IBM Communications Server for AIX

# CPI-C プログラマーズ・ガイド

*V6.3*

SC88-6954-02
(英文原典：SC31-8591-02)

IBM Communications Server for AIX

# CPI-C プログラマーズ・ガイド

*V6.3*

SC88-6954-02
(英文原典：SC31-8591-02)

**お願い**

本書および本書で紹介する製品をご使用になる前に、 205 ページの『付録 E. 特記事項』に記載されている情報をお読みください。

本書は、IBM Communications Server for AIX バージョン 6.3 (プログラム番号 5765-E51) および新しい版またはテクニカル・ニュースレターで明記されていない限り、以降のすべてのリリースおよびモディフィケーションに適用されます。

本マニュアルに関するご意見やご感想は、次の URL からお送りください。今後の参考にさせていただきます。

http://www.ibm.com/jp/manuals/main/mail.html

なお、日本 IBM 発行のマニュアルはインターネット経由でもご購入いただけます。詳しくは

http://www.ibm.com/jp/manuals/ の「ご注文について」をご覧ください。

(URL は、変更になる場合があります)

お客様の環境によっては、資料中の円記号がバックスラッシュと表示されたり、バックスラッシュが円記号と表示されたりする場合があります。

**原　典：** SC31–8591–02
IBM Communications Server for AIX
CPI-C Programmer's Guide
V6.3

**発　行：** 日本アイ・ビー・エム株式会社

**担　当：** ナショナル・ランゲージ・サポート

第1刷 2005.10

この文書では、平成明朝体™W3、平成明朝体™W7、平成明朝体™W9、平成角ゴシック体™W3、平成角ゴシック体™W5、および平成角ゴシック体™W7を使用しています。この(書体*)は、(財) 日本規格協会と使用契約を締結し使用しているものです。フォントとして無断複製することは禁止されています。

注* 平成明朝体™W3、平成明朝体™W7、平成明朝体™W9、平成角ゴシック体™W3、
平成角ゴシック体™W5、平成角ゴシック体™W7

# 目次


**表 . . . . . xv**

**図 . . . . . xvii**

**本書について . . . . . xix**
本書の対象読者 . . . . . xix
本書の使用法 . . . . . xix
本書の構成 . . . . . xix
表記上の規則 . . . . . xx
シンボルの意味 . . . . . xxi
新しい機能 . . . . . xxi
詳細について . . . . . xxii

**第 1 章 概念 . . . . . 1**
CPI-C とは . . . . . 1
CS/AIX CPI-C オプション・セット・サポート . . . . . 1
プログラム間の通信 . . . . . 2
論理装置 6.2 . . . . . 3
セッション . . . . . 3
会話 . . . . . 3
コンテンション . . . . . 3
特性 . . . . . 4
CPI-C コール . . . . . 4
会話プロセス . . . . . 4
会話タイプ . . . . . 4
単純なマップ式会話 . . . . . 5
会話の開始 . . . . . 5
データの送信 . . . . . 5
データの受信 . . . . . 6
会話の終了 . . . . . 6
確認処理 . . . . . 6
同期レベルの設定 . . . . . 7
確認要求の送信 . . . . . 7
確認要求の受信 . . . . . 7
確認要求への応答 . . . . . 7
会話の割り振り解除 . . . . . 7
会話状態 . . . . . 7
プログラム側から見た会話 . . . . . 9
状態の変化 . . . . . 9
状態チェック . . . . . 9
会話状態の変更 . . . . . 9
初期状態 . . . . . 10
受信状態への変更 . . . . . 10
送信状態への変更 . . . . . 11
サイド情報 . . . . . 11
基本会話 . . . . . 12
論理レコード . . . . . 12
エラー・ログ・データ . . . . . 13
複数会話 . . . . . 13
会話セキュリティーの概要 . . . . . 14

複数会話の会話セキュリティー . . . . . 14
検査済み会話セキュリティー . . . . . 15
非ブロッキング操作 . . . . . 16
CPI-C と LU 6.2 . . . . . 19

**第 2 章 CPI-C アプリケーションの作成 . . . . . 21**
CPI-C コールの要約 . . . . . 21
会話の開始 . . . . . 21
データの送信 . . . . . 23
データの受信 . . . . . 24
ASCII と EBCDIC 間のデータ変換 . . . . . 25
データ受信の確認とエラーの報告 . . . . . 25
非ブロッキング・モードでのコールの発行 . . . . . 26
ブロッキング・モードでのコールの発行 . . . . . 27
情報の取得 . . . . . 28
会話の終了 . . . . . 29
サイド情報の管理 . . . . . 29
初期会話特性 . . . . . 30
サイド情報 . . . . . 34
ローカル LU 別名 . . . . . 34
パートナー LU 名 . . . . . 34
パートナー・プログラムのタイプと名前 . . . . . 34
モード名 . . . . . 35
会話セキュリティー・タイプ . . . . . 35
セキュリティー・ユーザー ID とパスワード . . . . . 35
アプリケーション指定のサイド情報 . . . . . 35
構成 . . . . . 36
ローカル TP 名の指定 . . . . . 37
Specify_Local_TP_Name . . . . . 37
コンテキスト . . . . . 37
APPCTPN 環境変数 . . . . . 37
デフォルト値 . . . . . 38
ローカル LU の指定 . . . . . 38
Set_Local_LU_Name . . . . . 39
コンテキスト . . . . . 39
APPCLLU 環境変数 . . . . . 39
サイド情報 . . . . . 40
デフォルトのローカル LU . . . . . 40
制御点 LU . . . . . 40
プログラムの開始方法 . . . . . 40
呼び出し対象プログラム: 自動開始の場合 . . . . . 40
呼び出し対象プログラム: ユーザー開始の場合 . . . . . 41
AIX または Linux に関する考慮事項 . . . . . 41
CPI-C ヘッダー・ファイル . . . . . 42
マルチプロセス . . . . . 42
CPI-C アプリケーションのコンパイルとリンク . . . . . 42
Java CPI-C に関する考慮事項 . . . . . 43
Java CPI-C クラスの使用 . . . . . 43
使用例 . . . . . 44
Java CPI-C アプリケーションのコンパイルとリンク . . . . . 46
Java CPI-C アプリケーションの実行 . . . . . 46
Windows に関する考慮事項 . . . . . 46
Windows CPI-C ファイル . . . . . 46
関数のプロトタイプ . . . . . 47
複数プロセスと複数会話 . . . . . 47
Windows 関数コール . . . . . 47

