUTPUT-ERROR (dequeue record and read queue record only)

   c. Code = implementation-specific

   d. Exception-Code-Group = implementation-specific

   e. Source = System

   f. Level = Current

6. A-execute

   Perform the transactional operation.

   This generates one of the following internal events:

   a. E-done

   b. E-exception

7. A-P1-done

Issue an RM-P1-DONE indication.

8. A-P1-fail

   Issue an RM-P1-FAIL indication.

9. A-P2-done

   Issue an RM-P2-DONE indication.

10. A-rollback

    Roll back the transactional operations done by this resource manager in this transaction.

    This generates a E-rollback-done internal event.

11. A-rollback-done

    Issue an RM-ROLLBACK-DONE indication.

States

1. S-null

   The initial state.

2. S-operating

   Waiting for a resource manager operation to complete.

3. S-inactive

   Waiting for either the transaction to complete or another operation to perform.

4. S-do-rollback

   Waiting for the rollback within the resource manager to complete.

5. S-do-P1

   Waiting for phase 1 commit processing within the resource manager to complete.

6. S-wait-rollback

   Waiting for the transaction to be rolled back.

7. S-wait-P2

   Waiting for the either the transaction to be committed or rolled back.

8. S-do-P2

   Waiting for phase 2 commit processing within the resource manager to complete.

State Transition Tables

	S-null	**S-operating**	**S-inactive**
E-execute	S-operating A-execute		S-operating A-execute
E-cancel		S-wait-rollback A-cancel	
E-done		S-inactive A-done	
E-exception+ ¬P-trans		S-inactive A-exception	
E-exception+ P-trans		S-wait-rollback A-exception	
E-do-P1			S-do-P1 A-do-P1
E-rollback			S-do-rollback A-rollback

Table 18-57 Resource Manager (Null and Operating States)

	S-do-P1	**S-wait-P2**	**S-do-P2**	**S-wait-rollback**	**S-do-rollback**
E-rollback	S-do-rollback A-rollback	S-do-rollback A-rollback		S-do-rollback A-rollback	
E-P1-done	S-wait-P2 A-P1-done				
E-P1-fail	S-wait-rollback A-P1-fail				
E-do-P2		S-do-P2 A-do-P2			
E-P2-done			S-null A-P2-done		
E-rollback-done					S-null A-rollback-done

Table 18-58 Resource Manager (Commit and Rollback States)

18.11 Procedure Execution State Machines

The state machines in this section handle the execution of presentation and processing procedures.

18.11.1 Processing Procedure State Machine

The Processing Procedure state machine models the execution of processing procedures.

Multiple Processing Procedure state machine instances can exist within an execution context. A Processing Procedure state machine instance is created for each top-level processing procedure being executed by tasks within an execution context.

While this state machine models all of the processing for processing procedures, it does not explicitly model calls between processing procedures.

Service Primitives Provided

1. PROC-EXECUTE — request

   Executes a top-level processing procedure.

   Arguments = the INPUT and INOUT arguments as defined for the processing procedure in the processing group specification.

2. PROC-CANCEL — request

   Cancels the processing procedure execution.

   This request has no parameters.

3. PROC-CANCELED — indication

   Indicates that the processing procedure execution has been canceled.

   This indication has no parameters.

4. PROC-DONE — indication

   Reports that the top-level processing procedure execution successfully completed.

   Arguments = the OUTPUT and INOUT arguments as defined for the processing procedure in the processing group specification.

5. PROC-EXCEPTION — indication

   Reports that the processing procedure execution resulted in an exception.

   This indication has the following parameters:

   a. Type = Non-transaction, Transient Transaction, Permanent Transaction or Fatal Transaction

   b. Class = one of the classes defined in Appendix C on page 489.

   c. Code = the exception code as defined in Section 3.7.2 on page 62.

   d. Exception-Code-Group = the exception code group as defined in Section 3.7.2 on page 62.

   e. Source = System or Application

   f. Level = Current or Propagated

g. Exception-Procedure = the procedure in which the exception occurred as defined in Section 3.7.2 on page 62.

h. Exception-Procedure-Group = the procedure group in which the exception occurred as defined in Section 3.7.2 on page 62.

Service User

The service user for a Processing Procedure state machine instance is the state machine instance which issued the PROC-EXECUTE request to the Processing Procedure state machine. The indications generated by the Processing Procedure state machine instance are directed to this service user.

State Machines Used

1. Resource Manager (RM-)

   The processing procedure invokes resource managers each time a SQL database is accessed or a file is accessed.

2. Call (CALL-)

   The processing procedure state machine invokes a Call state machine each time a client stub is executed.

Variables

None.

Predicates

1. P-trans

   Is true if the exception has a type of transient transaction, permanent transaction or fatal transaction.

Events

1. E-call

   The called procedure called a client stub.

2. E-cancel

   A PROC-CANCEL request received.

3. E-canceled-c

   A CALL-CANCELED indication received.

4. E-canceled-io

   A RM-CANCELED indication received.

5. E-exception-c

   A CALL-EXCEPTION indication received.

6. E-exception-i

   The called procedure produces an exception or the exception variables are set.

7. E-exception-io

   A RM-EXCEPTION indication received.

8. E-execute

   A PROC-EXECUTE request received.

9. E-done

   The called procedure completes without an exception and the exception information variables are not set.

10. E-done-c

    A CALL-DONE indication received.

11. E-done-io

    An RM-DONE indication received.

12. E-io

    The processing procedure has encountered a file or database statement.

Actions

1. A-call

   Issue a CALL-TASK request with the following parameters:

   a. Local = true if the server task is in the task group of the task that called the top-level processing procedure, otherwise false

      An implementation can set this true if the TP system for the server task is the same TP system on which the processing procedure is executing.

   b. RTI-AP-Title = one of the following

      — If local=true, omitted

      — If local=false, the RTI-AP-Title from a task group entry in a destination entry. The implementation can choose to either determine the destination entry from the server task group name or use a default destination entry for the TP system. The client TP system chooses the task group entry within the destination by matching the server task group identification and determining whether or not the version number is compatible, as described in Section 3.3.1 on page 47 and Section 17.2.9 on page 306. An implementation can choose to use either the task group name or UUID as the task group identification.

   c. Task-Group-UUID = from the server <task-group-specification>

   d. Task-Group-Version = from the server <task-group-specification>

   e. Task-Operation-Value = the position of the server task's <task-interface> within the server <task-group-specification> as described in Section 18.9.1.1 on page 424.

   f. Transactional = from the server task's <task-interface>

   g. Task-Call-Info = the information listed in Section 18.9.1.1 on page 424.

   h. Arguments = the INPUT and INOUT arguments as defined in the server task <task-group-specification>

2. A-cancel-c

   Issue a CALL-CANCEL request.

3. A-cancel-io

   Issue an RM-CANCEL request.

4. A-canceled

   Cancel the processing procedure and issue the PROC-CANCELED indication.

5. A-continue

   Set the external exception variables to zero and continue the execution of the procedure that called the client stub.

   This generates one of the following internal events:

   a. E-call
   b. E-done
   c. E-io
   d. E-exception-i

6. A-continue-e

   Set the external exception variables to the exception received and continue the execution of the procedure that called the client stub.

   This generates one of the following internal events:

   a. E-call
   b. E-done
   c. E-io
   d. E-exception-i

7. A-done

   Issue a PROC-DONE indication.

8. A-exception

   Issue a PROC-EXCEPTION indication with the following parameters:

   a. Type = from exception class
   b. Class = from exception or exception information variable
   c. Code = from exception or exception information variable
   d. Exception-Code-Group = from exception or exception information variable
   e. Source = System if caused by an exception, or Application if caused by exception variable being set
   f. Level = Propagated
   g. Exception-Procedure = implementation specific if caused by an exception, or from the exception information variable if caused by the exception variable being set
   h. Exception-Procedure-Group = implementation specific if caused by an exception, or from the exception information variable if caused by the exception variable being set

9. A-exception-i

   Issue a PROC-EXCEPTION indication with the information from the indication received.

10. A-execute

    Call the requested top-level processing procedure.

    This generates one of the following internal events:

    a. E-call
    b. E-done
    c. E-io
    d. E-exception-i

11. A-io

    Issue an RM-DO-IO request.

States

1. S-null

   The initial state.

2. S-execute

   Waiting for the processing to complete.

3. S-call

   Waiting for a client stub to complete.

4. S-call-c

   Waiting for a client stub to complete after a cancel.

5. S-io

   Waiting for a file or database operation to complete.

6. S-io-c

   Waiting for a file or database operation to complete after a cancel.

State Transition Tables

	S-null	**S-execute**	**S-call**	**S-call-c**	**S-io**	**S-io-c**
E-execute	S-execute A-execute					
E-call		S-call A-call				
E-done-c			S-execute A-continue			
E-exception-c+ ¬P-trans			S-execute A-continue-e			
E-exception-c+ P-trans			S-null A-exception-i			
E-io		S-io A-io				
E-done-io					S-execute A-continue	
E-exception-io+ ¬P-trans					S-execute A-continue-e	
E-exception-io+ P-trans					(1)	
E-done		S-null A-done				
E-exception-i		S-null A-exception				
E-cancel		S-null A-canceled	S-call-c A-cancel-c		S-io-c A-cancel-io	
E-canceled-c				S-null A-canceled		
E-canceled-io						S-null A-canceled

Table 18-59 Processing Procedure

Notes:

1. An implementation can implement either S-null/A-exception-i or S-execute/A-continue-e.

18.11.2 Presentation Procedure State Machine

The Presentation Procedure state machine models the execution of presentation procedures including initialization and termination procedures.

Multiple Presentation Procedure state machine instances can exist within an execution context. One Presentation Procedure state machine instance is created for each presentation group used by tasks in the execution context. It is created when the presentation group receives its first PRES-EXECUTE request in that execution context.

Service Primitives Provided

1. PRES-EXECUTE — request

   Executes a presentation procedure.

   This request has one parameter:

   a. Data

2. PRES-CANCEL — request

   Cancels the execution of a presentation procedure.

   This request has no parameters.

3. PRES-CANCELED — indication

   Indicates that the presentation procedure execution has been canceled.

   This indication has no parameters.

4. PRES-DONE — indication

   Reports that the presentation procedure execution successfully completed.

   This indication has one parameter:

   a. Data

5. PRES-EXCEPTION — indication

   Reports that the presentation procedure execution resulted in an exception.

   This indication has the following parameters:

   a. Type = Non-transaction, Transient Transaction, Permanent Transaction or Fatal Transaction

   b. Class = one of the classes defined in Appendix C on page 489.

   c. Code = the exception code as defined in Section 3.7.2 on page 62.

   d. Exception-Code-Group = the exception code group as defined in Section 3.7.2 on page 62.

   e. Source = System or Application

   f. Level = Current or Propagated

   g. Exception-Procedure = the procedure in which the exception occurred as defined in Section 3.7.2 on page 62.

   h. Exception-Procedure-Group = the procedure group in which the exception occurred as defined in Section 3.7.2 on page 62.

6. PRES-TERMINATE — request

   Terminates the presentation group context.

   This request has no parameters.

Service User

The service user for a Presentation Procedure state machine instance is the last state machine instance to issue a request to the Presentation Procedure state machine. The indications generated by the Presentation Procedure state machine instance are directed to this service user.

State Machines Used

None.

Variables

None.

Predicates

None.

Events

1. E-cancel

   A PRES-CANCEL request received.

2. E-done

   The presentation procedure completes successfully.

3. E-exception

   An exception occurs during the execution of the presentation procedure or the presentation procedure sets the exception information variables.

4. E-execute

   A PRES-EXECUTE request received.

5. E-init-done

   The initialization procedure completes successfully.

6. E-init-exception

   An exception occurs during the execution of the initialization procedure or the initialization procedure sets the exception information variables.

7. E-terminate

   A PRES-TERMINATE request received.

8. E-terminate-done

   The termination procedure completes = either successfully or with an exception.

Actions

1. A-cancel

   Cancel the presentation procedure execution and issue a PRES-CANCELED indication.

2. A-cancel-init

   Cancel the initialization procedure execution and issue a PRES-CANCELED indication.

3. A-done

   Issue a PRES-DONE indication.

4. A-exception

   Issue a PRES-EXCEPTION indication with the following parameters:

   a. Type = from the Class as defined in Appendix C on page 489.

   b. Class = from exception or exception information variable

   c. Code = from exception or exception information variable

   d. Exception-Code-Group = from exception or exception information variable.

   e. Source = system if caused by an exception, or application if the exception information was set

   f. Level = Propagated

   g. Exception-Procedure = implementation-specific if caused by an exception, or from the exception information variable if caused by the exception variable being set

   h. Exception-Procedure-Group = implementation-specific if caused by an exception, or from the exception information variable if caused by the exception variable being set

5. A-execute

   Call the presentation procedure.

   This generates one of the following internal events:

   a. E-done

   b. E-exception

6. A-init

   Call the initialization procedure.

   This generates one of the following internal events:

   a. E-init-done

   b. E-init-exception

7. A-terminate

   Call the termination procedure.

   This generates a E-terminate-done internal event.

States

1. S-null

   The initial state.

2. S-init

   Waiting for the initialization procedure to complete.

3. S-execute

   Waiting for the presentation procedure to complete.

4. S-inactive

   Waiting for the next presentation procedure or termination procedure to execute after presentation context has been established.

5. S-terminate

   Waiting for the termination procedure to complete.

State Transition Tables

	S-null	**S-init**	**S-execute**	**S-inactive**	**S-terminate**
E-execute	S-init A-init			S-execute A-execute	
E-cancel		S-null A-cancel-init	(1)		
E-init-done		S-execute A-execute			
E-init-exception		S-null A-exception			
E-done			S-inactive A-done		
E-exception			S-inactive A-exception		
E-terminate				S-terminate A-terminate	
E-terminate-done					S-null

Table 18-60 Presentation Procedure

Notes:

1. An implementation can implement either S-null/A-cancel or S-inactive/A-cancel.

Structured Transaction Definition Language (STDL)

Part 3:

Appendices

X/Open Company Ltd.

Appendix A

Architectural Constants

This appendix is normative.

This appendix defines the constants that must be the same in every STDL implementation. Currently these constants cover the following:

- architectural limits
- exception class values.

These are discussed in the following sections.

A.1 Architectural Limits

In order for an implementation to be compliant to this specification and to allow applications to be portable from one compliant system to another, the following constants have been defined. The following tables contain the architecturally required minimum limits that an implementation must support. An implementation can support, unless otherwise specified, higher limits, but must support at least the limits shown in the tables. An implementation must support the limits simultaneously.

Table A-1 on page 482 shows the required minimum limits for TP system resources. (1KB equals 1024 bytes. 1MB equals 1024KB.)

	Profile B	**Profile A**
Private workspace pool size per task	511KB	255KB
Shared workspace pool size per task group	4095KB	1023KB
Max size for single private workspace	63KB	63KB
Max size for single shared workspace	63KB	63KB
Shared workspaces per task group	1023	127
Total workspace size per <call-presentation-procedure> statement	30KB	30KB
Record queues per TP system	31	7
Entries per task queue	32767	255
Entries per record queue	32767	127
Max size of all non-durable record queues per TP system	6MB	1MB
Broadcast lists per TP system	1023	0
Display names per broadcast list	10229	0
Distribution lists per TP system	1023	255
Destination names per distribution list	10229	255
Max size of an audit record that can be written to the audit log	30000 bytes	30000 bytes
Text characters in a broadcast send (Simple Latin, ISO Latin-1 or Katakana)	256	256
Text characters in a broadcast send (Kanji)	128	128
Destinations per TP system	32767	32767
Incoming authentications per TP system	32767	32767
Task groups per TP system	63	31

Table A-1 TP System Resource Minimum Limits

Table A-2 on page 483 shows the required minimum limits for transaction processing language translation.