ブロッキング・コール . . . . . 48
アプリケーションの終了 . . . . . 49
CPI-C アプリケーションのコンパイルとリンク . . . . . 49
移植可能なアプリケーションの作成 . . . . . 50

**第 3 章 CPI-C コール . . . . . 53**
CPI-C コールに対して提供される情報 . . . . . 53
データ・タイプ . . . . . 53
データ構造 . . . . . 54
記号定数 . . . . . 54
ストリング . . . . . 54
戻りパラメーターの妥当性 . . . . . 55
Windows 関数コールに対して提供される情報 . . . . . 55
Accept_Conversation (cmaccp). . . . . 55
関数コール . . . . . 55
Java CPI-C の関数コール . . . . . 55
指定パラメーター . . . . . 56
戻りパラメーター . . . . . 56
発行時の状態 . . . . . 56
状態の変化 . . . . . 56
使用上の注意 . . . . . 57
Accept_Incoming (cmacci) . . . . . 57
関数コール . . . . . 58
Java CPI-C の関数コール . . . . . 58
指定パラメーター . . . . . 58
戻りパラメーター . . . . . 58
発行時の状態 . . . . . 59
状態の変化 . . . . . 59
使用上の注意 . . . . . 59
Allocate (cmallc) . . . . . 60
関数コール . . . . . 60
Java CPI-C の関数コール . . . . . 60
指定パラメーター . . . . . 60
戻りパラメーター . . . . . 61
発行時の状態 . . . . . 61
状態の変化 . . . . . 61
使用上の注意 . . . . . 62
Cancel_Conversation (cmcanc). . . . . 62
関数コール . . . . . 63
Java CPI-C の関数コール . . . . . 63
指定パラメーター . . . . . 63
戻りパラメーター . . . . . 63
発行時の状態 . . . . . 63
状態の変化 . . . . . 63
使用上の注意 . . . . . 63
Check_For_Completion (cmchck) . . . . . 64
関数コール . . . . . 64
指定パラメーター . . . . . 64
戻りパラメーター . . . . . 64
発行時の状態 . . . . . 65
状態の変化 . . . . . 65
使用上の注意 . . . . . 65
Confirm (cmcfm) . . . . . 65
関数コール . . . . . 66
Java CPI-C の関数コール . . . . . 66
指定パラメーター . . . . . 66

戻りパラメーター . . . . . 66
発行時の状態 . . . . . 67
状態の変化 . . . . . 67
使用上の注意 . . . . . 68
Confirmed (cmcfmd) . . . . . 68
関数コール . . . . . 68
Java CPI-C の関数コール . . . . . 68
指定パラメーター . . . . . 69
戻りパラメーター . . . . . 69
発行時の状態 . . . . . 69
状態の変化 . . . . . 69
使用上の注意 . . . . . 70
Convert_Incoming (cmcnvi) . . . . . 71
関数コール . . . . . 71
Java CPI-C の関数コール . . . . . 71
指定パラメーター . . . . . 71
戻りパラメーター . . . . . 72
発行時の状態 . . . . . 72
状態の変化 . . . . . 72
使用上の注意 . . . . . 72
Convert_Outgoing (cmcnvo) . . . . . 72
関数コール . . . . . 72
Java CPI-C の関数コール . . . . . 73
指定パラメーター . . . . . 73
戻りパラメーター . . . . . 73
発行時の状態 . . . . . 74
状態の変化 . . . . . 74
使用上の注意 . . . . . 74
Deallocate (cmdeal) . . . . . 74
関数コール . . . . . 74
Java CPI-C の関数コール . . . . . 74
指定パラメーター . . . . . 75
戻りパラメーター . . . . . 75
発行時の状態 . . . . . 76
状態の変化 . . . . . 76
使用上の注意 . . . . . 76
Delete_CPIC_Side_Information (xcmdsi) . . . . . 77
関数コール . . . . . 77
指定パラメーター . . . . . 77
戻りパラメーター . . . . . 77
発行時の状態 . . . . . 77
状態の変化 . . . . . 78
使用上の注意 . . . . . 78
Extract_Conversation_Context (cmectx) . . . . . 78
関数コール . . . . . 78
Java CPI-C の関数コール . . . . . 78
指定パラメーター . . . . . 78
戻りパラメーター . . . . . 78
発行時の状態 . . . . . 79
状態の変化 . . . . . 79
使用上の注意 . . . . . 79
Extract_Conversation_Security_Type (xcecst) . . . . . 80
関数コール . . . . . 80
指定パラメーター . . . . . 80
戻りパラメーター . . . . . 80
発行時の状態 . . . . . 81