	B	A
Text characters in a string literal (Simple Latin, ISO Latin-1 or Katakana)	160	160
Text characters in a string literal (Kanji)	80	80
Numeric characters in a decimal literal (not including the sign)	18	18
Size in characters of a <text-workspace-field> (Simple Latin, ISO Latin-1 or Katakana)	30000	30000
Size in characters of a <text-workspace-field> (Kanji)	15000	15000
Size in characters of a <decimal-workspace-field> (not including the sign)	18	18
<audit-information-list> items per <audit> statement	30	30
<field-definition>s per <data-type-definition>	1023	1023
Arguments per <call-presentation-procedure>	29	29
Arguments per <call-task>, <call-procedure> or <submit-task>	30	30
Workspaces defined within a <task-definition>	127	127
Statements in a task definition	4095	4095
Number of destinations per <assignment>	15	15
<task-definition>s per task group	2000	1000
<processing-procedure>s per <processing-group-specification>	4095	1023
Levels of nested statements in a <statement-list>[3]	30	30
CASE clauses per <select-first> or <control-field>	255	255
Level of nested <include>[3]	10	10
Levels of nested <record-type>s[1,3]	15	15
Levels of nested arrays[1,3]	6	6
Array size	32767	32767
Exception code value range (1 . . . n)	8192	8192
Alternate mappings per display	5	N/A
Level of nested local composable tasks[3]	7	7
Max total size for marshaled task call arguments	1.5	0.75MB
Text characters in a <string-literal> or <workspace-field> used as a <key-value> (Simple Latin, ISO Latin-1 or Katakana)	120	120
Text characters in a <string-literal> or <workspace-field> used as a <key-value> (Kanji)	60	60
Line length (before preprocessor macro substitution and not including any end-of-line character)[2]	255	255
Line length (after preprocessor macro substitution and not including any end-of-line character)[2]	255	255
Preprocessor macro substitution recursion depth	10	10
Maximum number of <message-parameter>s in a <message-text>	9	9

Table A-2 STDL Minimum Limits

Notes:

1. When a <data-type-definition> includes both nested <record-type>s and nested arrays, the minimum limit for both applies to the entire <data-type-definition>.

2. An implementation is not required to continue processing a line that exceeds the defined length after macro substitution.
3. The top-level syntax element is not counted.

Table A-3 shows the required minimum limits for I/O operations.

	Profile	Profile
Files that can be accessed simultaneously per TP system[1]	1000	31
Files (other than indexed files) that can be accessed simultaneously within a transaction on a single TP system[1]	127	15
COBOL indexed files that can be accessed simultaneously within a single transaction on a single TP system[1]	31	15
C-ISAM files that can be accessed simultaneously within a single transaction on a TP system	10	10
Number of alternate indexes for a C-ISAM file	15	15
SQL tables that can be accessed per TP system	2047	63
SQL tables that can be accessed within a transaction on a single TP system	63	31

Table A-3 I/O Minimum Limits

Notes:

1. ''Files'' refers to files opened by the TP system and does not include files opened directly by COBOL and C procedures.

Table A-4 shows the required minimum limits for environment execution.

	Profile	**Profile**
Keys for a single record queue	8	8
CPU time limit	960 sec	960 sec
Transaction time limit	3600 sec	3600 sec
Presentation procedure time limit	3600 sec	3600 sec
Dequeue/read queue record time limit	65535 sec	65535 sec
Statement sequences (threads) in <concurrent-block> processing	16	8
Execution priorities	8	8
Max fault limit per processing procedure	32767	32767
Max fault limit per task	32767	32767
Restart limit per task	15	15
Retry limit per task queue	15 min	15 min
Retry limit per task queue	32767	32767
Size in octets of authentication information (default or customer-written)	255	255
Size in octets of a principal name	64	64

Table A-4 Environment Execution Minimum Limits

A.2 Exception Class Values

Exception class values are assigned as follows:

- no class has a value of zero
- negative numbers are used for faults
- positive numbers are used for errors.

The following lists the values assigned to faults:

–1 FATAL-TIMEOUT-FAULT

–2 FATAL-EXECUTION-FAULT

–3 AP-INVOCATION-FAULT

–4 ENV-INVOCATION-FAULT

–5 AP-RESPONSE-FAULT

–6 AP-EXECUTION-FAULT

–7 ENV-EXECUTION-FAULT

–8 SYSTEM-SHUTDOWN-FAULT

–9 AP-PROCESSING-FAULT

–10 ENV-UNSPECIFIED-FAULT.

The following lists the values assigned to errors:

1 ENV-INVOCATION-ERROR

2 TXN-FAILURE-ERROR

3 AP-INCOMPLETE-ERROR

4 TXN-TIMEOUT-ERROR

5 TXN-INCOMPLETE-ERROR

6 ENV-EXECUTION-ERROR

7 REQUEST-TIMEOUT-ERROR

8 INVALID-INPUT-ERROR

9 NO-OUTPUT-ERROR.

Audit Data Type Definition

This appendix is normative.

This appendix defines the common format for the system audit log records. Each vendor must produce a utility that converts the system proprietary audit log into a sequential file, as specified in Section 3.8.6 on page 71, or a stream file, as specified in Section 3.8.7 on page 71, depending on the programming languages supported by the system.

The data within the converted log must have the following format:

```
TYPE audit IS RECORD;
    event-time TEXT SIZE 17;
    source INTEGER;
    task-name TEXT SIZE 32;
    task-group-name TEXT SIZE 32;
    tp-system-name TEXT SIZE 256;
    default-device TEXT SIZE 256;
    data-length INTEGER;
    audit-data ARRAY
        SIZE 0 TO 30000 DEPENDING ON data-length
        OF OCTET;
END RECORD;
```

1. The event-time field is a 17-character text field, which represents the time that the event occurred. The format of the text is YYYYMMDDHHMMSSMMM, where:

   — YYYY is the year from 0000 to 9999

   — MM is the month from 1 to 12

   — DD is the day from 1 to 31

   — HH is the hour from 0 to 23

   — MM is the minute from 0 to 59

   — SS is the second from 0 to 59

   — MMM is the millisecond from 000 to 999.

   Implementor Note:

   > The time recorded in the event-time field must be significant to at least the second (as provided by the system clock). An implementation can choose to record times with a granularity of up to the millisecond.

2. The source field is an integer that indicates who generated the audit record. The possible values are:

   0 The TP system generated the audit record.

   1 The task generated the audit record using <audit>.

3. The contents of the following four fields are taken from the SYSTEM-INFO-WORKSPACE for the current task when that audit record is associated with a particular task instance. If the audit record is not associated with a task instance, these fields contain spaces.

— The task-name field is a 32-character text field containing the task invocation name used to invoke the task that produced the audit record.

— The task-group-name is a 32-character text field containing the name of the task group that produced the audit record.

— The tp-system-name is a 256-character text field containing the symbolic address for the system that produced the audit record.

— The default-device is a 256-character text field containing the display name for the submitter's device.

4. The data-length is an integer field containing the length of the optional data associated with the audit record. For audit records generated by the task using <audit>, this length represents the total length of all workspaces and/or workspace fields specified in <audit>.

5. The audit-data is a varying-length array of octets containing the optional data associated with the audit record. For audit records generated by the task using <audit>, the audit data contains the contents of all workspaces and/or workspace fields specified in <audit>.

Exception Classes

The list of exception classes and their types in this appendix is normative. Table C-1 on page 499 is informative.

This appendix lists STDL exception classes. Following each exception class name is the default exception type associated with that class.

STDL defines the following exception classes:

1. FATAL-TIMEOUT-FAULT — Fatal Transaction Exception
   a. The operation in progress failed to complete before an applicable time limit.
   b. This exception class indicates the following error condition:
      — The CPU time limit environmental attribute was exceeded.
   c. This exception class indicates one of the following responses:
      — Correct the problem that caused the operation not to complete within the allotted CPU time.
      — Write key events and any applicable data to the audit log within the operation that failed to complete.
      — Modify the current value of the CPU time limit environmental attribute.
2. FATAL-EXECUTION-FAULT — Fatal Transaction Exception
   a. The task, presentation procedure or processing procedure determined that the current task cannot complete successfully.
   b. This exception class indicates one of the following error conditions:
      — An exception code was returned by the task, processing procedure or presentation procedure that maps to this exception class in the message group.
      — This exception class was returned by the task, processing procedure or presentation procedure.
   c. This exception class indicates one of the following responses:
      — Check the exception code and correct the problem that caused the fatal transaction exception.
      — Write key events and any applicable data to the audit log.
3. AP-INVOCATION-FAULT — Permanent Transaction Exception
   a. The operation being requested cannot be started due to a specification or named record mismatch condition.
   b. This exception class indicates one of the following error conditions:
      — A specification mismatch occurred on a <call-procedure>. The interface in the <processing-group-specification> for the called procedure that was used when processing the calling <task-definition> into an executable task is incompatible with the interface in the called processing procedure.

— A specification mismatch occurred on a <call-task>. The <task-group-specification> that was used when processing the calling <task-definition> into an executable task is incompatible with the <task-group-specification> that was used when processing the called <task-definition> into an executable task.

— A specification mismatch occurred on a <submit-task>. The <task-group-specification> that was used when processing the submitting <task-definition> into an executable task is incompatible with the <task-group-specification> that was used when processing the submitted <task-definition> into an executable task.

— A specification mismatch occurred on a <call-presentation-procedure> statement for a customer-written send, receive or transceive presentation procedure. The interface in the <presentation-group-specification> that was used when processing the calling <task-definition> into an executable task is incompatible with the interface in the called presentation procedure.

c. This exception class indicates one of the following responses:

— Correct the task group, processing group or presentation group specification that caused the specification mismatch condition to occur.

— Correct the task, processing procedure or presentation procedure in which the error occurred.

— Correct the vendor-supplied forms system that caused the named record mismatch condition to occur.

— Correct the control text format or length in the vendor-supplied forms system.

4. ENV-INVOCATION-FAULT — Permanent Transaction Exception

a. The operation being requested cannot be started due to an environmental information problem. A retry of the operation cannot be successful unless either the environmental information is corrected or the the syntactical reference to the environmental information is changed.

b. This exception class indicates one of the following error conditions:

— The processing group specified by the <processing-group-identifier> in the <call-procedure> is not defined within the environmental information of the TP system in which the task is executing.

— The task group specified by the <task-group-identifier> in the <call-task> is not defined within the environmental information of the server TP system.

— The task specified by the <task-identifier> in the <call-task> could not be started due to the access to the called task being denied.

— The destination specified by the value of destination name in the <call-task> is not defined within the environmental information of the TP system in which the task is executing.

— If the destination was not explicitly specified in the <call-task>, then the destination specified by the <task-group-identifier> is not defined within the environmental information.

— The task queue is not defined within the environmental information of the submitter TP system.

- — The distribution list specified by the value of distribution list in the <submit-task> is not defined within the environmental information of the submitter TP system.
- — If the destination name or distribution list is not specified in a <submit-task> and if a default destination is used, that default destination is not defined within the environmental information of the submitter TP system.
- — The record queue name is not defined within the environmental information of the TP system in which the task is executing.
- — The key specified by the <workspace-field> in the <dequeue-record> is not defined within the environmental information of the TP system in which the task is executing.
- — The key specified by the <workspace-field> in the <read-queue-record> is not defined within the environmental information of the TP system in which the task is executing.
- — The presentation group specified by the <presentation-group-identifier> in the <call-presentation-procedure> statement is not defined within the environmental information of the TP system in which the task is executing.
- — The broadcast list specified by the value of <broadcast-list-name> in the <call-presentation-broadcast> is not defined within the environmental information of the TP system in which the task is executing.
- — A second task call was executed using a dialogue that accepts only a single task call per dialogue.
- — The destination specified by the value of destination name or a destination name included in a distribution list of a <submit-task> is not defined within the environmental information of the submitter TP system.
- — A task was forwarded using transactions and the client TP system does not support distributed transactions.
- — The <uuid-string-literal> in the task group specification used by the client program for the server task in not the same as the <uuid-string-literal> used by the server task.
- — The major version number used by the client program is not equal to the major version number used by the server task.
- — The minor version number used by the client program is greater than the minor version number used by the server task.

c. This exception class indicates one of the following responses:

- — Correct the environmental problem by modifying the environmental information to allow a successful operation.
- — Correct the API problem by modifying the STDL source to allow a successful operation.

5. ENV-INVOCATION-ERROR — Permanent Transaction Exception

a. The operation being requested cannot be started due to the current setting of an environmental information attribute or the availability of the environmental entity due to system failure. The operation may be successful if retried at a later time.

b. This exception class indicates one of the following error conditions:

— The processing procedure specified by the <processing-procedure-identifier> in the <call-procedure> is disabled.

— The processing group specified by the <processing-group-identifier> in the <call-procedure> is not available due to a system failure.

— The task specified by the <task-identifier> in the <call-task> could not be started due to the task instance high water mark being exceeded on the server TP system.

— The task specified by the <task-identifier> in the <call-task> is disabled.

— The task specified by the <task-identifier> in the <call-task> could not be started due to the server TP system not being available because of system failure or not being started.

— The task group specified by the <task-group-identifier> in the <call-task> is not available due to a system failure.

— The task specified by the <task-identifier> in the <call-task> could not be started due to a communications failure.

— The task queue is disabled.

— The presentation group specified by the <presentation-group-identifier> in the <call-presentation-procedure> statement is not available due to a system failure.

— The display used by the <call-presentation-procedure> statement is off line and not currently not available for use.

— The task enqueuer is not available due to a system failure.

— The task enqueuer is not available due to a communications failure.

c. This exception class indicates one of the following responses:

— Retry the operation at some later time when the TP system or system manager may have modified the environmental attribute to allow for successful operation.

— Retry the operation at some later time when the communications problem may have been corrected.

— Retry the operation at some later when the TP system or system manager may have corrected the system failure problem.

— If unsuccessful after retrying the operation, correct the problem using information in the exception code that caused the exception.

6. TXN-FAILURE-ERROR — Permanent Transaction Exception

a. The current transaction failed to commit successfully.

b. This exception class indicates one of the following error conditions:

— One of the resource managers participating in the transaction was unable to commit the transaction.

— One of the resource managers participating in the transaction failed to acknowledge a message when attempting to commit the transaction.

c. This exception class indicates one of the following responses:

   — Based on the exception code, correct the problem that prevented the transaction from committing and retry the operation.

7. AP-RESPONSE-FAULT — Permanent Transaction Exception

   a. The operation being requested could not complete due to invalid results.

   b. This exception class indicates one of the following error conditions:

      — The results of the <call-procedure> could not be returned to the <task-definition> that called the processing procedure due to a syntax error.

      — The results of the <call-task> could not be returned to the calling <task-definition> due to a syntax error.

      — The results of the <call-presentation-procedure> statement could not be returned to the <task-definition> that called the presentation procedure due to a syntax error.

   c. This exception class indicates one of the following responses:

      — Correct the task group, processing group or presentation group specification that caused the error condition to occur.

      — Correct the task, processing procedure or presentation procedure in which the error occurred.

      — Correct the vendor-supplied forms system that caused the error condition to occur.

8. AP-INCOMPLETE-ERROR — Permanent Transaction Exception

   a. The task, presentation procedure or processing procedure determined that the current operation can not be completed successfully at the present time since the data passed as arguments on the call was erroneous. The task, presentation procedure or processing procedure wanted to roll back any transactional resources modified during the operation.

   b. This exception class indicates one of the following error conditions:

      — Any exception code returned by the task, processing procedure or presentation procedure that maps to this exception class in the message group.

      — This exception class returned by the task, processing procedure or presentation procedure.

   c. This exception class indicates one of the following responses:

      — Modify the input data and reexecute the called task, processing procedure or presentation procedure

      — Based on the exception code, correct the problem that caused the permanent transaction exception.

      — Record key events and any applicable data to the audit log.

9. TXN-TIMEOUT-ERROR — Transient Transaction Exception

   a. The transaction in progress has failed to complete before an applicable time limit.

   b. This exception class indicates the following error condition:

      — The transaction time limit environmental attribute was exceeded.

c. This exception class indicates one of the following responses:

— This exception class will not be delivered to an exception handler unless the retry limit has been exceeded or a non-transactional <call-presentation-procedure> statement was performed during the execution of this transaction.

— Correct the problem that caused the operation not to complete in the allotted transaction time.

— Record key events and any applicable data to the audit log.

— Review the value of the "transaction time limit" environmental attribute.

10. TXN-INCOMPLETE-ERROR — Transient Transaction Exception

a. The task, presentation procedure or processing procedure determined that the current transaction can not be completed successfully at the present time but may be successful if retried.

b. This exception class indicates one of the following error conditions:

— Any exception code was returned by the task, processing procedure or presentation procedure that maps to this exception class in the message group.