状態の変化 . . . . . 81
Extract_Conversation_Security_User_ID (cmecsu) . . . . . 81
Extract_Conversation_Security_User_ID (xcecsu) . . . . . 82
Extract_Conversation_State (cmecs) . . . . . 82
関数コール . . . . . 82
Java CPI-C の関数コール . . . . . 82
指定パラメーター . . . . . 82
戻りパラメーター . . . . . 83
発行時の状態 . . . . . 83
状態の変化 . . . . . 83
Extract_Conversation_Type (cmect) . . . . . 83
関数コール . . . . . 83
Java CPI-C の関数コール . . . . . 83
指定パラメーター . . . . . 84
戻りパラメーター . . . . . 84
発行時の状態 . . . . . 84
状態の変化 . . . . . 84
Extract_CPIC_Side_Information (xcmesi) . . . . . 84
関数コール . . . . . 85
指定パラメーター . . . . . 85
戻りパラメーター . . . . . 85
発行時の状態 . . . . . 87
状態の変化 . . . . . 87
使用上の注意 . . . . . 87
Extract_Local_LU_Name (cmelln) . . . . . 87
関数コール . . . . . 88
Java CPI-C の関数コール . . . . . 88
指定パラメーター . . . . . 88
戻りパラメーター . . . . . 88
発行時の状態 . . . . . 88
状態の変化 . . . . . 89
使用上の注意 . . . . . 89
Extract_Maximum_Buffer_Size (cmembs) . . . . . 89
関数コール . . . . . 89
Java CPI-C の関数コール . . . . . 89
指定パラメーター . . . . . 89
戻りパラメーター . . . . . 89
発行時の状態 . . . . . 90
状態の変化 . . . . . 90
Extract_Mode_Name (cmemn) . . . . . 90
関数コール . . . . . 90
Java CPI-C の関数コール . . . . . 90
指定パラメーター . . . . . 90
戻りパラメーター . . . . . 91
発行時の状態 . . . . . 91
状態の変化 . . . . . 91
Extract_Partner_LU_Name (cmepln) . . . . . 91
関数コール . . . . . 91
Java CPI-C の関数コール . . . . . 91
指定パラメーター . . . . . 92
戻りパラメーター . . . . . 92
発行時の状態 . . . . . 92
状態の変化 . . . . . 92
Extract_Security_User_ID (cmesui または cmecsu) . . . . . 92
関数コール . . . . . 93
Java CPI-C の関数コール . . . . . 93

指定パラメーター . . . . . 93
戻りパラメーター . . . . . 93
発行時の状態 . . . . . 94
状態の変化 . . . . . 94
使用上の注意 . . . . . 94
Extract_Sync_Level (cmesl) . . . . . 94
関数コール . . . . . 94
Java CPI-C の関数コール . . . . . 94
指定パラメーター . . . . . 95
戻りパラメーター . . . . . 95
発行時の状態 . . . . . 95
状態の変化 . . . . . 95
Extract_TP_Name (cmetpn) . . . . . 95
関数コール . . . . . 96
Java CPI-C の関数コール . . . . . 96
指定パラメーター . . . . . 96
戻りパラメーター . . . . . 96
発行時の状態 . . . . . 97
状態の変化 . . . . . 97
Flush (cmflus) . . . . . 97
バッファーに入れられるデータのソース . . . . . 97
関数コール . . . . . 97
Java CPI-C の関数コール . . . . . 97
指定パラメーター . . . . . 97
戻りパラメーター . . . . . 98
発行時の状態 . . . . . 98
状態の変化 . . . . . 98
Initialize_Conversation (cminit) . . . . . 98
関数コール . . . . . 99
Java CPI-C の関数コール . . . . . 99
指定パラメーター . . . . . 99
戻りパラメーター . . . . . 99
発行時の状態 . . . . . 100
状態の変化 . . . . . 100
使用上の注意 . . . . . 100
Initialize_For_Incoming (cminic) . . . . . 100
関数コール . . . . . 100
Java CPI-C の関数コール . . . . . 101
指定パラメーター . . . . . 101
戻りパラメーター . . . . . 101
発行時の状態 . . . . . 101
状態の変化 . . . . . 101
Prepare_To_Receive (cmptr) . . . . . 101
関数コール . . . . . 102
Java CPI-C の関数コール . . . . . 102
指定パラメーター . . . . . 102
戻りパラメーター . . . . . 102
発行時の状態 . . . . . 103
状態の変化 . . . . . 103
使用上の注意 . . . . . 104
Receive (cmrcv) . . . . . 104
プログラムでのデータの受信方法 . . . . . 105
関数コール . . . . . 105
Java CPI-C の関数コール . . . . . 105
指定パラメーター . . . . . 106
戻りパラメーター . . . . . 106

発行時の状態 . . . 110
状態の変化 . . . 110
使用上の注意 . . . 113
Release_Local_TP_Name (cmrltp) . . . 114
関数コール . . . 114
Java CPI-C の関数コール . . . 114
指定パラメーター . . . 114
戻りパラメーター . . . 114
発行時の状態 . . . 115
状態の変化 . . . 115
使用上の注意 . . . 115
Request_To_Send (cmrts) . . . 115
パートナー・プログラムのアクション . . . 115
ローカル・プログラムがデータを送信できる時点 . . . 115
関数コール . . . 116
Java CPI-C の関数コール . . . 116
指定パラメーター . . . 116
戻りパラメーター . . . 116
発行時の状態 . . . 116
状態の変化 . . . 117
使用上の注意 . . . 117
Send_Data (cmsend) . . . 117
関数コール . . . 118
Java CPI-C の関数コール . . . 118
指定パラメーター . . . 118
戻りパラメーター . . . 118
発行時の状態 . . . 120
状態の変化 . . . 120
使用上の注意 . . . 120
Send_Error (cmserr) . . . 121
関数コール . . . 121
Java CPI-C の関数コール . . . 121
指定パラメーター . . . 121
戻りパラメーター . . . 121
発行時の状態 . . . 124
状態の変化 . . . 124
使用上の注意 . . . 125
Set_Conversation_Context (cmsctx) . . . 126
関数コール . . . 126
Java CPI-C の関数コール . . . 126
指定パラメーター . . . 126
戻りパラメーター . . . 126
発行時の状態 . . . 127
状態の変化 . . . 127
使用上の注意 . . . 127
Set_Conversation_Security_Password (cmscsp) . . . 127
関数コール . . . 128
Java CPI-C の関数コール . . . 128
指定パラメーター . . . 128
戻りパラメーター . . . 128
発行時の状態 . . . 129
状態の変化 . . . 129
使用上の注意 . . . 129
Set_Conversation_Security_Password (xcscsp) . . . 129
Set_Conversation_Security_Type (cmscst) . . . 130
関数コール . . . 130

Java CPI-C の関数コール . . . . . 130
指定パラメーター . . . . . 130
戻りパラメーター . . . . . 131
発行時の状態 . . . . . 132
状態の変化 . . . . . 132
使用上の注意 . . . . . 132
Set_Conversation_Security_Type (xcscst) . . . . . 132
Set_Conversation_Security_User_ID (cmscsu) . . . . . 132
関数コール . . . . . 132
Java CPI-C の関数コール . . . . . 132
指定パラメーター . . . . . 133
戻りパラメーター . . . . . 133
発行時の状態 . . . . . 134
状態の変化 . . . . . 134
使用上の注意 . . . . . 134
Set_Conversation_Security_User_ID (xcscsu) . . . . . 134
Set_Conversation_Type (cmsct) . . . . . 134
関数コール . . . . . 135
Java CPI-C の関数コール . . . . . 135
指定パラメーター . . . . . 135
戻りパラメーター . . . . . 135
発行時の状態 . . . . . 136
状態の変化 . . . . . 136
使用上の注意 . . . . . 136
Set_CPIC_Side_Information (xcmssi) . . . . . 136
関数コール . . . . . 136
指定パラメーター . . . . . 137
戻りパラメーター . . . . . 139
発行時の状態 . . . . . 139
状態の変化 . . . . . 139
使用上の注意 . . . . . 140
Set_Deallocate_Type (cmsdt) . . . . . 140
関数コール . . . . . 140
Java CPI-C の関数コール . . . . . 140
指定パラメーター . . . . . 140
戻りパラメーター . . . . . 142
発行時の状態 . . . . . 142
状態の変化 . . . . . 142
使用上の注意 . . . . . 142
Set_Error_Direction (cmsed) . . . . . 142
関数コール . . . . . 143
Java CPI-C の関数コール . . . . . 143
指定パラメーター . . . . . 143
戻りパラメーター . . . . . 143
発行時の状態 . . . . . 144
状態の変化 . . . . . 144
使用上の注意 . . . . . 144
Set_Fill (cmsf) . . . . . 144
関数コール . . . . . 144
Java CPI-C の関数コール . . . . . 145
指定パラメーター . . . . . 145
戻りパラメーター . . . . . 145
発行時の状態 . . . . . 146
状態の変化 . . . . . 146
使用上の注意 . . . . . 146
Set_Local_LU_Name (cmslln) . . . . . 146

関数コール . . . . . 146
Java CPI-C の関数コール . . . . . 146
指定パラメーター . . . . . 147
戻りパラメーター . . . . . 147
発行時の状態 . . . . . 147
状態の変化 . . . . . 147
使用上の注意 . . . . . 147
Set_Log_Data (cmsld) . . . . . 148
関数コール . . . . . 148
Java CPI-C の関数コール . . . . . 148
指定パラメーター . . . . . 148
戻りパラメーター . . . . . 149
発行時の状態 . . . . . 149
状態の変化 . . . . . 149
使用上の注意 . . . . . 149
Set_Mode_Name (cmsmn). . . . . 150
関数コール . . . . . 150
Java CPI-C の関数コール . . . . . 150
指定パラメーター . . . . . 150
戻りパラメーター . . . . . 151
発行時の状態 . . . . . 151
状態の変化 . . . . . 151
使用上の注意 . . . . . 151
Set_Partner_LU_Name (cmspln) . . . . . 152
関数コール . . . . . 152
Java CPI-C の関数コール . . . . . 152
指定パラメーター . . . . . 152
戻りパラメーター . . . . . 153
発行時の状態 . . . . . 153
状態の変化 . . . . . 153
使用上の注意 . . . . . 153
Set_Prepare_To_Receive_Type (cmsptr) . . . . . 153
関数コール . . . . . 154
Java CPI-C の関数コール . . . . . 154
指定パラメーター . . . . . 154
戻りパラメーター . . . . . 155
発行時の状態 . . . . . 155
状態の変化 . . . . . 155
使用上の注意 . . . . . 155
Set_Processing_Mode (cmspm) . . . . . 155
関数コール . . . . . 156
指定パラメーター . . . . . 156
戻りパラメーター . . . . . 157
発行時の状態 . . . . . 157
状態の変化 . . . . . 157
使用上の注意 . . . . . 157
Set_Receive_Type (cmsrt). . . . . 157
関数コール . . . . . 158
Java CPI-C の関数コール . . . . . 158
指定パラメーター . . . . . 158
戻りパラメーター . . . . . 158
発行時の状態 . . . . . 159
状態の変化 . . . . . 159
使用上の注意 . . . . . 159
Set_Return_Control (cmsrc) . . . . . 159
関数コール . . . . . 159