— This exception class was returned by the task, processing procedure or presentation procedure.

c. This exception class indicates one of the following responses:

— Based on the exception code, correct the problem that caused the transient transaction exception.

— Record key events and any applicable data to the audit log.

11. AP-EXECUTION-FAULT — Permanent Transaction Exception

a. The operation being requested could not complete successfully due to an application logic or programming error.

b. This exception class indicates one of the following error conditions:

— The processing procedure specified by the <processing-procedure-identifier> in the <call-procedure> caused the TP system to generate an exception. The exception was due to a access violation, an unhandled divide by zero or other unhandled programming or logic errors caused by the execution of the processing procedure.

— The presentation procedure executed by the <call-presentation-procedure> statement caused the TP system to generate an exception. The exception was due to a access violation, an unhandled divide by zero, or other unhandled programming or logic error caused by the execution of the presentation procedure.

— The presentation procedure executed by the <call-presentation-procedure> statement caused the TP system to generate an exception. The exception was due to a display error condition, caused when the presentation procedure tried to access the display.

— Non-composable task was called to execute within the client task's transaction

— Composable task was called to execute independently from the client task's transaction

- The subscript used to reference an element in a fixed-length array is invalid (referring to A(n), where n < 1 or n > maximum size of array).
- The subscript used to reference an element in a varying-length array is invalid (referring to A(n), where n < 1 or n > the value of the depending on field).
- The value of the depending on field used when referencing an element of a varying-length array is invalid (the value of the depending on field < minimum size of array or the value of the depending on field > maximum size of array).
- Source and destination of <assignment> do not agree in data type.
- The <integer-workspace-field-2> of a <get-message> contains a value other than 0 or 1.
- The exception code specified in the <restart-transaction> or <raise-exception> is not defined within an associated <message-group-definition>.
- The exception class specified in the <restart-transaction> or <raise-exception> is not one of the standard, predefined exception classes.
- The size of the workspace specified by <workspace-identifier> does not match the size of the target <data-type-definition> associated with the record queue. For a workspace or record that contains a varying-length array, the size of the workspace or record is determined by the DEPENDING ON field for the array.
- The value of the DEPENDING ON field was invalid when referencing a <workspace-identifier> or <workspace-field> containing a varying-length array.
- The character-set-supported data type of the <text-workspace-field-1> or <text-workspace-field-2> cannot handle the human language of the message text.
- IN <message-group-identifier> is specified and the value of <integer-workspace-field-2> is 0 (for system).
- The value of <uuid-workspace-field> or <uuid-string-literal> did not match a known exception group or the value of <message-group-identifier> did not match a known <message-group-definition>.
- A DEQUEUE NEXT RECORD operation was unable to complete successfully because the previous operation was neither a DEQUEUE FIRST RECORD nor a DEQUEUE NEXT RECORD.
- A <read-queue-next> operation was unable to complete successfully because the previous operation was neither a <read-queue-first> nor a <read-queue-next>.
- A <read-queue-key-next> operation was unable to complete successfully because the previous operation was neither a <read-queue-key-first> nor a <read-queue-key-next>.

c. This exception class indicates the following response:

- Correct the application logic or programming error

12. ENV-EXECUTION-FAULT — Non-transaction Exception

a. The operation being requested has initiated, but did not complete successfully due to an environmental problem. A retry of the operation will not be successful until the environmental problem is corrected.

b. This exception class indicates one of the following error conditions:

— The submitter TP system encountered a problem storing the task submission request onto the task queue. The submitter TP system encountered a problem removing the task submission request from the task queue.

— The TP system in which the task is executing encountered a problem storing an item into the record queue.

— The TP system in which the task is executing encountered a problem removing (dequeuing) an item from the record queue.

— The TP system in which the task is executing encountered a problem reading an item on the record queue.

— The TP system in which the task is executing is unable to write the <audit-information-list> onto the audit log.

— The task enqueuer could not complete the forwarding operation due to a problem encountered in storing the task submission request onto the task queue.

— The submitted task specified by <id-workspace-field> could not be removed from the task queue because no submitted task with the value specified by <id-workspace-field> was found on the local task queue, or the submitted task is executing.

c. This exception class indicates one of the following responses:

— Correct the environmental problem.

13. ENV-EXECUTION-ERROR — Non-transaction Exception

a. The operation being requested has initiated, but did not complete successfully due to an environmental problem. The operation maybe successful if retried at some later time.

b. This exception class indicates one of the following error conditions:

— The processing procedure specified by the <processing-procedure-identifier> in the <call-procedure> could not complete due to a system failure on the TP system in which the task is executing.

— The task specified by the <task-identifier> in the <call-task> could not complete successfully due to a system server TP system.

— The task specified by the <task-identifier> in the <call-task> could not complete and return results of the call due to a communication failure between the client TP system and the remote server TP system.

— The presentation procedure executed by the <call-presentation-procedure> statement could not complete due to a system failure of the TP system in which the task is executing.

— The task enqueuer could not complete the forwarding operation and return results due to a communications failure.

— The call was cancelled.

c. This exception class indicates one of the following responses:

— Retry the operation at some later time since the TP system or system manager may have corrected the system failure problem.

— Retry the operation at some later time since the communication problem may have been corrected.

— If unsuccessful after desired retries, correct problem based on the exception code that caused the exception.

14. AP-PROCESSING-FAULT — Non-transaction Exception

    a. The task, presentation procedure or processing procedure determined that an error condition occurred during the current operation which may be successful if retried at a later time.

    b. This exception class indicates one of the following error conditions:

       — Any exception code was returned by the task, processing procedure or presentation procedure that maps to this exception class in in the message group.

       — This exception class was returned by the task, processing procedure or presentation procedure.

    c. This exception class indicates one of the following responses:

       — Based on the exception code, correct the problem that caused the non-transaction exception.

       — Record key events and any applicable data to the audit log.

15. REQUEST-TIMEOUT-ERROR — Non-transaction Exception

    a. The requested operation failed to complete before an applicable time limit.

    b. This exception class indicates one of the following error conditions:

       — The presentation procedure time limit environmental attribute was exceeded.

       — The dequeue record time limit environmental attribute was exceeded.

    c. This exception class indicates one of the following responses:

       — Retry the operation again.

       — Take an alternate action.

16. SYSTEM-SHUTDOWN-FAULT — Non-transaction Exception

    a. The TP system aborted the operation in progress.

    b. This exception class indicates the following error condition:

       — A system shutdown request was issued while the queue operation is waiting (WAIT is specified) for an item to appear in the record queue.

    c. This exception class indicates one of the following responses:

       — The exception handler should complete processing and end the task to allow the system shutdown to complete.

17. INVALID-INPUT-ERROR — Non-transaction Exception

    a. The called task, processing procedure or presentation procedure determined that the data passed as arguments on the call was erroneous.

    b. This exception class indicates one of the following error conditions:

       — Any exception code returned by the task, processing procedure or presentation procedure that maps to this exception class in the message group.

    - This exception class returned by the task, processing procedure or presentation procedure.

c. This exception class indicates one of the following responses:

    - Modify the input data and retry the operation.

18. NO-OUTPUT-ERROR — Non-transaction Exception

    a. The requested operation was unable to return the requested data.

    b. This exception class indicates one of the following error conditions:

        - WAIT was not specified and the record queue is empty or no record is found, so no data is returned.

        - Any exception code returned by the task, processing procedure or presentation procedure that maps to this exception class in the message group.

        - This exception class returned by the task, processing procedure or presentation procedure.

        - The value of the <get-message-number> specified in the <get-message> did not match a message number defined in the <message-group-definition>, so no text is returned.

        - The value of the human language name specified in the <get-message> did not match a human language name defined in the <message-group-definition>.

        - The value of <uuid-workspace-field> or <uuid-string-literal> did not match a known exception group or message group identifier did not match <message-group-definition>.

    c. This exception class indicates one of the following responses:

        - Take an alternate action.

        - Try the operation again with different data.

19. ENV-UNSPECIFIED-FAULT — Permanent Transaction Exception

    a. The requested operation failed due to an RTI protocol error.

    b. This exception class indicates the following error condition:

        - The called task did not complete due to an unknown error condition in the RTI protocol.

    c. This exception class indicates one of the following responses:

        - Check the exception code and correct the problem that caused the exception.

        - Write key events and any applicable data to the audit log.

The exception types specified next to the exception classes indicates exception when the exception was initially generated.

The following exception classes are never raised by the TP system:

- FATAL-EXECUTION-FAULT
- AP-PROCESSING-FAULT
- AP-INCOMPLETE-ERROR

- TXN-INCOMPLETE-ERROR
- INVALID-INPUT-ERROR.

Table C-1 shows the exception types when exceptions are raised.

This table is informative.

Initial Exception Type ()	Fatal T'out Fault (F)	Fatal Exec Fault (F)	Ap Inv Fault (P)	Env Inv Fault (P)	Env Inv Error (P)	Txn Fail Error (P)	Ap Resp Fault (P)	Ap In-compl Error (P)	Ap Exec Fault (P)	Env U'spec Fault (P)	Txn T'out Error (T)	Txn In-compl Error (T)	Env Exec Fault (N)	Env Exec Error (N)	Req T'out Error (N)	Sys S'dwn Fault (N)	Invld I'put Error (N)	No O'put Error (N)	Ap Proc'g Fault (N)
By Proc/Pres Pr'dures	F	F	P	P	P	P	P	P	P	P	T	T	N	N	N	N	N	N	N
<call-procedure>			P	P	P		P												
<call-task> (dep)			P	P	P		P		P	P									
<call-task> (indep)			P	P	P		P		P	P									
<submit-task> (dep)			P	P	P				P				N						
<submit-task> (indep)			P	P	P				P				N						
<enqueue-record>				P					P				N						
<dequeue-record>				P					P				N		N	N		N	
<read-queue-record>				P					P				N		N	N		N	
<assignment>									P										
send proc (txn)			P	P	P				P										
send proc (non-txn)			P	P	P		P		P						N				
receive proc (txn)			P	P	P		P		P						P				
receive proc (non-txn)			P	P	P		P		P						N				
transceive proc (non-txn)			P	P	P		P		P						N				
<audit>									P				N						
<get-message>									P									N	
<restart-transaction>	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T
<raise-exception>	F	F	P	P	P	P	P	P	P	P	T	T	N	N	N	N	N	N	N
<raise-exception> (w/R)	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P
<transaction-control>						P													
conditionals									P										
During task execution	F										T								

Table C-1 Exception Types

Notes:

N Non-transaction.

P Permanent transaction.

T Transient transaction.

F Fatal transaction.

Appendix D

Complete BNF

This appendix is informative.

This appendix includes a subsection for the complete BNF for each part of STDL syntax.

D.1 Data Type Definition

```
<data-type-definition> ::
    TYPE <data-type-identifier> [IS] <record-type> ;

<data-type-identifier>::
    <external-identifier>

<record-type> ::
    RECORD
        <field-list>
    END [RECORD]

<field-list> ::
    <field-definition> ...

<field-definition> ::
    <field-identifier> [IS] <data-type> ;

<field-identifier>  ::
    <internal-identifier>

<data-type> ::
      <array-type>
    | <data-type-identifier>
    | <decimal-string-type>
    | <integer-type>
    | <octet-type>
    | <record-type>
    | <text-type>
    | <uuid-type>

<octet-type> ::
    OCTET

<integer-type> ::
    INTEGER <integer-initializer>

<integer-initializer> ::
      <nil>
    | = <integer-literal>

<text-type> ::
```

```
      TEXT <text-character-set> SIZE <integer-literal> <text-initializer>
    | NATIONAL TEXT SIZE <integer-literal> <text-initializer>

<text-character-set> ::
      <nil>
    | CHARACTER SET <character-set-identifier>

<text-initializer> ::
      <nil>
    | = <string-literal>

<decimal-type> ::
    DECIMAL STRING
        SIZE <integer-literal-1>
        <decimal-scale>
        <decimal-initializer>

<decimal-scale> ::
      <nil>
    | SCALE <integer-literal-2>

<decimal-initializer> ::
      <nil>
    | = <decimal-literal>

<array-type> ::
      <fixed-length-array-type>
    | <varying-length-array-type>

<fixed-length-array-type> ::
    ARRAY
        SIZE <integer-literal-1>
        OF <data-type>

<varying-length-array-type> ::
    ARRAY
        SIZE <integer-literal-2>
        TO <integer-literal-3>
        DEPENDING ON <field-identifier>
        OF <data-type>

<uuid-type> ::
    UUID
```

D.2 Message Definition Language

```
<message-group-definition> ::
    MESSAGE [GROUP] <message-group-identifier>
        <message-group-attribute-list>
        <message-list>
    END [MESSAGE] [GROUP] ;

<message-group-identifier> ::
    <external-identifier>

<message-group-attribute-list> ::
    <message-group-attribute> ...

<message-group-attribute> ::
      <message-group-language>
    | <message-group-uuid>

<message-group-language> ::
    LANGUAGE [IS] <human-language-literal> ;

<message-group-uuid> ::
    UUID [IS] <uuid-string-literal> ;

<message-list> ::
    <message-definition> ...

<message-definition> ::
    <message-identifier>
        VALUE [IS] <integer-literal>
        CLASS [IS] <class-identifier>
        TEXT [IS] <message-text> ;

<message-identifier> ::
    <external-identifier>

<message-text> ::
    <message-text-element> ...

<message-text-element> ::
      <string-literal>
    | <message-parameter>
```

D.3 Presentation Group Specification

```
<presentation-group-specification> ::
    PRESENTATION GROUP [SPECIFICATION] <presentation-group-identifier>
        <presentation-group-attribute-list>
        <presentation-procedure-list>
    END [PRESENTATION] [GROUP] [SPECIFICATION] ;

<presentation-group-identifier> ::
    <external-identifier>

<presentation-group-attribute-list> ::
    <presentation-group-attribute> ...

<presentation-group-attribute> ::
      <presentation-initialization-procedure>
    | <presentation-source-language>
    | <presentation-termination-procedure>

<presentation-source-language> ::
    SOURCE [LANGUAGE] [IS] <computer-language-literal> ;

<presentation-initialization-procedure> ::
    INITIALIZATION PROCEDURE [IS] <presentation-procedure-identifier> ;

<presentation-procedure-identifier> ::
    <external-identifier>

<presentation-termination-procedure> ::
    TERMINATION PROCEDURE [IS] <presentation-procedure-identifier> ;

<presentation-procedure-list> ::
    <presentation-procedure-interface> ...

<presentation-procedure-interface> ::
    PROCEDURE <presentation-procedure-identifier>
    <presentation-procedure-argument-type> ;

<presentation-procedure-argument-type> ::
      <presentation-procedure-send-argument-list>
    | <presentation-procedure-receive-argument-list>
    | <presentation-procedure-transceive-argument-list>

<presentation-procedure-send-argument-list> ::
    SENDING <presentation-procedure-argument-list>

<presentation-procedure-receive-argument-list> ::
    RECEIVING <presentation-procedure-argument-list>

<presentation-procedure-transceive-argument-list> ::
    SENDING <presentation-procedure-argument-list>
    RECEIVING <presentation-procedure-argument-list>
```

```
<presentation-procedure-argument-list> ::
      <nil>
    | <data-type-identifier> , ...
```

D.4 Processing Group Specification

```
<processing-group-specification> ::
    PROCESSING GROUP [SPECIFICATION] <processing-group-identifier>
        <processing-source-language>
        <processing-procedure-list>
    END [PROCESSING] [GROUP] [SPECIFICATION] ;

<processing-group-identifier> ::
    <external-identifier>

<processing-source-language> ::
    SOURCE [LANGUAGE] [IS] <computer-language-literal> ;

<processing-procedure-list> ::
    <processing-procedure-interface> ...

<processing-procedure-interface> ::
    PROCEDURE <processing-procedure-identifier>
        <processing-argument-specification> ;

<processing-argument-specification> ::
      <nil>
    | USING <processing-interface-argument-list>

<processing-interface-argument-list> ::
    <processing-interface-argument> , ...

<processing-interface-argument> ::
    <data-type-identifier> <processing-interface-argument-passing>

<processing-interface-argument-passing> ::
      <nil>
    | PASSED [AS] <processing-interface-argument-passing-direction>

<processing-interface-argument-passing-direction> ::
      INPUT
    | OUTPUT
    | INOUT

<processing-procedure-identifier> ::
    <external-identifier>
```