Java CPI-C の関数コール . . . . . 159
指定パラメーター . . . . . 160
戻りパラメーター . . . . . 160
発行時の状態 . . . . . 160
状態の変化 . . . . . 160
使用上の注意 . . . . . 160
Set_Send_Type (cmsst) . . . . . 161
関数コール . . . . . 161
Java CPI-C の関数コール . . . . . 161
指定パラメーター . . . . . 161
戻りパラメーター . . . . . 162
発行時の状態 . . . . . 162
状態の変化 . . . . . 162
使用上の注意 . . . . . 162
Set_Sync_Level (cmssl) . . . . . 163
関数コール . . . . . 163
Java CPI-C の関数コール . . . . . 163
指定パラメーター . . . . . 163
戻りパラメーター . . . . . 164
発行時の状態 . . . . . 164
状態の変化 . . . . . 164
使用上の注意 . . . . . 164
Set_TP_Name (cmstpn) . . . . . 164
関数コール . . . . . 165
Java CPI-C の関数コール . . . . . 165
指定パラメーター . . . . . 165
戻りパラメーター . . . . . 165
発行時の状態 . . . . . 166
状態の変化 . . . . . 166
使用上の注意 . . . . . 166
Specify_Local_TP_Name (cmsltp) . . . . . 166
関数コール . . . . . 167
Java CPI-C の関数コール . . . . . 167
指定パラメーター . . . . . 167
戻りパラメーター . . . . . 167
発行時の状態 . . . . . 168
状態の変化 . . . . . 168
使用上の注意 . . . . . 168
Specify_Windows_Handle (xchwnd) . . . . . 168
関数コール . . . . . 169
指定パラメーター . . . . . 169
戻りパラメーター . . . . . 169
発行時の状態 . . . . . 169
状態の変化 . . . . . 169
Test_Request_to_Send_Received (cmtrts) . . . . . 170
関数コール . . . . . 170
Java CPI-C の関数コール . . . . . 170
指定パラメーター . . . . . 170
戻りパラメーター . . . . . 170
発行時の状態 . . . . . 171
状態の変化 . . . . . 171
Wait_For_Conversation (cmwait) . . . . . 171
関数コール . . . . . 172
指定パラメーター . . . . . 172
戻りパラメーター . . . . . 172
発行時の状態 . . . . . 173

状態の変化 . . . 173
使用上の注意 . . . 173
WinCPICCleanup . . . 174
関数コール . . . 174
指定パラメーター . . . 174
戻り値 . . . 174
WinCPICIsBlocking . . . 174
関数コール . . . 174
指定パラメーター . . . 175
戻り値 . . . 175
WinCPICSetBlockingHook . . . 175
関数コール . . . 175
指定パラメーター . . . 175
戻り値 . . . 175
使用法 . . . 176
WinCPICStartup . . . 176
関数コール . . . 176
指定パラメーター . . . 176
戻り値 . . . 177
WinCPICUnhookBlockingHook . . . 178
関数コール . . . 178
指定パラメーター . . . 178
戻り値 . . . 178
WinCPICSetEvent . . . 178
関数コール . . . 178
指定パラメーター . . . 178
戻りパラメーター . . . 179
使用上の注意 . . . 179
WinCPICExtractEvent . . . 179
関数コール . . . 179
指定パラメーター . . . 179
戻りパラメーター . . . 180
使用上の注意 . . . 180

**第 4 章 サンプル CPI-C トランザクション・プログラム . . . 181**
処理の概要 . . . 181
疑似コード . . . 181
CSAMPLE1 (呼び出し側プログラム) . . . 182
CSAMPLE2 (呼び出し対象 TP) . . . 182
TP のテスト . . . 182

**第 5 章 サンプルの Java CPI-C トランザクション・プログラム . . . 185**
概要 . . . 185
サンプル・プログラムのコンパイル、リンク、および実行 . . . 185

**付録 A. 戻りコードの値 . . . 189**

**付録 B. 共通な戻りコード . . . 191**
パートナー・プログラムに共通する戻りコード . . . 191
CPI-C LU 6.2 以外のパートナー・プログラム . . . 195

**付録 C. 会話状態の変化 . . . 197**

**付録 D. アクセシビリティ . . . 203**
支援機能の使用 . . . 203
ユーザー・インターフェースのキーボード・ナビゲーション . . . 203

z/OS の情報 . . . . . 203

**付録 E. 特記事項 . . . . . 205**
商標 . . . . . 207

**参考文献. . . . . 209**
CS/AIX バージョン 6.3 の資料 . . . . . 209
IBM Communications Server for AIX バージョン 4.2 関連資料 . . . . . 211
IBM Redbooks . . . . . 211
ブロック・マルチプレクサーおよび S/390 ESCON チャネル PCI アダプター 関連資料 . . . . . 212
AnyNet/2 ソケットおよび SNA 関連資料 . . . . . 212
AIX オペレーティング・システム関連資料 . . . . . 212
システム・ネットワーク体系 (SNA) 関連資料 . . . . . 213
ホスト構成関連資料 . . . . . 213
z/OS Communications Server 関連資料 . . . . . 213
マルチプロトコル・トランスポート・ネットワーキング関連資料 . . . . . 214
TCP/IP 関連資料 . . . . . 214
X.25 関連資料 . . . . . 214
APPC 関連資料. . . . . 214
プログラミング関連資料. . . . . 214
その他の IBM ネットワーキング関連資料. . . . . 215

**索引 . . . . . 217**

# 表


1. 表記上の規則 . . . . . xx
2. X/Open 関数と IBM CPI-C 2.0 関数の間のマッピング . . . . . 2
3. 単純なマップ式会話 . . . . . 5
4. 確認処理 . . . . . 6
5. 会話状態の変更 . . . . . 9
6. 非ブロッキング操作. . . . . 16
7. 初期会話特性を変更する Set_* コール . . . . . 22
8. Extract_* コールとそのアクション . . . . . 28
9. サイド情報の追加、置換、検索、または削除のためのコール . . . . . 29
10. 初期会話特性の変更. . . . . 30
11. Java CPI-C の定数 . . . . . 43
12. Allocate コールによる状態の変化 . . . . . 61
13. Confirm コールによる状態の変化 . . . . . 68
14. Confirmed コールによる状態の変化 . . . . . 70
15. Deallocate コールの発行時の会話状態 . . . . . 76
16. Deallocate コールによる状態の変化. . . . . 76
17. Prepare_To_Receive コールによる状態の変化 . . . . . 103
18. 受信状態で Receive コールを発行したときの状態の変化 . . . . . 111
19. 送信状態で Receive コールを発行したときの状態の変化 . . . . . 111
20. 送信 - 保留状態で Receive コールを発行したときの状態の変化. . . . . 111
21. 任意の許可状態で Receive コールを発行したときの状態の変化 . . . . . 112
22. データ伝送エラーによる状態の変化 . . . . . 112
23. Send_Data コールによる状態の変化 . . . . . 120
24. Send_Error コールによる状態の変化 . . . . . 124
25. 会話状態の変化 . . . . . 198