D.5 Task Group Specification

```
<task-group-specification> ::
    TASK GROUP [SPECIFICATION] <task-group-identifier>
        <task-group-attribute-list>
        <task-interface-list>
    END [TASK] [GROUP] [SPECIFICATION] ;

<task-group-identifier> ::
    <external-identifier>

<task-group-attribute-list> ::
    <task-group-attribute> ...

<task-group-attribute> ::
      <task-group-uuid-attribute>
    | <task-group-version-attribute>

<task-group-uuid-attribute> ::
    UUID [IS] <uuid-string-literal> ;

<task-group-version-attribute> ::
    VERSION [IS] <decimal-literal> ;

<task-interface-list> ::
    <task-interface> ...

<task-interface> ::
      <composable-task-interface>
    | <noncomposable-task-interface>

<noncomposable-task-interface> ::
    TASK <task-identifier>
        <task-interface-argument-list-definition> ;

<composable-task-interface> ::
    COMPOSABLE TASK <task-identifier>
        <task-interface-argument-list-definition> ;

<task-identifier> ::
    <external-identifier>

<task-interface-argument-list-definition> ::
      <nil>
    | USING <task-interface-argument-list>

<task-interface-argument-list> ::
    <task-interface-argument-definition> , ...

<task-interface-argument-definition> ::
    <data-type-identifier> <task-interface-argument-passing>

<task-interface-argument-passing> ::
```

```
        <nil>
      | PASSED [AS] <task-interface-argument-passing-direction>

<task-interface-argument-passing-direction> ::
        INPUT
      | OUTPUT
      | INOUT
```

D.6 Task Definition Language

```
<task-definition> ::
      <composable-task>
    | <noncomposable-task>

<noncomposable-task> ::
    TASK <task-identifier> IN <task-group-identifier>
        <task-argument-list-definition>
        <task-attribute-list>
        <statement-list>
    END [TASK] ;

<composable-task> ::
    COMPOSABLE TASK <task-identifier> IN <task-group-identifier>
        <task-argument-list-definition>
        <task-attribute-list>
        <statement-list>
    END [TASK] ;

<task-argument-list-definition> ::
      <nil>
    | USING <task-argument-list> ;

<task-argument-list> ::
    <task-argument-definition> , ...

<task-argument-definition> ::
    <data-type-identifier> <task-argument-passing>
        <task-argument-transactional>
    | <workspace-identifier> [IS] <data-type-identifier>
        <task-argument-passing>
        <task-argument-transactional>

<task-argument-passing> ::
      <nil>
    | PASSED [AS] <task-argument-passing-direction>

<task-argument-passing-direction> ::
      INPUT
    | OUTPUT
    | INOUT

<task-argument-transactional> ::
      <nil>
    | TRANSACTIONAL
    | NOT TRANSACTIONAL

<workspace-identifier> ::
    <internal-identifier>

<task-attribute-list> ::
      <nil>
```

```
    | <task-attribute> ...

<task-attribute> ::
      <restartability>
    | <send-display>
    | <workspace-list-definition>

<restartability> ::
      NOT RESTARTABLE ;
    | RESTARTABLE ;

<workspace-list-definition> ::
      WORKSPACE [IS] <workspace-definition-list> ;
    | WORKSPACES [ARE] <workspace-definition-list> ;

<workspace-definition-list> ::
    <workspace-definition> , ...

<workspace-definition> ::
      <data-type-identifier> <workspace-attribute-list>
    | <workspace-identifier> [IS] <data-type-identifier>
        <workspace-attribute-list>

<workspace-attribute-list> ::
      <nil>
    | <workspace-attribute> ...

<workspace-attribute> ::
      NOT TRANSACTIONAL
    | TRANSACTIONAL
    | SHARED
    | SHARED READ
    | SHARED UPDATE
    | PRIVATE

<workspace-identifier> ::
    <internal-identifier>

<send-display> ::
      SEND DISPLAY ;
    | SEND DISPLAY [IS] <os-name-value> ;

<statement-list> ::
      <nil>
    | <statement> ...

<statement> ::
    <statement-label> <statement-type>;

<statement-label> ::
      <nil>
    | <label-identifier>;
```

```
<statement-type> ::
      <audit>
    | <assignment>
    | <call-presentation-procedure>
    | <call-procedure>
    | <call-task>
    | <cancel-submit>
    | <concurrent-block>
    | <control-field>
    | <dequeue-record>
    | <enqueue-record>
    | <exit-block>
    | <exit-task>
    | <get-message>
    | <go-to>
    | <if>
    | <raise-exception>
    | <read-queue-record>
    | <reraise-exception>
    | <restart-transaction>
    | <select-first>
    | <statement-block>
    | <submit-task>
    | <transaction-block>
    | <while>

<label-identifier> ::
    <internal-identifier>

<transaction-block> ::
    BLOCK [WITH] TRANSACTION
        <statement-list>
    END [BLOCK] <transaction-block-end-label>
        <exception-handler>

<transaction-block-end-label> ::
      <nil>
    | <label-identifier>

<statement-block> ::
    BLOCK
        <statement-list>
    END [BLOCK] <statement-block-end-label>
        <exception-handler>

<statement-block-end-label> ::
      <nil>
    | <label-identifier>

<concurrent-block> ::
    BLOCK [WITH] CONCURRENT EXECUTION
        <concurrent-list>
```

```
    END [BLOCK] <concurrent-block-end-label>
        <exception-handler>

<concurrent-list> ::
      <nil>
    | <statement> ...

<concurrent-block-end-label> ::
      <nil>
    | <label-identifier>

<exception-handler> ::
      <nil>
    | <exception-handler-list>

<exception-handler-list> ::
    EXCEPTION [HANDLER] [IS]
        <statement-list>
    END [EXCEPTION] [HANDLER]

<if> ::
    IF <boolean-expression> THEN
        <statement-list>
    <else>
    END [IF] <if-end-label>

<else> ::
      <nil>
    | ELSE <statement-list>

<if-end-label> ::
      <nil>
    | <label-identifier>

<select-first> ::
    SELECT FIRST [TRUE] [OF]
        <select-first-case-list>
    END [SELECT] [FIRST] <select-first-end-label>

<select-first-case-list> ::
    <select-first-case> ...

<select-first-case> ::
      CASE <boolean-expression> : <statement-list>
    | CASE NOMATCH : <statement-list>

<select-first-end-label> ::
      <nil>
    | <label-identifier>

<control-field> ::
    CONTROL FIELD <control-field-to-match>
```

```
        <control-field-case-list>
    END [CONTROL] [FIELD] <control-field-end-label>

<control-field-to-match> ::
      <integer-workspace-field>
    | <text-workspace-field>
    | <decimal-workspace-field>

<control-field-case-list> ::
    <control-field-case> ...

<control-field-case> ::
      CASE <control-field-value> : <statement-list>
    | CASE NOMATCH : <statement-list>

<control-field-value> ::
      <integer-literal>
    | <message-identifier>
    | <class-identifier>
    | <integer-workspace-field>
    | <decimal-literal>
    | <decimal-workspace-field>
    | <string-literal>
    | <text-workspace-field>

<control-field-end-label> ::
      <nil>
    | <label-identifier>

<while> ::
    WHILE <boolean-expression> DO
        <statement-list>
    END [WHILE] <while-end-label>

<while-end-label> ::
      <nil>
    | <label-identifier>

<call-procedure> ::
    CALL PROCEDURE <processing-procedure-identifier>
        IN <processing-group-identifier>
        <call-procedure-using>

<call-procedure-using> ::
      <nil>
    | USING <call-procedure-argument-list>

<call-procedure-argument-list> ::
    <workspace-identifier> , ...

<call-task> ::
    CALL TASK <task-identifier>
```

```
        <call-task-parts-list>
        <call-task-with>
        <call-task-using>

<call-task-with>   ::
      <nil>
    | [WITH] INDEPENDENT [WORK]
    | [WITH] DEPENDENT [WORK]

<call-task-parts-list> ::
      <nil>
    | <call-task-part> ...

<call-task-part> ::
      <call-task-in>
    | <call-task-at>

<call-task-in> ::
    IN <task-group-identifier>

<call-task-at> ::
    AT <os-name-value>

<call-task-using> ::
      <nil>
    | USING <call-task-argument-list>

<call-task-argument-list>::
    <workspace-identifier>, ...

<submit-task> ::
    SUBMIT TASK <task-identifier>
        <submit-parts-list>
        <submit-with>
        <submit-using>

<submit-with> ::
      <nil>
    | [WITH] INDEPENDENT [WORK]
    | [WITH] DEPENDENT [WORK]

<submit-parts-list> ::
      <nil>
    | <submit-part> ...

<submit-part> ::
      <submit-in>
    | <submit-at>
    | <submit-id>
    | <submit-repeat>
    | <submit-trigger>
```

```
<submit-in> ::
    IN <task-group-identifier>

<submit-at> ::
    AT <os-name-value>

<submit-id> ::
      ID <id-workspace-field>
    | IDENTIFIER <id-workspace-field>

<submit-using> ::
      <nil>
    | USING <submit-argument-list>

<submit-argument-list> ::
    <workspace-identifier> , ...

<submit-trigger> ::
      HOLD FOR OPERATOR
    | HOLD FOR <delta-time-value>
    | HOLD UNTIL <absolute-time-value> <submit-hold-time-of>

<submit-hold-time-of> ::
      <nil>
    | OF SUBMITTER [SYSTEM]
    | OF CLIENT [SYSTEM]

<submit-repeat> ::
    REPEATING [EVERY] <delta-time-value>

<cancel-submit> ::
    CANCEL [TASK] SUBMIT <cancel-id>
        <cancel-with>

<cancel-with> ::
      <nil>
    | [WITH] INDEPENDENT [WORK]
    | [WITH] DEPENDENT [WORK]

<cancel-id> ::
      ID <id-workspace-field>
    | IDENTIFIER <id-workspace-field>

<enqueue-record> ::
    ENQUEUE RECORD
        <enqueue-parts-list>
        USING <workspace-identifier>

<enqueue-parts-list> ::
    <enqueue-part> ...

<enqueue-part> ::
```

```
      <enqueue-queue>
    | <enqueue-element-identifier>

<enqueue-queue> ::
    ON <os-name-literal>

<enqueue-element-identifier> ::
      ID <id-workspace-field>
    | IDENTIFIER <id-workspace-field>

<dequeue-record> ::
    DEQUEUE <dequeue-type>
        <dequeue-parts-list>
        INTO <workspace-identifier>

<dequeue-type> ::
      <dequeue-first>
    | <dequeue-key>
    | <dequeue-id>

<dequeue-first> ::
    FIRST RECORD

<dequeue-key> ::
    RECORD BY KEY <workspace-field-1> VALUE
        <dequeue-key-value>

<dequeue-id> ::
      RECORD BY ID <id-workspace-field-1>
    | RECORD BY IDENTIFIER <id-workspace-field-1>

<dequeue-key-value> ::
      <workspace-field-2>
    | <string-literal>
    | <integer-literal>
    | <decimal-literal>

<dequeue-parts-list> ::
    <dequeue-part> ...

<dequeue-part> ::
      <dequeue-queue>
    | <dequeue-wait>
    | <dequeue-element-identifier>

<dequeue-queue> ::
    FROM <os-name-literal>

<dequeue-wait> ::
      WITH WAIT
    | WITH NO WAIT
```

```
<dequeue-element-identifier> ::
      ID <id-workspace-field-2>
    | IDENTIFIER <id-workspace-field-2>

<read-queue-record> ::
    READ QUEUE <read-queue-type>
        <read-queue-parts-list>
        INTO <workspace-identifier>

<read-queue-type> ::
      <read-queue-first>
    | <read-queue-next>
    | <read-queue-id>
    | <read-queue-key-first>
    | <read-queue-key-next>

<read-queue-first> ::
    FIRST RECORD

<read-queue-next> ::
    NEXT RECORD

<read-queue-id> ::
      RECORD BY ID <id-workspace-field-1>
    | RECORD BY IDENTIFIER <id-workspace-field-1>

<read-queue-key-first> ::
    FIRST RECORD
        BY KEY <workspace-field-1>
        VALUE <read-queue-key-value>

<read-queue-key-value> ::
      <workspace-field-2>
    | <string-literal>
    | <integer-literal>
    | <decimal-literal>

<read-queue-key-next> ::
    NEXT RECORD BY KEY

<read-queue-parts-list> ::
    <read-queue-part> , ...

<read-queue-part> ::
      <read-queue-queue>
    | <read-queue-wait>
    | <read-queue-element-identifier>

<read-queue-queue> ::
    FROM <os-name-literal>

<read-queue-wait> ::
```

```
      WITH WAIT
    | WITH NO WAIT

<read-queue-element-identifier> ::
      ID <id-workspace-field-2>
    | IDENTIFIER <id-workspace-field-2>

<call-presentation-procedure> ::
    CALL PRESENTATION <presentation-procedure-identifier>
        IN <presentation-group-identifier>
        <call-presentation-with>
        <call-presentation-lists>

<call-presentation-with> ::
      <nil>
    | WITH <call-presentation-attribute>

<call-presentation-attribute> ::
      NO TRANSACTIONAL [WORK]
    | TRANSACTIONAL [WORK]
    | BROADCAST LIST <os-name-value>

<call-presentation-lists> ::
      <call-presentation-send-list>
    | <call-presentation-receive-list>
    | <call-presentation-transceive-list>

<call-presentation-send-list> ::
    SENDING <call-presentation-workspace-list>

<call-presentation-receive-list>::
    RECEIVING <call-presentation-workspace-list>

<call-presentation-transceive-list> ::
    SENDING <call-presentation-workspace-list>
    RECEIVING <call-presentation-workspace-list>

<call-presentation-workspace-list> ::
      <nil>
    | <workspace-identifier> , ...

<assignment> ::
    <assignment-workspace-field-list> = <expression>

<assignment-workspace-field-list> ::
    <workspace-field> ,...

<expression> ::
      <integer-expression>
    | <message-identifier>
    | <message-group-identifier>
    | <class-identifier>
```

```
     | <string-literal>
     | <uuid-string-literal>
     | <decimal-literal>
     | <octet-workspace-field>
     | <integer-workspace-field>
     | <text-workspace-field>
     | <decimal-workspace-field>
     | <record-workspace-field>
     | <array-workspace-field>
     | <uuid-workspace-field>

<audit> ::
    AUDIT <audit-information-list>

<audit-information-list> ::
    <audit-information> , ...

<audit-information> ::
      <workspace-field>
    | <string-literal>

<get-message> ::
    GET MESSAGE NUMBER <get-message-number> <get-message-group>
        <get-message-parts-list>
        <get-message-using>
        INTO <text-workspace-field-1>

<get-message-number> ::
      <message-identifier>
    | <integer-literal>
    | <integer-workspace-field-1>

<get-message-group> ::
      <nil>
    | IN <message-group-identifier>
    | IN <uuid-string-literal>
    | IN <uuid-workspace-field>

<get-message-parts-list> ::
      <nil>
    | <get-message-part> ...

<get-message-part> ::
      <get-message-source>
    | <get-message-length>
    | <get-message-language>

<get-message-source> ::
      SOURCE [IS] APPLICATION
    | SOURCE [IS] SYSTEM
    | SOURCE [IS] <integer-workspace-field-2>
```

```
<get-message-length> ::
    LENGTH [IS] <integer-workspace-field-3>

<get-message-language> ::
    LANGUAGE [IS] <get-message-language-expression>

<get-message-language-expression> ::
      <human-language-literal>
    | <s-text-workspace-field>

<get-message-using> ::
      <nil>
    | USING <get-message-argument-list>

<get-message-argument-list> ::
    <text-workspace-field-2> , ...

<reraise-exception>::
    RERAISE EXCEPTION

<restart-transaction> ::
    RESTART [TRANSACTION] [WITH] <exception-information>

<raise-exception> ::
    RAISE  <exception-information> <exception-rollback>

<exception-rollback> ::
      <nil>
    | [WITH] ROLLBACK [TRANSACTION]

<exception-information> ::
      [EXCEPTION] CLASS <exception-class-expression>
    | [EXCEPTION] CODE <exception-code-expression> <exception-group>

<exception-class-expression> ::
      <class-identifier>
    | <integer-workspace-field>
    | <integer-literal>

<exception-code-expression> ::
      <message-identifier>
    | <integer-workspace-field>
    | <integer-literal>

<exception-group> ::
      <nil>
    | IN <message-group-identifier>
    | IN <uuid-workspace-field>
    | IN <uuid-string-literal>

<exit-block> ::
    EXIT BLOCK <transaction-control>
```

```
<exit-task> ::
    EXIT TASK <transaction-control>

<go-to> ::
      GO TO <label-identifier> <transaction-control>
    | GOTO <label-identifier> <transaction-control>

<transaction-control> ::
      <nil>
    | WITH ROLLBACK [TRANSACTION]
    | WITH COMMIT [TRANSACTION]
```

Appendix E

Data Type Mapping

E.1 Introduction

This appendix is normative.

This appendix describes the mapping of STDL data types to their corresponding COBOL and C data types.

STDL data types must be mappable and transferable to and from C and COBOL data types.

This appendix does not specify how data is transferred between languages, nor does it specify call semantics for transferring data between languages.

E.2 Definition of Data Type Mapping

The data types to which data is possibly transferred and the data types mapping in respective languages are as follows:

In this definition, "-" indicates that the type cannot be handled in this language. For `n`, `m`, `a` and `b`, an appropriate decimal digit is designated, and for `id`, `number`, `rec` and `rec0`, an appropriate name is designated. An `M` indicates an integer which depends on the implementation for representing its data type.

1. Alphanumeric character string type:

   COBOL: `PIC X(n)`

   C: `char id[n]`

   STDL: `TEXT CHARACTER SET SIMPLE-LATIN SIZE n`

   The data transferred can be used directly as a character string of the alphanumeric character set.

2. Long binary integer type:

   COBOL: `PIC S9(9) USAGE BINARY`

   C: `long`
   (or equivalent `signed long`, `long int` or `signed long int`)

   STDL: `INTEGER`

   The data transferred can be used directly as an integer of the long binary integer type.

3. Fixed-point-number type:

   COBOL:
   ```
   PIC S9(a-b)V9(b) USAGE BINARY
       SIGN IS LEADING SEPARATE CHARACTER
   ```

   C: `char id[a+1]`

   STDL: `DECIMAL STRING SIZE a SCALE b`

   Excluding C, the data transferred can be used directly as a real number of the fixed decimal number type.

   In C, the data transferred is a character string in ISO 6093: 1985, signed NR1 format; that is, a row of one sign (or blank character) and digits which does not include characters indicating a decimal point. For that reason, an application program should convert the representation to a decimal-number.

4. Bit string type:

   COBOL: `PIC X`

   C: `unsigned char`

   STDL: `OCTET`

5. Kanji character string type:

   COBOL: `PIC N(n)`

   C: `char id[M]`

   STDL: `TEXT CHARACTER SET KANJI SIZE n`

Representing methods for *n* characters (including the size of storage area) may differ in accordance with the type of languages. For that reason, an application program should convert the representation.

6. Katakana character string type:

   COBOL: `PIC X(M)`

   C: `char id[M]`

   STDL: `TEXT CHARACTER SET KATAKANA SIZE n`

   Representing methods for *n* characters (including the size of storage area) may differ in accordance with the type of languages. For that reason, an application program should convert the representation.

7. ISO Latin-1 character string type:

   COBOL: `PIC X(n)`

   C: `char id[n]`

   STDL: `TEXT CHARACTER SET ISO-LATIN-1 SIZE n`

8. ISO Latin-2 character string type:

   COBOL: `PIC X(n)`

   C: `char id[n]`

   STDL: `TEXT CHARACTER SET ISO-LATIN-2 SIZE n`

9. ISO UCS-2 character string type:

   COBOL: `PIC N`

   C: `wchar_t id[n]`

   STDL: `TEXT CHARACTER SET ISO-UCS-2 SIZE n`

10. Array:

    <One dimension>

    COBOL: `OCCURS n <type>`

    C: `<type> id[n]`

    STDL `ARRAY SIZE n OF <type>`

    <Two dimensions>

    COBOL: `OCCURS n. OCCURS m <type>`

    C: `<type> id[n][m]`

    STDL: `ARRAY SIZE n OF ARRAY SIZE m OF <type>`

    Also, arrays of three dimensions or more are mapped as above.

11. Variable repetition array:

    COBOL: `OCCURS n TO m DEPENDING ON number <type>`

    C: Refer to the following (the number of repetitions is identified by the application program).

STDL: `ARRAY SIZE n TO m DEPENDING ON number OF <type>`

The *number* which indicates the number of repetitions is mapped as follows including the same record as an array:

COBOL:
```
01   rec.
         02 number PIC S9(9) USAGE BINARY.
         02 id OCCURS n TO m DEPENDING ON number <type>.
```

C:
```
struct rec {
    long int number ;
    <type> id[m] ; } ;
```

STDL:
```
RECORD rec
    number INTEGER ;
    id ARRAY SIZE n TO m DEPENDING ON number OF <type> ;
END;
```

Data type mapping between C and COBOL shall not be permitted.

12. Record:

COBOL: `02-49`
(All structures that have level 02 to 49 are allowed.)

C: `struct`

STDL: `RECORD`

Data type mapping between C and COBOL shall not be permitted.

Appendix F

Human Language Mapping

F.1 Introduction

This appendix is normative.

This appendix describes the allowed execution character sets for the various human languages used in:

- workspace fields that are the targets of <get-message> operations
- <string-literals> for <message-text>s in the <message-group-definition>.

An implementation is not required to support all of the languages listed in this appendix and can support additional languages. If an implementation supports a language, it must support the first execution character set specified for the language in Section F.2 on page 528. Note that these mappings only depend on the <language> part of the human language name.

The language names in this appendix are specified in ISO 639: 1988. The list of language names is derived from:

- Section 2, Field of Application, of ISO 8859-1
- Section 2, Field of Application, of ISO 8859-2
- Japanese (as defined within STDL).

F.2 Language Names and Execution Character Sets

The following table maps STDL language names to execution character sets:

Description	Language Name	Execution Character Set
Albanian	sq	ISO Latin 2 or ISO UCS-2
Czech	cs	ISO Latin 2 or ISO UCS-2
Danish	da	ISO Latin 1 or ISO UCS-2
Dutch	nl	ISO Latin 1 or ISO UCS-2
English	en	Simple Latin, ISO Latin 1, ISO Latin 2, Kanji or ISO UCS-2
Faeroese	fj	ISO Latin 1 or ISO UCS-2
Finnish	fi	ISO Latin 1 or ISO UCS-2
French	fr	ISO Latin 1 or ISO UCS-2
German	de	ISO Latin 1, ISO Latin 2 or ISO UCS-2
Hungarian	hu	ISO Latin 2 or ISO UCS-2
Icelandic	is	ISO Latin 1 or ISO UCS-2
Irish	ga	ISO Latin 1 or ISO UCS-2
Italian	it	ISO Latin 1 or ISO UCS-2
Japanese	ja	Kanji or ISO UCS-2
Norwegian	no	ISO Latin 1 or ISO UCS-2
Polish	pl	ISO Latin 2 or ISO UCS-2
Portugese	pt	ISO Latin 1 or ISO UCS-2
Rumanian	ro	ISO Latin 2 or ISO UCS-2
Serbocroation	sh	ISO Latin 2 or ISO UCS-2
Slovak	sk	ISO Latin 2 or ISO UCS-2
Slovene	sl	ISO Latin 2 or ISO UCS-2
Spanish	es	ISO Latin 1 or ISO UCS-2
Swedish	sv	ISO Latin 1 or ISO UCS-2

Appendix G

Encoding Rules of STDL Data Types

G.1 Introduction

This appendix is normative.

This appendix describes the mapping of STDL data types to their ASN.1 representation and describes any special data encoding rules. The encoding rules for ASN.1 defined data types are specified in ISO ASN.1 BER.

Special encoding rules are defined in Section G.3 on page 532.

The following UUID and version number is defined to represent the transfer syntax value to be used in the presentation negotiation for the STDL data types and their ASN.1 representations:

UUID: 92E66ACC-1CEB-11C9-8FED-08002B104860
Version: 1

Note that there is not a unique mapping of DCE IDL to ASN.1. The transfer syntax specified in this appendix represents a mapping of the subset of IDL used by STDL.

G.2 Mapping of APDU Data Types to ASN.1

This describes the ASN.1 data types for data exchanged between STDL implementations. This data corresponds to the ‘‘arguments’’ parameter of the RTI service primitives RTI-CALL-TASK and RTI-CALL-RESULT. The ‘‘arguments’’ parameter consists of any number of concatenated Stdl-Data-Type data elements.

The format, order and types of the data elements are defined in Chapter 5 on page 121.

```
Stdl-Data-Type ::= CHOICE {
    [0]  IMPLICIT Stdl-Octet,
    [1]  IMPLICIT Stdl-Integer,
    [2]  IMPLICIT Stdl-Text-Simple-Latin,
    [3]  IMPLICIT Stdl-Text-Katakana,
    [4]  IMPLICIT Stdl-Text-Kanji,
    [5]  IMPLICIT Stdl-Decimal-String,
    [6]  IMPLICIT Stdl-Octet-Array,
    [7]  IMPLICIT Stdl-Array,
    [8]  IMPLICIT Stdl-Record,
    [9]  IMPLICIT Stdl-Text-Iso-Latin-1,
    [10] IMPLICIT Stdl-Text-Iso-Latin-2,
    [11] IMPLICIT Stdl-Text-Iso-Ucs-2}

Stdl-Octet ::= OCTET STRING SIZE (1..1)
    STDL type "OCTET".
    A single octet.

Stdl-Integer ::= INTEGER (-2**31..2**31-1)
    STDL type "INTEGER".
    A 32 bit signed integer.

Stdl-Text-Simple-Latin ::= VisibleString
    STDL type "TEXT CHARACTER SET SIMPLE-LATIN".
    VisibleString refers to the ISO 646 IRV character set as
    indicated in ISO Latin 1.
    The ISO 646 IRV character set defines a 7-bit character encoding.
    For this usage, the transfer syntax sets the most significant bit
    (bit 8) of each octet to 0. To optimize transmission per
    formance, some or all of the trailing space characters are
    optionally not encoded (they can be "stripped"). The TxRPC protocol
    partner must space-fill the item to its specified length.

Stdl-Text-Katakana ::= OCTET STRING
    STDL type "TEXT CHARACTER SET KATAKANA".
    To optimize transmission performance, some or all of the
    trailing space characters are optionally not encoded
    (they can be "stripped"). The TxRPC protocol partner must
    space-fill the item to its specified length.

Stdl-Text-Kanji ::= OCTET STRING
    STDL type "TEXT CHARACTER SET KANJI".
    To optimize transmission performance, some or all of the
    trailing space characters are optionally not encoded
    (they can be "stripped"). The TxRPC protocol partner must
```

```
    space-fill the item to its specified length.

Stdl-Decimal-String ::= PrintableString
    STDL type "DECIMAL STRING".
    The PrintableString character set is defined in ISO Latin 1.
    The legal set of characters from the PrintableString character
    set are limited to the characters "Digits", "Plus sign",
    "Hyphen", and -- "Full stop" as defined in Table 5.

Stdl-Octet-Array ::= OCTET STRING
    STDL type "ARRAY SIZE n OF OCTET".
    Encoding for STDL defined arrays of data type octet.

Stdl-Array ::= SEQUENCE OF Stdl-Data-Type
    STDL type "ARRAY".
    Stdl-Data-Type may not include "Stdl-Octet".

Stdl-Record ::= SEQUENCE OF Stdl-Data-Type
    STDL type "RECORD".

Stdl-Text-Iso-Latin-1 ::= GraphicString
    STDL type "TEXT CHARACTER SET ISO-LATIN-1".
    VisibleString refers to the ISO 8859-1 IRV character set
    as indicated in ISO Latin 1.
    To optimize transmission performance, some or all of the
    trailing space characters are optionally not encoded (they can be
    "stripped"). The TxRPC protocol partner must space-fill the
    item to its specified length.

Stdl-Text-Iso-Latin-2 ::= GraphicString
    STDL type "TEXT CHARACTER SET ISO-LATIN-2".
    GraphicString refers to the ISO 8859-2 IRV character set
    as indicated in ISO Latin 2. To optimize transmission performance,
    some or all of the trailing space characters are optionally not
    encoded (they can be "stripped"). The TxRPC protocol partner
    must space-fill the item to its specified length.

Stdl-Text-Iso-Ucs-2 ::= GraphicString
    STDL type "TEXT CHARACTER SET ISO-UCS-2" GraphicString refers
    to the ISO UCS-2 character set as indicated in the ISO UCS-2
    standard. To optimise transmission performance, some or all of
    the trailing space characters are optionally not encoded (they
    can be "stripped"). The TxRPC protocol partner must
    space-fill the item to its specified length.
```

G.3 APDU Data Type Encoding Rules

G.3.1 Katakana Character Set Encoding Rules

The encoding rules for the Katakana character set are defined in JIS Katakana. The encoding rules defined in JIS Katakana for the 8-bit environment are used.

G.3.2 Kanji Character Set Encoding Rules

Each character consists of two octets encoded as specified by the encoding rules defined in JIS Kanji as follows. JIS Kanji-defined characters are encoded in the least seven bits of the octet tuple with the most significant bit of each octet set to 0. JIS X0212-1990 defined characters are encoded in the least seven bits of the octet tuple with the most significant bit of each octet set to 1.

G.3.3 UUID Encoding Rules

For encoding into ASN.1, a <workspace-field> of type UUID shall be mapped into a record with the following definition:

```
TYPE UUID IS RECORD
    TIME-LOW IS INTEGER;
    TIME-MID IS SHORT-INTEGER;
    TIME-HI-AND-VERSION IS SHORT-INTEGER;
    CLOCK-SEQ-HI-AND-RESERVED IS OCTET;
    CLOCK-SEQ-LOW IS OCTET;
    NODE-ID IS ARRAY SIZE 6 OF OCTET;
END RECORD;
```

The fields of the UUID are defined in the referenced X/Open TxRPC specification. The SHORT-INTEGER is a 16-bit integer which is transmitted as an INTEGER. This data type does not otherwise exist within STDL.

Appendix H

Use of DCE RPC Protocol

This appendix describes the use of DCE RPC protocol for STDL. This appendix is normative.

This appendix incorporates terminology from various sources, including X/Open DCE RPC and X/Open TxRPC. The referenced X/Open TxRPC specification incorporates terminology from the OSF DCE RPC specification and from the ISO OSI TP specification. The resulting terminology alignment problem is resolved to the extent possible.

H.1 Introduction

The information in this appendix is based on the the Internet protocols (TCP/IP) and the DCE RPC. This appendix makes use of the DCE RPC Specification. This appendix makes use of the RTI Model, RTI Protocol, and RTI Services as described in the referenced X/Open TxRPC specification.

H.2 Scope

H.2.1 Introduction

This section identifies those subjects that are within the scope of the appendix and those subjects that are not within the scope of the appendix.

H.2.2 Subjects within the Scope of the Appendix

This appendix defines a mapping of a portion of the RTI Service directly onto native DCE RPC for the purpose of specifying a protocol for non-transactional remote task invocation over TCP/IP for external client programs. It does this by:

- defining a subset of the RTI Model
- defining a subset of the RTI Service Primitives
- defining a mapping of the subsetted service primitives onto DCE RPC according to the subsetted model.

H.2.3 Subjects Not within the Scope of the Appendix

This appendix does not define an abstract service or an abstract model. This appendix subsets the existing RTI Service and RTI Model. This appendix does not provide an Application Programming Interface (API).

This appendix does not constitute a design specification or specify any concrete interfaces suitable for use in a design specification. In other words, this appendix does not constrain the implementation of entities and interfaces within a computer system for the purposes of satisfying a particular implementation or product that makes use of the RTI model subset, service primitive subset or service primitive subset mapping in this appendix.