# 図

1. プログラム間の通信 . . . . . 3
2. 複数会話 . . . . . 13

# 本書について

本書は、共通プログラミング・インターフェース・コミュニケーション (CPI-C) を使用してシステム・ネットワーク体系 (SNA) 環境でデータの交換を行う C 言語または Java® のアプリケーション・プログラムを開発するためのガイドです。Communications Server for AIX (以後 CS/AIX と呼びます) は、AIX® を実行するサーバーが、SNA ネットワーク上の他のノードと情報を交換できるようにするための IBM® ソフトウェア製品です。

CPI-C の CS/AIX インプリメンテーションは、IBM が OS/2® 製品において実現している CPI-C を基にしています (AIX 環境に合わせて変更を加えています)。

CPI-C の CS/AIX インプリメンテーションを使用するために作成されたプログラムは、SNA 論理装置 (Logical Unit: LU) 6.2 アーキテクチャーに従っている他の CPI-C のインプリメンテーションを使用するために作成されたプログラムと、データを交換できます。

本書は、AIX バージョン 5.2 以降の基本オペレーティング・システム上で稼働する CS/AIX バージョン 6.3 に適用されます。

## 本書の対象読者

本書は、CS/AIX がインストールされたシステム用のシステム・ネットワーク体系 (SNA) のトランザクション・プログラムを作成する、熟練した C プログラマーまたは Java プログラマーを対象にしています。ただし、SNA や CS/AIX の通信機能についての実務経験はなくても構いません。

アプリケーション・プログラマーは、CS/AIX プログラミング・インターフェースを使用して SNA ネットワーク上でデータを送受信するトランザクション・プログラムおよびアプリケーション・プログラムの設計およびコーディングを行います。したがって、アプリケーション・プログラマーは、SNA、トランザクション・プログラムまたはアプリケーション・プログラムの通信相手のリモート・プログラム、および AIX または Linux オペレーティング・システムのプログラミング環境と操作環境に関して十分理解している必要があります。

アプリケーション・プログラムの作成の詳細については、各 API またはバックレベルの API の資料を参照してください。CS/AIX の関連資料の追加情報については、参考文献を参照してください。

## 本書の使用法

この節では、本書の構成と、本書で使用する表記規則について説明します。

### 本書の構成

本書は、次のように構成されています。

- 1 ページの『第 1 章 概念』では、CPI-C の基本概念を紹介します。この章は CPI-C にあまり慣れていないプログラマーを対象としています。
- 21 ページの『第 2 章 CPI-C アプリケーションの作成』では、CPI-C プログラマーが CPI-C アプリケーションを作成するときに必要な一般情報を収めてあります。
- 53 ページの『第 3 章 CPI-C コール』では、各 CPI-C コールについて詳しく説明します。各コールの説明には、目的、パラメーター、コールを発行できる会話状態、およびコール実行後の会話状態の変化に関する説明が含まれています。異なるオペレーティング・システムの CPI-C のインプリメンテーションの相違は、相違が発生するところで説明されます。
- 181 ページの『第 4 章 サンプル CPI-C トランザクション・プログラム』では、C プログラムにおける CPI-C コールの使用法を示す CS/AIX CPI-C サンプル・プログラムについて説明します。またこのプログラムのコンパイル、リンク、および実行方法についても説明します。


- 185 ページの『第 5 章 サンプルの Java CPI-C トランザクション・プログラム』では、Java アプリケーションにおける CPI-C コールの使用法を示す CS/AIX Java CPI-C サンプル・プログラムについて説明します。また、このプログラムのコンパイル、リンク、および実行方法についても説明します。

- 189 ページの『付録 A. 戻りコードの値』 は、CPI-C インターフェースの発生しうるすべての戻りコードを番号順にリストし、意味を説明します。
- 191 ページの『付録 B. 共通な戻りコード』では、複数のコールに共通する特定の戻りコードについて説明します。
- 197 ページの『付録 C. 会話状態の変化』では、CPI-C 会話状態について説明します。各会話状態で使用できる CPI-C コール、各コールから戻ったときに起こる会話状態の変化を示します。

## 表記上の規則

本書では、表 1 に示すような表記上の規則を使用しています。

*表 1. 表記上の規則*

| 内容 | 表記例 |
|---|---|
| 資料名 | 「*Communications Server for AIX 管理ガイド*」 |
| ファイル名またはパス名 | **cmc.h** |
| コマンドまたは AIX / Linux ユーティリティー | **vi** |
| オプションまたはフラグ | **-I** |
| パラメーターまたは Motif フィールド | *data_received*、*request_to_send_received* |
| ユーザーが入力できるリテラル値または選択項目 (デフォルト値を含む) | `0;32,767` |
| 定数またはシグナル | `CM_NONE` |

*表 1. 表記上の規則 (続き)*

| 内容 | 表記例 |
|---|---|
| 戻り値 | `CM_OK;CM_PRODUCT_SPECIFIC_ERROR` |
| 指定値を表す変数 | *functionname* |
| 環境変数 | `APPCTPN` |
| プログラミング verb | RECEIVE |
| ユーザーの入力 | **cc -I** |
| 関数、コール、またはエントリー・ポイント | `WinCPICSetEvent` |
| データ構造 | `WCPICDATA` |
| 16 進値 | `0x20` |

## シンボルの意味

AIX、LINUX

このシンボルは、AIX または Linux システムだけに該当する説明のセクション開始を表します。これは AIX サーバーと、 AIX、Linux、Linux for pSeries 、または Linux for zSeries 上で稼働する IBM Remote API Client に適用されます。