H.3 RTI Model Subset

H.3.1 Introduction

The basic structure of this subset of the RTI Model is derived directly from the RTI Model presented in the referenced X/Open TxRPC specification.

The RTI Service Primitives, as specified in the referenced X/Open TxRPC specification, define the RTI Service. The boundary between an RTI Service User (RTI-SUI) and the protocol machinery that provides the RTI Service is referred to as the RTI Service Boundary. This service boundary is invariant to an RTI-SUI regardless of the underlying communications protocol or protocols used to implement the services which compose this boundary.

Taking advantage of the invariance of the RTI Service Boundary and RPC oriented client/server design of RTI, this appendix specifies the mapping the of a complete, non-transactional subset of the RTI Service Primitives directly onto DCE RPC.

H.3.2 RTI Model Subset Components

This subset of the RTI Model makes use of only the RTI-SUI and RTI-PM components of the RTI Model.

H.3.2.1 RTI Service User Invocation

This component is used as defined in the referenced X/Open TxRPC specification.

H.3.2.2 RTI Protocol Machine

DCE RPC is used to implement a non-transactional RTI Protocol Machine (RTI-PM) and thereby provide the necessary RTI Service Primitives to deliver a non-transactional RTI Service. For this subset of the RTI Model, the RTI-PM is defined as the protocol machine of DCE RPC.

Figure H-1 shows a non-transactional RTI model for TCP/IP.

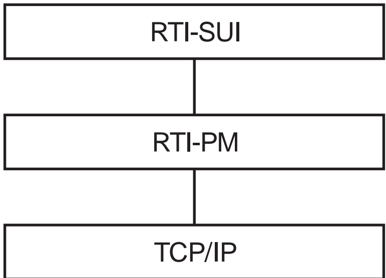


Figure H-1 RTI Model Subset

H.3.3 RTI Model Subset Component Relationships

The relationship between the RTI-SUI and the RTI-PM for this subset of the RTI Model is the same as defined in the referenced X/Open TxRPC specification.

H.4 RTI Service Primitive Subset

H.4.1 Introduction

In order to provide a complete non-transactional RTI Service, each of the service primitives contained in the Kernel and Non-transactional functional units of the RTI Service must be mapped to the appropriate DCE RPC stub calls and services of the DCE RPC Application Programming Interface (service primitives). As shown in Figure H-2, this is a mapping between an abstract specification (RTI service primitives) and a concrete specification (DCE RPC services). The mapping of RTI semantics onto DCE-RPC services, together with using the same protocol machine, implies that the same generated protocol is associated with each specification.

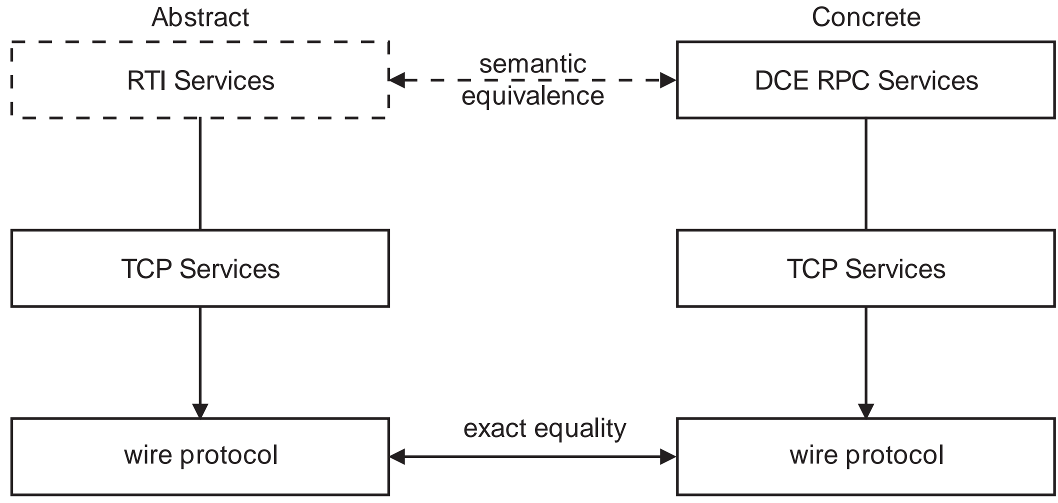


Figure H-2 Relationship between RTI and DCE RPC Services

These RTI Service Primitives constitute a subset of the entire set of RTI Service Primitives. This subset is specified in Table H-1. Refer to the referenced X/Open TxRPC specification for details of these service primitives.

Service Name	Client Primitives	Server Primitives	Functional Unit
RTI-ESTABLISH-CONTEXT	req		Kernel
RTI-CALL-TASK	req	ind	Kernel
RTI-CANCEL-CALL	req	ind	Kernel
RTI-CALL-FAILURE	ind		Kernel
RTI-CALL-RESULT	ind	req	Kernel
RTI-RELEASE-CONTEXT	req/ind		Non-trans

Table H-1 RTI Service Primitives Required for Non-transactional RTI Service

H.5 RTI Service Primitive Mapping

H.5.1 Introduction

This section details the mapping of the Kernel and Non-transactional RTI service primitives onto the service primitives of DCE RPC. A summary of this mapping, including references to the appropriate section of the appendix that details the mapping of each service primitive is specified in Table H-2.

RTI Service Primitive	DCE RPC Service Primitive(s)
RTI-ESTABLISH-CONTEXT.req	rpc_binding_from_string_binding
RTI-CALL-TASK.req	Client local procedure call to RPC stub
RTI-CALL-TASK.ind	Server local procedure invocation by RPC stub
RTI-CANCEL-CALL.req	pthread_cancel
RTI-CANCEL-CALL.ind	Server local thread cancel
RTI-CALL-FAILURE.ind	Thread exception handler or RPC return parameters
RTI-CALL-RESULT.req	Return from server local procedure call to RPC stub
RTI-CALL-RESULT.ind	Return to client local procedure call from RPC stub
RTI-RELEASE-CONTEXT.req	rpc_binding_free
RTI-RELEASE-CONTEXT.ind	Return to client local procedure call from RPC stub

Table H-2 Summary of RTI to DCE RPC Service Primitive Mapping

H.5.2 Kernel Functional Unit

H.5.2.1 RTI-ESTABLISH-CONTEXT.req

The semantics of RTI-ESTABLISH-CONTEXT.req are mapped to the following DCE RPC binding services[4]:

DCE RPC Service	rpc_binding_from_string_binding		
	Data Type	**Name**	**Contents**
input parameter	unsigned_char_t	*string_binding	string binding derived from RTI-AP-Title
output parameter	rpc_binding_handle_t	*binding	Returned binding handle
	unsigned32	*status	OK or invalid param

Table H-3 rpc_binding_from_string_binding service

4. There is no need in the non-transactional case to use DCE RPC's context handle mechanisms. Section H.6 on page 542 shows an alternative implementation that uses the DCE Name Service.

DCE RPC Service	rpc_binding_set_auth_info		
	Data Type	**Name**	**Contents**
input parameters	rpc_binding_handle_t	binding	Binding handle
	unsigned_char_t	*server_princ_name	Destination server name
	unsigned32	protect_level	"rpc_c_protect_level_connect"
	unsigned32	authn_svc	"rpc_c_authn_none"
	rpc_auth_identity_handle_t	auth_identity	Client_Name Client_Authenticator
	unsigned32	authz_svc	"rpc_c_authz_name"
output parameters	unsigned32	*status	OK or invalid

Table H-4 rpc_binding_set_auth_info service

RTI-AP-Title must derive all of the information necessary to find the server (i.e., the IP address and DCE RPC instance id of the server)[5].

The RTI Interface-Version-Major, Interface-Version-Minor, and Interface-UUID are implicitly contained in the RPC call since this information is contained in the RPC stub itself.

Context-Type must be set to NON-TRANSACTIONAL.

Client-Authenticator-Type must be set to CUSTOMER-WRITTEN-SECURITY (Kerberos)[6].

Use of "rpc_c_protect_level_connect" implies that authentication is on establishment of the connection only.

Use of "rpc_c_authn_none" implies that the server performs no authentication.

H.5.2.2 RTI-CALL-TASK.req

The semantics of RTI-CALL-TASK.req are mapped to a client local procedure call to the DCE RPC client stub specified by the parameters on the RTI-CALL-TASK.req. The encoding of stub data is as same as defined in Appendix G on page 529.

H.5.2.3 RTI-CALL-TASK.ind

The semantics of RTI-CALL-TASK.ind are mapped to an invocation of a server local procedure by the DCE RPC server stub. There is a correspondence between the server stub to the Operation-number, Interface-UUID, Interface-Version-Major and Interface-Version-minor parameters specified on the RTI-CALL-TASK.ind and RTI-ESTABLISH-CONTEXT services.

---

5. The format of the string binding derived from RTI-AP-Title is one of the following:

```
Object-UUID@Protocol-Sequence:Network-Address[endpoint,Option...]
Object-UUID@Network-Address[endpoint,Option...]
```

The first form allows the use of a Protocol-Sequence specification. In both forms, zero or more options may follow the endpoint specification.

6. DCE RPC currently does not support authorization unless a particular implementation defines its own security mechanism. A routine, rpc_binding_set_auth_info, is planned in future releases of DCE RPC.

H.5.2.4 RTI-CANCEL-CALL.req

The semantics of RTI-CANCEL-CALL.req are mapped a client local pthread_cancel of the thread of an outstanding client local procedure call to the DCE RPC client stub. The format of pthread_cancel is as follows.

DCE Service	pthread_cancel		
	Data Type	**Name**	**Contents**
input parameters	pthread_t	*thread	id of the thread executing the RPC call to be cancelled.

Table H-5 pthread_cancel service

The DCE RPC runtime on the client side detects the pthread_cancel and sends a cancel PDU to the server.

H.5.2.5 RTI-CANCEL-CALL.ind

The semantics of RTI-CANCEL-CALL.ind are mapped to the arrival of a cancel PDU at the server.

A call cancel is automatically executed in the server provided that *general cancel delivery* is enabled (which is the default case) and the server reaches a DCE RPC-defined *cancellation point.* The server can be cancelled at any time if *asynchronous cancelability* is enabled.

Alternatively, a server can use the thread exception mechanism to detect the cancel, should the server's thread package support exceptions.

H.5.2.6 RTI-CALL-FAILURE.ind

The semantics of RTI-CALL-FAILURE.ind are mapped to the DCE RPC status reporting mechanism. There are two ways to detect failures:

- The application has declared a threads exception handler (TRY block).
- The application has declared in an Attribute Configuration File (ACF) in which fault_status is returned as a parameter for each procedure[7].

  When a call returns and *fault_status* has any value other than rpc_s_ok a failure has occurred.

H.5.2.7 RTI-CALL-RESULT.req

The semantics of RTI-CALL-RESULT.req are mapped to the return from a server local procedure call to the server DCE RPC stub. The encoding of stub data is as same as defined in Appendix G on page 529.

7. The values for these status parameters are mutually exclusive, therefore they can both be returned through the use of a single parameter.

Implementor Note:

An implementation that is unable to handle more than one operation on a single association shall send a shutdown PDU before sending the response PDU. Also, the context_handle attribute in IDL shall not be used.

H.5.2.8 RTI-CALL-RESULT.ind

The semantics of RTI-CALL-RESULT.ind are mapped to an error-free return to a client local procedure call from the client DCE RPC stub.

H.5.3 Non-transactional Functional Unit

H.5.3.1 RTI-RELEASE-CONTEXT.req

The semantics of RTI-RELEASE-CONTEXT.req are mapped to the following service.

DCE RPC Service	rpc_binding_free		
	Data Type	**Name**	**Contents**
input parameters	rpc_binding_handle_t	*binding	Binding handle
output parameters	unsigned32	*status	OK or invalid binding

Table H-6 rpc_binding_free service

H.5.3.2 RTI-RELEASE-CONTEXT.ind

The semantics of RTI-RELEASE-CONTEXT.ind are mapped to a return to a client local procedure call from the client DCE RPC stub with a shutdown PDU.

H.6 Optional Use of DCE Name Service

H.6.1 Introduction

This section shows how the DCE Name Service can be used for RTI over TCP/IP. In place of the rpc_binding_from_string_binding call in the RTI-ESTABLISH-CONTEXT request primitive, a sequence of RPC name service primitives can be used, as shown in Table H-7.

RTI Service Primitive	DCE RPC Service Primitive(s)
RTI-ESTABLISH-CONTEXT.req	rpc_ns_binding_import_begin rpc_ns_binding_import_next rpc_ns_binding_import_done rpc_binding_set_auth_info

Table H-7 Summary of RTI to DCE RPC Name Service Mapping

H.6.2 Name Service Primitives

DCE RPC Service	rpc_ns_binding_import_begin		
	Data Type	**Name**	**Contents**
input parameters	unsigned32	entry_name_syntax	"rpc_c_ns_syntax_default"
	unsigned_char_t	*entry_name	name space entry for the starting point of the interface look up
	rpc_if_handle_t	if_handle	Interface-Version-Major Interface-Version-Minor Interface-UUID
	uuid_t	*obj_uuid	NULL
output parameters	rpc_ns_handle_t	*import_context	Returned name service handle
	unsigned32	*status	OK or invalid param

Table H-8 rpc_ns_binding_import_begin service

DCE RPC Service	rpc_ns_binding_import_next		
	Data Type	**Name**	**Contents**
input parameters	rpc_ns_handle_t	*import_context	name service handle from rpc_ns_binding_import_begin
output parameters	rpc_binding_handle_t	*binding	Binding handle
	unsigned32	*status	OK or name service

Table H-9 rpc_ns_binding_import_next service

DCE RPC Service	rpc_ns_binding_import_done		
	Data Type	**Name**	**Contents**
input parameters	rpc_ns_handle_t	*import_context	Name service handle from rpc_ns_binding_import_begin
output parameters	unsigned32	*status	OK or name service

Table H-10 rpc_ns_binding_import_done

Interface-Version-Major, Interface-Version-Minor, and Interface-UUID should be the standard format that is generated by DCE RPC. As the interface handle, this information is used as the search criteria through the name space starting at the specified name space entry.

H.7 Mapping of APDU Data Types to NDR

The mapping of STDL data types to NDR is done as if the following mappings of STDL data types were mapped to DCE RPC IDL data types:

- The STDL data type OCTET is mapped to the DCE RPC IDL data type byte.
- The STDL data type INTEGER is mapped to the DCE RPC IDL data type long.
- The STDL data type TEXT CHARACTER SET SIMPLE-LATIN SIZE x is mapped to the DCE RPC IDL data type char [x] where x is the size of the STDL TEXT data type.
- The STDL data type TEXT CHARACTER SET ISO-LATIN-1 SIZE x is mapped to the DCE RPC IDL data type ISO_LATIN_1 [x] where x is the size of the STDL TEXT data type.
- The STDL data type TEXT CHARACTER SET ISO-LATIN-2 SIZE x is mapped to the DCE RPC IDL data type byte [x] where x is the size of the STDL TEXT data type and each byte is encoded using the ISO 8859-2 IRV character set as indicated in ISO Latin 2.
- The STDL data type TEXT CHARACTER SET KATAKANA SIZE x is mapped to the DCE RPC IDL data type byte [x] where x is the size of the STDL TEXT data type and each byte is encoded using the Katakana character set as defined in JIS Katakana using the 8-bit environment.
- The STDL data type TEXT CHARACTER SET KANJI SIZE x is mapped to the DCE RPC IDL data type ISO_MULTI_LINGUAL [x] where x is the size of the STDL TEXT data type and each character is encoded into two bytes using the Kanji character sets as defined in JIS Kanji. JIS Kanji-defined characters are encoded in the least seven bits of the octet tuple with the most significant bit of each octet set to 0. JIS X0212-1990-defined characters are encoded in the least seven bits of the octet tuple with the most significant bit of each octet set to 1.
- The STDL data type TEXT CHARACTER SET ISO-UCS-2 SIZE x is mapped to the DCE IDL data type UNICODE and the bytes are encoded using the ISO UCS-2 character set as indicated in the ISO UCS-2 standard.
- The STDL data type DECIMAL STRING SIZE x is mapped to the DCE RPC IDL data type char [x+1] where x is the size of the STDL DECIMAL STRING data type.
- The STDL data type UUID is mapped to the DCE RPC IDL data type uuid_t.
- The STDL data type ARRAY SIZE x of d is mapped to the DCE RPC IDL array data type.
- The STDL data type ARRAY SIZE x TO y DEPENDING ON f OF d is mapped to the DCE RPC IDL array data type using the length_is attribute to refer to the depending field.
- The STDL data type RECORD is mapped to the DCE RPC IDL data type struct. The fields in the RECORD are mapped to fields in the DCE RPC IDL struct in the same order as they appear in the RECORD.

Index

<absolute-time-value> 117
in <submit-trigger> 242
<array-type> 121
<array-workspace-field>
in <boolean-comparison> 110
in <workspace-field> 91
<assignment> 184
<audit> 186
<boolean-and-expression> 108
<boolean-comparison> 109
<boolean-expression> 107
<boolean-negated-condition> 112
<boolean-term> 109
<call-presentation-procedure> 188
<call-procedure> 193
<call-task> 196
<cancel-submit> 201
<character-set-identifier> 80, 130, 502
<class-identifier> 80
<composable-task-interface> 163
<composable-task> 169
<concurrent-block> 203
<control-field> 205
<data-type-definition> 124
<data-type-identifier> 85
<data-type> 123
<decimal-literal> 92
<decimal-string-type> 125
<delta-time-value> 118
<dequeue-record> 207
<enqueue-record> 212
<exception-handler> 214
<exception-information> 215
<exit-block> 216
<exit-task> 217
<external-identifier> 84
<field-identifier> 87
<get-message> 218
<go-to> 222
<if> 223
<integer-expression> 113
<integer-literal> 92
<integer-type> 127
<internal-identifier> 86
<label-identifier> 87
<message-definition> 138
<message-group-attribute-list> 134
<message-group-definition> 133
<message-group-identifier> 85
<message-group-language> 135
<message-group-uuid> 136
<message-identifier> 85
<message-list> 137
<message-parameter> 95
<message-text> 139
<multiplicative-expression> 114
<noncomposable-task-interface> 164
<noncomposable-task> 171
<octet-type> 128
<operator> 94
<os-name-value> 119
<pre-define> 102
<pre-if> 103
<pre-include> 105
<pre-undef> 106
<presentation-group-attribute-list> 142
<presentation-group-identifier> 85
<presentation-group-specification> 141
<presentation-initialization-procedure> 143
<presentation-procedure-identifier> 85
<presentation-procedure-interface> 147
<presentation-procedure-list> 146
<presentation-source-language> 144
<presentation-termination-procedure> 145
<primary-expression> 116
<processing-group-identifier> 86
<processing-group-specification> 151
<processing-procedure-identifier> 86
<processing-procedure-interface> 154
<processing-procedure-list> 153
<processing-source-language> 152
<punctuator> 95
<raise-exception> 225
<read-queue-record> 226
<record-type> 129
<reraise-exception> 231
<restart-transaction> 232
<restartability> 175
<select-first> 233
<send-display> 176
<statement-block> 235
<statement-list> 182

<string-literal> 92-93
<submit-repeat> 241
<submit-task> 237
<submit-trigger> 242
<task-argument-list-definition> 172
<task-attribute-list> 174
<task-definition> 168
<task-group-attribute-list> 158
<task-group-identifier> 86
<task-group-specification> 157
<task-group-uuid-attribute> 159
<task-group-version-attribute> 160
<task-identifier> 86
<task-interface-argument-list-definition> 165
<task-interface-list> 162
<text-type> 130
<transaction-block> 244
<transaction-control> 246
<unary-expression> 115
<uuid-type> 131
<while> 247
<workspace-field> 89
<workspace-identifier> 80, 87
<workspace-list-definition> 177
absolute time value
  in TRIGGER-INFORMATION 430
actions
  client TP system 53
  exception generated 65
  state machines 326
  submitter TP system 52
ACTIVATION-TIME 180
AP-EXECUTION-FAULT 65, 138, 185-186, 191
  .... 194, 199-200, 206, 209, 213, 221, 224-225
  .... 229, 232, 234, 240, 247, 270, 280, 311
  .... 314, 440-441, 465, 494
  exception class value 486
AP-INCOMPLETE-ERROR 493, 499
  exception class value 486
AP-INVOCATION-FAULT 191, 194, 198, 239
  .... 311, 314, 343, 420, 441, 465, 489
  exception class value 486
AP-PROCESSING-FAULT 311, 314, 497, 499
  exception class value 486
AP-RESPONSE-FAULT ....
  .... 192, 195, 199, 311, 314, 493
  exception class value 486
APDU data types
  encoding rules 532
  mapping to ASN.1 530
  mapping to NDR 544
application
  moving 297
application execution 319
  modelling method 321
application flow 11
architectural constants 481
architectural limits 481
arguments
  presentation procedure 275, 285
  top-level processing procedure 274, 284
arithmetic operator 94
ARRAY 121
assignment operator 94
asynchronous cancelability 540
AUDIT 186
audit
  access to log 72
  data type definition 487
  log 72, 303
audit log 72
  entity 303
authentication 56
  call mechanism type 307
  data 302, 309
  forwarding mechanism type 307
  incoming 309
  information list 302, 309
  mechanism type 302, 309
  outgoing 301
  user 317
authentication data 57
authentication methanism type 57
authentication user 56
authorization 56, 58
BLOCK 235
BLOCK WITH CONCURRENT EXECUTION ....
  .... 23, 203, 511
BLOCK WITH TRANSACTION 23, 29, 244, 313
BNF 501
  data type definition 501
  message definition language 503
  presentation group specification 504
  processing group specification 506
  task definition 509
  task group specification 507
broadcast list 45, 81
BROADCAST LIST 188
broadcast list
  entity 303
broadcast send procedure 257

C
- data type mapping ... 279
- exception information ... 280
- exceptions returned ... 281
- header files ... 281, 287
- raising exceptions ... 280
- use of ... 279

CALL ... 23
call ... 330
- state machine ... 433
- state tables ... 445

CALL PRESENTATION ... 23, 188, 518
CALL PROCEDURE ... 193, 513
CALL TASK ... 23, 196, 514
CALL-CANCEL ... 335, 375, 435-436, 438, 471
CALL-CANCELED ... 334, 373, 435-436, 439, 469
CALL-DONE ... 334, 336, 373, 408, 413 ... 435-436, 439, 470
CALL-EXCEPTION ... 334, 336, 374 ... 408, 413, 435-436, 440, 469
CALL-FORWARD-TASK ... 414, 435-436, 438, 442
CALL-RELEASE-CONTEXT ... 336, 342 ... 409, 413, 436, 438, 449
CALL-TASK ... 335, 377-379, 434 ... 436, 438-439, 441-443, 470
CALL-TASK-MARSHALED ... 409, 434 ... 438-439, 441-443
CALL-TASK-MARSHALLED ... 438
calling task
- RPC interfaces ... 424

CANCEL SUBMIT ... 23
cancel submit ... 52
CANCEL SUBMIT ... 201
cancellation point ... 540
CASE NOMATCH ... 205, 233
categories ... 34
character set ... 75
CHARACTER SET ... 130
CLASS ... 138
client program ... 41, 47, 263
- customer-written ... 263
- exceptions ... 263
- menu client programs ... 264

client program call ... 328-329
- state machine ... 333
- state tables ... 337

client TP system ... 47
COBOL
- COPY files ... 271, 277
- data type mapping ... 269
- exception information ... 270
- exceptions returned ... 271
- raising exceptions ... 270
- STDL identifier conversion ... 276
- use of ... 269

comment character ... 95
comment text ... 95
committing ... 37
common elements ... 75
communication ... 328, 330
- RPC interfaces ... 423
- state machine ... 423

communications ... 21
COMPOSABLE TASK ... 12, 23
composable task ... 41, 47
COMPOSABLE TASK ... 163, 169, 507, 509
computer language ... 81
CONC-CANCEL ... 375, 396, 398-399
CONC-CANCELED ... 373, 396, 400
CONC-DONE ... 373, 396, 400
CONC-EXCEPTION ... 374, 396, 400
CONC-EXECUTE ... 376, 396, 398-399
CONC-EXECUTE-THREAD ... 404
CONC-EXIT-TASK ... 374, 396, 400
CONC-EXIT-TASK-COMMIT ... 374, 396, 400
CONC-EXIT-TASK-ROLLBACK ... 375, 396, 400
CONC-THREAD-CANCEL ... 397-398, 400, 404
CONC-THREAD-CANCELED ... 397, 399, 405
CONC-THREAD-DONE ... 397, 399, 405
CONC-THREAD-EXCEPTION ... 397, 399-400, 405
CONC-THREAD-EXECUTE ... 322 ... 397-398, 400, 404
CONC-THREAD-EXIT ... 400
CONC-THREAD-EXIT-COMMIT ... 400
CONC-THREAD-EXIT-ROLLBACK ... 400
CONC-THREAD-EXIT-TASK ... 397, 399, 405
CONC-THREAD-EXIT-TASK-COMMIT ... 397, 399, 405
CONC-THREAD-EXIT-TASK-ROLLBACK ... 397, 399, 405
concepts ... 37
concurrent block ... 41, 330
- state machine ... 396
- state tables ... 402

concurrent execution ... 15
concurrent task ... 31
concurrent thread ... 330
- state machine ... 404
- state tables ... 406

conformance ... 32, 292
constants
- architectural ... 481

context
  presentation procedure....261
CONTEXT-TYPE-NOT-SUPPORTED....437, 440
CONTROL FIELD....205
conventions....30, 325
  state machine....325
CP-CALL-TASK....333-335
CP-CANCEL-TASK....334
CP-EXIT....334
CP-TASK-DONE....333, 336
CP-TASK-EXCEPTION....333, 335-336
customer-defined identifier....84
customer-written client....47
customer-written client program....263
customer-written presentation procedure....261
data type definition....296
  BNF....501
  EXCEPTION-INFORMATION....430
  exceptions....430
  format/layout....296
data type definitions....121
data type mapping....523
  C....279
  COBOL....269
database access....18
DCE name service
  optional use of....542
  primitives....542
DCE RPC....31
  use of....533
DECIMAL STRING....125
DEFINE....102
definitions....31, 291
DEPENDENT....196, 201, 237
DEQUEUE....207
dequeue operation....61
destination....81
  entity....306
dialogue....54
dialogue client system....54
dialogue server system....54
dirty read....38
display....44, 82
  access....19
  entity....305
distributed transaction....41
distribution list....82
  entity....306
durable attributes....73
durable resource....37
durable resources....37
dynamic attribute....298
dynamic entity....298
ELIF....103
encoding rules
  APDU data types....532
  Kanji....532
  Katakana....532
  STDL data types....529
  UUID....532
END [IF]....223
enqueue operation....61
ENQUEUE RECORD....212, 515
ENQUEUE-DONE....438
ENQUEUE-EXCEPTION....438, 443
ENQUEUE-RELEASE-CONTEXT....439
entity....31, 299
ENV-EXECUTION-ERROR....335, 342
  ....408, 412, 418-419, 440, 496
  exception class value....486
ENV-EXECUTION-FAULT....311, 314
  ....419, 440-441, 465, 495
  exception class value....486
ENV-INVOCATION-ERROR....301, 311
  ....315, 343, 408, 412, 420, 440-441, 465, 491
  exception class value....486
ENV-INVOCATION-FAULT....311, 314
  ....343, 420, 440-441, 465, 490
  exception class value....486
ENV-UNSPECIFIED-FAULT....440-441, 498
  exception class value....486
environmental information....59, 293
error handling rules
  processing procedure....256
exception....62
  application-defined....74
  generation....64
exception class....62
EXCEPTION CLASS....215
exception class....489
exception class value....486
exception code....63
EXCEPTION CODE....215
exception code group....63
EXCEPTION HANDLER....23
exception handler....64
EXCEPTION HANDLER....214
exception handling....16
  customer-written clients....65
  external clients....65
  fatal transaction in tasks....68
  non-transaction in tasks....66

permanent transaction in tasks 67
presentation procedures 66
processing procedures..66
transient transaction in tasks 67
exception level...63
current..63
propagated 63
exception location..63
exception source 63
exception type 62
EXCEPTION-CLASS 178, 432
EXCEPTION-CODE 178, 432
EXCEPTION-CODE-GROUP 179, 432
EXCEPTION-INFO-WORKSPACE................64, 80
....178, 180, 214, 231
EXCEPTION-INFORMATION...................424, 426
....430-431, 437, 442
data type definition...430
EXCEPTION-LEVEL 179, 432
EXCEPTION-PROCEDURE........................179, 432
EXCEPTION-PROCEDURE-GROUP 179, 432
EXCEPTION-SOURCE 179, 432
EXCEPTION-TYPE 431-432, 437, 442
exceptions
client program...263
data type definition...430
processing procedure 266
execution
TP entity 298
execution character set 76
execution context..323
state machine..323
execution environment 26, 59
execution information 298
EXIT BLOCK 23, 216
EXIT TASK...23, 217
external client 48
processing procedure 266
external client task..48
external identifier...84-85
external task dequeuer 48
fatal transaction exception.....................................62
fatal transaction in tasks
exception handling...68
FATAL-EXECUTION-FAULT.............311, 314, 499
exception class value 486
FATAL-TIMEOUT-FAULT 311, 314
exception class value 486
fault_status 540
file
entity 308
file access 18
processing procedure........................254, 272, 282
file descriptor..295
C procedures 295
COBOL procedures...295
file transfer
processing procedure 256
FIRST RECORD 207, 226
forms system 31
functional capabilities..32
general cancel delivery...540
general rules
presentation procedure....................................257
GET MESSAGE..23, 218
GO TO...23, 222
HOLD...242
human language 82
human language mapping 527
identifier 60, 88
IDENTIFIER 201, 207, 212, 227, 237
identifier
customer-defined...84
external 84
internal..84, 86
syntactical reference 88
IF THEN ELSE...23
INCLUDE..105
incoming authentication
entity 309
INDEPENDENT....................................196, 201, 237
indexed file..70
indication
service primitive..322
INITIALIZATION PROCEDURE.......................143
initialization procedures 262
instance context
state machine..322
instances
state machine..321
INTEGER...127
integer expressions..113
interface definition 423
INTERFACE-PERMANENTLY-UNAVAILABLE..
....437, 440
INTERFACE-TEMPORARILY-UNAVAILABLE....
....437, 440
INTERFACE-UNKNOWN..........................437, 440
internal client task 47
internal identifier..84, 86
internal task dequeuer...48
INVALID-INPUT-ERROR 499

exception class value 486
ISO6093: 1985 524
isolation level 38
Kanji
encoding rules 532
Katakana
encoding rules 532
Kernel functional unit 538
LANGUAGE 135, 218
LENGTH 218
levels 35
limits
architectural 481
literals
workspace field 92
mapping
terms and concepts 325
marshalling 423
menu client program 48
menu client programs
client program 264
MESSAGE 133
message definition language 133
BNF 503
message group 74
message parameter 95
model 328
NATIONAL TEXT 130
nested processing procedure 250
arguments 251
NEXT RECORD 226
NO-OUTPUT-ERROR 465
exception class value 486
non-composable task 41, 47
non-repeatable read 38
non-restartable task 67
non-transaction exception 62
non-transaction exception handler 64
non-transaction tasks
exception handling 66
non-transactional dialogue 54
non-transactional functional unit 541
non-transactional operation 37
non-transactional presentation procedures 19
non-transactional resource 37
non-transactional resource manager 37
non-transactional statement list 329
state machine 349
state tables 360
non-transactional task call 13
NONTX-STATEMENT-CANCEL 342
........349-350, 352, 354
NONTX-STATEMENT-CANCELED 341
........349, 352, 354
NONTX-STATEMENT-DONE ...341, 349, 352, 354
NONTX-STATEMENT-EXCEPTION 341
........349, 352, 354
NONTX-STATEMENT-EXECUTE 345
........349-350, 353, 356
NONTX-STATEMENT-EXIT-BLOCK 341
........349, 353, 355
NONTX-STATEMENT-EXIT-TASK 341
........350, 353, 355
NONTX-STATEMENT-GOTO 350-351
........353, 355, 377
NONTX-STATEMENT-GOTO-COMMIT 377
NONTX-STATEMENT-GOTO-ROLLBACK 377
NOT 112
NOT TRANSACTIONAL 177
OCTET 128
OS names 81
permanent transaction exception 62
permanent transaction in tasks
exception handling 67
PERMANENT-COMMUNICATION-FAILURE....
........437, 440
phantom 38
predefined words 80
preprocessing 99
PRES-CANCEL 375, 455, 474-475
PRES-CANCELED 373, 455-456, 474, 476
PRES-DONE 373, 455-456, 474, 476
PRES-EXCEPTION 374, 455, 474, 476
PRES-EXECUTE 383, 456, 474-475
PRES-TERMINATE 342, 475
presentation context 260
presentation group 45
PRESENTATION GROUP 141
presentation group
entity 309
presentation group context 45
presentation group handle 261
presentation group specification 141
BNF 504
presentation procedure..45, 257, 275, 285, 331, 474
arguments 275, 285
context 261
general rules 257
restrictions 275, 285
send 257
state tables 478
vendor-supplied forms 262