WINDOWS

このシンボルは、Windows 上の IBM Remote API Client に該当する説明のセクション開始を表します。

このシンボルは、オペレーティング・システム固有の説明のセクションの終了を表します。このシンボルに続く情報は、どのオペレーティング・システムにも適用されます。

# 新しい機能

Communications Server for AIX V6.3 は、Communications Server for AIX V6.1 に置き換わるものです。

サポートが続く本製品のリリースは、次の通りです。

- Communications Server for AIX V6.1

サポートされなくなった本製品のリリースは、次の通りです。

- Communications Server for AIX バージョン 6 (V6)
- Communications Server for AIX バージョン 5 (V5)
- Communications Server for AIX バージョン 4 リリース 2 (V4R2)
- Communications Server for AIX バージョン 4 リリース 1 (V4R1)
- SNA サーバー for AIX バージョン 3 リリース 1.1 (V3R1.1)
- SNA サーバー for AIX バージョン 3 リリース 1 (V3R1)

- AIX SNA サーバー/6000 バージョン 2 リリース 2 (V2R2)
- AIX 3.2 で稼働する AIX SNA サーバー/6000 バージョン 2 リリース 1 (V2R1)
- AIX SNA サービス /6000 バージョン 1

## 詳細について

CS/AIX ライブラリーのその他の資料、および SNA ワークステーションと AIX ワークステーション関連事項についての追加情報は、『参考文献』を参照してください。

CS/AIX 資料に収録された情報は、HTML 形式でも入手できます。このライブラリーを使用して、特定の情報を検索したり、各 CS/AIX 資料のオンライン・バージョンを参照することができます。

# 第 1 章 概念

この章では、分散処理環境での CPI-C の基本概念を紹介します。この章で説明するトピックは次のとおりです。

- CPI-C とは
- 単純なマップ式会話の例
- 確認処理
- 会話状態
- 会話状態の変更方法
- サイド情報
- 基本会話
- 複数会話
- 会話セキュリティー
- 非ブロッキング操作
- CPI-C と LU 6.2

## CPI-C とは

CPI-C とは、共通プログラミング・インターフェース・コミュニケーションのことです。CPI-C は、SNA 環境のプログラム間の対等通信を可能にする、移植性の高いアプリケーション・プログラミング・インターフェース (API) です。

CPI-C を使用すると、ネットワーク全体に分散されているアプリケーション・プログラムを協同で作業させることができます。つまり、互いに通信しデータを交換することにより、リモート・データベースへの照会、リモート・ファイルのコピー、または電子メールの送受信などのような 1 つの処理作業をすべてのアプリケーション・プログラムが実行することができます。

これらのプログラムは、対等ベースで (階層関係でなく) 通信します。ローカル・エリア・ネットワークまたは広域ネットワークに分散しているプログラムは、協同で分散処理を実行します。

### CS/AIX CPI-C オプション・セット・サポート

AIX、LINUX

(Java プログラムではなく) C プログラムの場合は、CS/AIX CPI-C は IBM の CPI-C 2.0 をインプリメントします。CS/AIX CPI-C は、必須の CPI-C 2.0 準拠クラスである会話をサポートするほか、次に示すオプションの準拠クラスもサポートしています。

- LU 6.2
- 会話レベルの非ブロッキング操作

- サーバー
- データ変換ルーチン
- セキュリティー

また、CS/AIX CPI-C は、X/Open CPI-C のインプリメンテーションの一部として定義され、CPI-C 2.0 に組み込まれている追加関数もサポートします。CS/AIX は、既存の CPI-C アプリケーションとの後方互換のために、これらのエントリー・ポイントをサポートしています。CPI-C プログラマーは、可能な場合は必ず IBM CPI-C 2.0 バージョンの関数を使用する必要があります。X/Open 関数と IBM CPI-C 2.0 関数との間のマッピングを表 2 に示します。

*表 2. X/Open 関数と IBM CPI-C 2.0 関数の間のマッピング*

| X/Open 関数 | IBM CPI-C 2.0 関数 |
|---|---|
| Extract_Conversation_Security_User_ID (xcecsu) | Extract_Security_User_ID (cmesui) |
| Set_Conversation_Security_Password (xcscsp) | Set_Conversation_Security_Password (cmscsp) |
| Set_Conversation_Security_Type (xcscst) | Set_Conversation_Security_Type (cmscst) |
| Set_Conversation_Security_User_ID (xcscsu) | Set_Conversation_Security_User_ID (cmscsu) |

Java プログラムの場合、CS/AIX は Java CPI-C を IBM の CS/Windows 製品の場合と同様にインプリメントします (パッケージ COM.ibm.eNetwork.cpic)。また CS/AIX には、CPI-C 関数が 3 つ追加されています (Set_Conversation_Context、Set_Local_LU_Name、および Extract_Local_LU_Name)。これらの関数は標準の CS/AIX CPI-C インプリメンテーションの一部ですが、CS/Windows には含まれていません。

WINDOWS

Windows 上の CS/AIX CPI-C は、Windows CPI-C (WOSA SNA 仕様により定義済み) をインプリメントします。

## プログラム間の通信

SNA 環境で 2 つのプログラムが相互に通信するためには、多数のハードウェアおよびソフトウェアのエレメントが必要です。次のダイアグラムは、プログラマーに関係のあるエレメントを示しています。