presentation procedures
  exception handling....66
principal....56
  TP system....56
  user....56
principal name....57
PRIVATE....177
private workspace....69
PROC-CANCEL....375, 468-469
PROC-CANCELED....373, 468, 471
PROC-DONE....373, 468, 471
PROC-EXCEPTION....374, 468, 471-472
PROC-EXECUTE....377, 468-470
PROCEDURE....147, 154
procedure....294
procedure execution....328, 331, 468
processing group....43
PROCESSING GROUP....151
processing group
  entity....310
processing group context....43
processing group specification....151
  BNF....506
processing procedure....43, 249
  ....265, 272, 282, 331, 468
  error handling rules....256
  exceptions....266
  external....48
  external client....266
  file access....254, 272, 282
  file transfer....256
  internal....48
  nested....43, 250
  restrictions....272, 282
  SQL access....255
  state tables....473
  top-level....43, 250
processing procedures
  context....252
  exception handling....66
Profile A....31
Profile B....31
PROTOCOL-MACHINE-FAILURE....437, 440
PROTOCOL-VERSION-NOT-SUPPORTED....
  ....437, 440
queue
  cursor....60
  head....60
  tail....60
RAISE....225
RAISE EXCEPTION....23
READ QUEUE....226
read queue operation....61
REASON-NOT-SPECIFIED....438, 440
receive presentation procedure....259
RECEIVING....147, 188
RECORD....129
RECORD BY IDENTIFIER....207, 226
RECORD BY KEY....207
record queue....60, 72, 83
  context....60
  dequeue operation....61
  enqueue operation....61
  entity....304
  operations....61
  ordering....60
  position....60
  read queue operation....61
  record....60
record queuing....60
relational operator....94
relative file....70
remote authentication....56-57
REPEATING....241
request
  service primitive....322
REQUEST-TIMEOUT-ERROR....408, 465
  exception class value....486
RERAISE EXCEPTION....23, 231
reserved words....79
resource....69
resource manager....328, 331, 454, 458
  access....4
  state tables....467
RESTART TRANSACTION....23, 232
RESTARTABLE....175
restartable task....67
restrictions
  presentation procedure....275, 285
  processing procedure....272, 282
RM-CANCEL....375, 418, 458, 461, 463, 471
RM-CANCELED....373, 417, 461, 464, 469
RM-DEQUEUE-RECORD....373, 460, 463-464
RM-DEQUEUE-RECORD-FIRST....381, 459, 463
RM-DEQUEUE-RECORD-ID....382, 460, 463
RM-DEQUEUE-RECORD-KEY....381, 459, 463
RM-DEQUEUE-RECORD-NEXT....382
RM-DEQUEUE-TASK....407-409, 413-414
  ....458-459, 463-464
RM-DO-IO....461, 463, 472
RM-DO-P1....449, 461, 463
RM-DO-P2....449, 461, 463

RM-DONE....................374, 418, 461, 464-465, 470
RM-ENQUEUE-RECORD 373, 381, 459, 463
RM-ENQUEUE-TASK 373, 379-381
....410, 418, 458, 463
RM-EXCEPTION..................374, 418, 461, 465, 470
RM-P1-DONE......................................448, 462, 466
RM-P1-FAIL...448, 462, 466
RM-P2-DONE......................................448, 462, 466
RM-READ-QUEUE-RECORD.....373, 460, 463-464
RM-READ-QUEUE-RECORD-FIRST.....................
....382, 460, 463
RM-READ-QUEUE-RECORD-ID.......382, 460, 463
RM-READ-QUEUE-RECORD-KEY...................382
RM-READ-QUEUE-RECORD-KEY-FIRST............
....460, 463
RM-READ-QUEUE-RECORD-KEY-NEXT............
....382, 460, 463
RM-READ-QUEUE-RECORD-NEXT........460, 463
RM-REMOVE-TASK 381, 459, 463
RM-ROLLBACK...................................449, 462, 464
RM-ROLLBACK-DONE......................448, 462, 466
RM-TRANSACTIONAL-SEND..........383, 460, 463
ROLLBACK-IN-PROGRESS.......................438, 440
rolling back 37
root task 41
root TP system...41
RPC-ACCESS-VIOLATION 440
RPC-CANCEL..440
RPC-FLOATING-DIVIDE-BY-ZERO 440
RPC-FLOATING-ERROR 440
RPC-FLOATING-OVERFLOW...........................440
RPC-FLOATING-UNDERFLOW 441
RPC-INSUFFICIENT-RESOURCES 441
RPC-INTEGER-DIVIDE-BY-ZERO 441
RPC-INTEGER-OVERFLOW.............................441
RPC-INVALID-OPERATION-NUMBER...........441
RPC-INVOCATION-FAILURE 441
RPC-MARSHALING-ERROR 441
RPC-PROTOCOL-ERROR.................................441
RPC-REASON-NOT-SPECIFIED.......................441
RR-CANCEL 375, 454-455
RR-CANCELED.....................................373, 454-455
RR-DONE..374, 454-456
RR-EXCEPTION 374, 454-456
RR-RECEIVE 382, 454-455
RR-RESET..356, 454-455
RTI...31, 292, 328
  Protocol machine 535
RTI model...535
  subset component relationships 536
  subset components 535
RTI service
  primitive mapping 538
  primitive subset 537
  user invocation 535
RTI-CALL-FAILURE 320, 437-438, 440
RTI-CALL-FAILURE.ind 540
RTI-CALL-RESULT...............437-438, 440, 442-443
RTI-CALL-RESULT.ind......................................541
RTI-CALL-RESULT.req 540
RTI-CALL-RESULTS..424
RTI-CALL-TASK 322, 424
....................................434-435, 438-439, 441, 443
RTI-CALL-TASK.ind..539
RTI-CALL-TASK.req..539
RTI-CANCEL-CALL 438, 442
RTI-CANCEL-CALL.ind.....................................540
RTI-CANCEL-CALL.req.....................................540
RTI-COMMIT-TRANS 448-449
RTI-END-TRANS 450
RTI-ESTABLISH-CONTEXT 442
RTI-ESTABLISH-CONTEXT.req 538
RTI-HEURISTIC-REPORT 448
RTI-PREPARE-TRANS..449
RTI-RELEASE-CONTEXT 438, 443
RTI-RELEASE-CONTEXT.ind 541
RTI-RELEASE-CONTEXT.req 541
RTI-ROLLBACK-TRANS.............................449-450
RTI-SERVICE-UNKNOWN.........................437, 440
RTI-TRANS-COMPLETE 448
RTI-TRANS-DONE...449
RTI-TRANS-READY 449-450
SCALE...125
security..56
  authentication 56
  authentication data 57
  authentication mechanism type 57
  authentication user 56
  authorization..56, 58
  principal 56
  principal name 57
  remote authentication...................................56-57
  user authentication 57
security
  principal 56
SELECT FIRST..233
send
  presentation procedure.....................................257
SEND DISPLAY 176
SENDING 147, 188
sequential file..71
serial execution 38

serialisable execution 38
server task 47
server task group 47
server TP system 47
SERVER-TASK-GROUP 430
SERVER-TASK-INFORMATION 428
service primitive
  indication 322
  request 322
SHARED 177
SHARED READ 177
SHARED UPDATE 177
shared workspace 69
SOURCE 218
source character set 75
  STDL Alphanumeric 75
  STDL Kanji 76
source file 97
SOURCE LANGUAGE 144, 152
source line 97
SQL access
  processing procedure 255
SQL database 69
SQL environment
  entity 311
Standard C or C 31, 291
Standard COBOL or COBOL 31, 291
Standard ISAM 31, 292
Standard programming languages 31, 291
Standard SQL or SQL 31, 291
Standard XFTAM 31, 292
state machine 321
  call 330, 433
  categories 328
  client program call 329, 333
  communication 330, 423
  concurrent block 330, 396
  concurrent thread 330, 404
  conventions 325
  execution context 323
  instance context 322
  instances 321
  non-transactional statement list 329, 349
  overall relationships 329
  presentation procedure 331, 474
  procedure execution 331, 468
  processing procedure 331, 468
  relationships among 331
  resource manager 331, 454, 458
  submitted task operations 330, 407
  task 329, 338
  task dequeuer 330, 407
  task enqueuer 330, 416
  task execution 329, 338
  task forwarder 330, 412
  thread context 323
  transaction 330, 447
  transactional receive 331, 454
  transactional statement list 330, 369
state machine category
  client program call 328
  communication 328
  procedure execution 328
  resource manager 328
  RTI 328
  submitted task operations 328
  task execution 328
state tables
  call 445
  client program call 337
  concurrent block 402
  concurrent thread 406
  non-transactional statement list 360
  presentation procedure 478
  processing procedure 473
  resource manager 467
  task 347
  task dequeuer 411
  task enqueuer 422
  task forwarder 415
  transaction 452
  transactional receive 457
  transactional statement list 386
static attribute 298
static entity 298
STDL
  application flow 11
  client program 10
  introduction 3
  language components 22
  language" .... 3"
  major features 3
  mapping to client/server models 5
  mapping to TP monitors 9
  mapping to X/Open DTP model 8
  model 5
  source files 22
  the specification 26
  transaction control 11
STDL Alphanumeric 75
STDL data types
  encoding rules 529

STDL identifier
  conversion to C 286
  conversion to COBOL word 276
STDL Kanji 76
stream file 71
stub procedure 267
submission request 316
submit task 12
SUBMIT TASK 23, 237
SUBMIT-TASK-GROUP 430
submitted task operations 328, 330
  state machine 407
SUBMITTED-TASK-ARGUMENT 426
submitter task 50
submitter TP system 50
syntax notation 29, 291
syntax reference
  workspace field 91
system failure 37
system manager 298
system workspace 69
system workspace identifiers 80
SYSTEM-INFO-WORKSPACE 80, 180
SYSTEM-SHUTDOWN-FAULT 465
  exception class value 486
TASK 23
task 41
TASK 164, 171
task 329
  state machine 338
  state tables 347
task call 11, 47
task definition
  BNF 509
task definition language 167
task dequeuer 50, 330
  state machine 407
  state tables 411
task enqueuer 330
  RPC interfaces 426
  state machine 416
  state tables 422
task execution 328-329
  state machine 338
task forwarder 330
  state machine 412
  state tables 415
task group 42
  entity 312
task group context 42
TASK GROUP SPECIFICATION 157
task group specification 157
  BNF 507
  transfer 297
task invocation 47
task queue 51, 72
  entity 315
task submit 50
TASK-CALL-INFORMATION 220, 335
  ....338, 379-381, 424-425, 429, 434-435
TASK-CANCEL 338-339, 341, 439
TASK-CANCELED 338, 342, 438
TASK-DONE 338, 342, 438-439, 442
TASK-EXCEPTION 339, 342-343, 438, 440, 443
TASK-EXECUTE 338-341, 439
TASK-FORWARD-INFORMATION 416
  ....426-428, 435, 458-459
TASK-GROUP-NAME 180
TASK-GROUP-UUID 409, 428
TASK-NAME 180
TASK-OPERATION-ID 409, 428
TASK-RELEASE-CONTEXT 339, 344, 439
TERMINATION PROCEDURE 145
termination procedures 262
TEXT 138
text messages 16
thread context 323
  state machine 323
token 78
top-level processing procedure 250, 274, 284
  arguments 250, 274, 284
  entity 310
TP entity 298
TP system 40, 56, 299
  entity 301
  model 40
TP system
  entity **299**
TRANS-COMMIT 354, 409, 413, 418, 447-448
TRANS-COMMIT-DONE 341, 352
  ....408, 413, 417, 447, 449
TRANS-DISTRIBUTED 439, 447-448
TRANS-ROLLBACK 344-345, 355
  ....410, 414, 420, 447-448
TRANS-ROLLBACK-DONE 341, 353
  ....408, 413, 418, 447, 449
TRANS-ROLLBACK-RECEIVED 341
  ....353, 408, 413, 418, 447, 449
TRANS-START 345, 356-357, 409
  ....413, 420, 447, 449
transaction 37, 330, 447, 452
transaction block 41

transaction control.....11
transaction exception.....62
transaction exception handler.....64
transaction processing entity.....298
transaction tree.....41
TRANSACTIONAL.....172, 177, 188
transactional attributes.....73
transactional dialogue.....54
transactional operation.....37
transactional presentation procedures.....20
transactional receive.....72, 331, 454
state tables.....457
transactional resource.....37
transactional resource manager.....37
transactional resources.....37
transactional send.....72
transactional send procedure.....257
transactional statement list.....330
state machine.....369
state tables.....386
transactional submit task.....14
transactional task call.....13
transactions.....11
transceive presentation procedure.....259
transient transaction exception.....62
transient transaction in tasks
exception handling.....67
TRANSIENT-COMMUNICATION-FAILURE.....437, 440
translation.....293
data type definition.....296
file descriptor.....295
procedures.....294
source files.....293
SQL Access information.....296
translation environment.....59
translation information.....293
translation time.....293
TX-STATEMENT-CANCEL.....342, 354, 369-370, 373, 375, 405
TX-STATEMENT-CANCELED.....341, 352, 369, 373, 376, 404
TX-STATEMENT-DONE.....341, 352, 369, 374, 376, 404
TX-STATEMENT-EXCEPTION.....341, 352, 369, 374, 376, 404-405
TX-STATEMENT-EXECUTE.....345, 356-357, 369-370, 374, 377, 405
TX-STATEMENT-EXIT-BLOCK.....341, 353, 369, 374, 376, 404
TX-STATEMENT-EXIT-BLOCK-COMMIT.....353, 370, 374, 376
TX-STATEMENT-EXIT-BLOCK-ROLLBACK
indication......374
TX-STATEMENT-EXIT-BLOCK-ROLLBACK.....353, 370, 376
TX-STATEMENT-EXIT-TASK.....341, 353, 370, 374, 376, 404
TX-STATEMENT-EXIT-TASK-COMMIT.....353, 370, 374, 376, 405
TX-STATEMENT-EXIT-TASK-ROLLBACK.....353, 370, 374, 376, 405
TX-STATEMENT-GOTO.....353, 370, 372, 375
TX-STATEMENT-GOTO-COMMIT...353, 370, 375
TX-STATEMENT-GOTO-ROLLBACK.....353, 370, 375
TXN-FAILURE-ERROR.....356, 408, 412, 419, 492
exception class value.....486
TXN-INCOMPLETE-ERROR.....408, 494, 499
exception class value.....486
TXN-TIMEOUT-ERROR.....313, 344, 356, 408, 412, 419, 493
exception class value.....486
TxRPC Specification.....31, 292
TYPE.....124
UNDEF.....106
unmarshalling.....423
user.....56
user authentication.....56-57, 317
user profile.....317
entity.....317
UUID.....131, 136, 159
encoding rules.....532
VALUE.....138
values.....117
vendor-supplied forms
presentation procedure.....262
VERSION.....160
volatile attribute.....298
volatile entity.....298
WHILE.....23, 247
white space.....78
WITH COMMIT.....23, 246
WITH NO WAIT.....207, 227
WITH ROLLBACK.....23, 246
WITH ROLLBACK TRANSACTION.....225
WITH WAIT.....207, 227
WORKSPACE.....177
workspace field.....89
literals.....92
syntax reference.....91