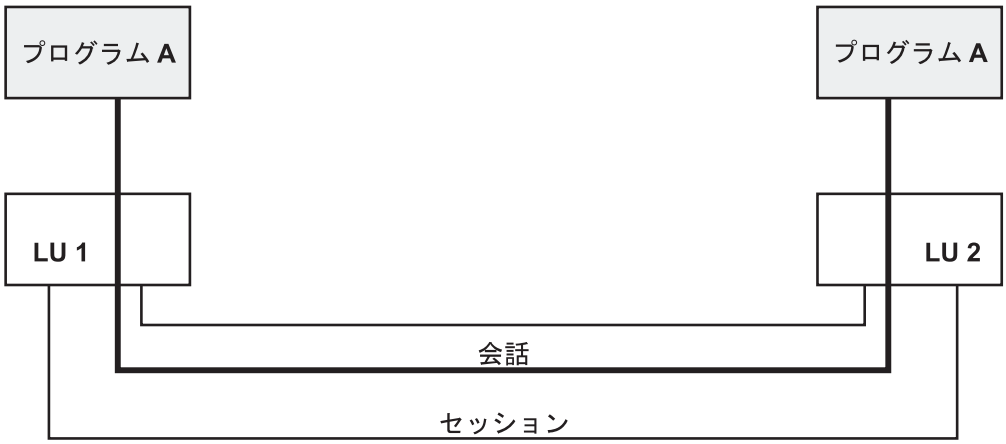


*図 1. プログラム間の通信*

## 論理装置 6.2

各プログラムは論理装置 (LU) に関連付けられます。LU は、プログラマーがネットワークに入るためのアクセス・ポイントです。 CPI-C は、LU 間の対等通信をサポートする LU タイプ 6.2 (LU 6.2) を使用します。1 つの LU に複数のプログラムを関連付けることができます。

## セッション

2 つのプログラムが通信するには、LU-LU セッション、つまり 2 つの LU 間の論理接続により、それらのプログラムの LU を接続する必要があります。このセッションは、特定のモード、つまり LU がセッションを使用する方法を決定する一連のネットワーク特性を使用して確立されます。

LU タイプ 6.2 (LU 6.2) では、複数セッション (パートナー LU が異なる 2 つ以上の同時セッション) と並列セッション (パートナー LU が同一の 2 つ以上の同時セッション) を設定できます。構成時に、システム管理者またはユーザーは、特定の LU がサポートするセッション数、およびその LU が並列セッションをサポートするかどうかを決定します。

## 会話

2 つのプログラム間の通信は、LU-LU セッション内の会話として発生します。1 つのプログラムで同時に複数の会話を取り扱うことができます。

## コンテンション

両方の LU が同じセッションで同時に会話を割り振ろうとした場合は、必ず一方が勝ち (コンテンション勝者)、もう一方が負ける (コンテンション敗者) ことになります。2 つの LU が使用するモードにより、それぞれの LU ごとに、コンテンション勝者とコンテンション敗者になるセッション数が指定されます。コンテンション勝者の LU とコンテンション敗者の LU は、セッションが確立されたときに決まります。

セッション内では、コンテンション敗者の LU は、会話を割り振る前に、コンテンション勝者の LU に許可を求める必要があります。コンテンション勝者は、許可する場合としない場合があります。コンテンション勝者の LU の方からは、必要なときに会話を割り振ることができます。

## 特性

会話には、会話全体の操作および個々のコールの動作を制御する一連の内部値があります。これらの値を特性といいます。

## CPI-C コール

プログラムは、CPI-C コールを介して指定 CPI-C にアクセスします。各コールは、会話の開始または終了、データの送信または受信、後続の CPI-C コールがどのように動作するかを決定するオプションの設定、現在使用中のオプションについての情報の入手などのような、特定のアクションを実行します。コールのたびに、プログラムは CPI-C にパラメーターを提供します。CPI-C は要求された機能を実行し、新しいパラメーターをプログラムに戻します。

コールを発行するプログラムをローカル・プログラムといいます。相手側のプログラムをパートナー・プログラムといいます。同様に、ローカル・プログラムにサービスを提供する LU をローカル LU といい、パートナー・プログラムにサービスを提供する LU をパートナー LU といいます。

ネットワークの反対側のノードにあるプログラムおよび LU は、リモート・プログラムおよびリモート LU とも呼ばれます。

## 会話プロセス

会話は、次の両方の状態が発生したときに開始されます。

- 片方のプログラム (呼び出し側プログラム) が CS/AIX に他方のプログラム (呼び出し対象プログラム) を開始するよう指示し、2 つのプログラム間に会話を割り振る。
- 呼び出し対象プログラムが、呼び出し側プログラムとの通信の準備ができていることを CS/AIX に通知する。

会話中に、2 つのプログラムは状況情報およびアプリケーション・データを交換します。通常は、どちらかのプログラムが CS/AIX に会話の割り振り解除を指示したときに、会話が終了します。

## 会話タイプ

会話は、マップ式会話または基本会話のどちらかです。

一般に、マップ式会話はアプリケーション・プログラムが使用します。アプリケーション・プログラムは、エンド・ユーザー用のタスクを実行するプログラムです。マップ式会話は、基本会話ほど複雑ではありません。マップ式会話では、プログラムは、データ・レコードを一度に 1 つずつ送受信します。

基本会話は、一般にサービス・プログラムが使用します。サービス・プログラムは、他のローカル・プログラムにサービスを提供するプログラムです。熟練した

LU 6.2 プログラマーは、基本会話を使用して、データの伝送と処理を大幅に制御できます。詳しくは、 12 ページの『基本会話』を参照してください。

## 単純なマップ式会話

次の表に、単純なマップ式会話の例を示します。この表には、会話を開始し、データを交換し、会話を終了するために使用される CPI-C コールが示されています。矢印はデータのフローを示します。コール・パラメーターの一部と戻りコードの一部についても括弧で囲んで示しています。

*表 3. 単純なマップ式会話*

| 呼び出し側プログラム | | 呼び出し対象プログラム |
|---|---|---|
| Initialize_Conversation<br>Allocate<br>Send_Data<br>Deallocate | → | Accept_Conversation<br>Receive<br>(*data_received*=CM_COMPLETE_DATA_RECEIVED)<br>(*return_code*=CM_DEALLOCATED_NORMAL) |

### 会話の開始

会話を開始するには、呼び出し側プログラムが次のコールを発行します。

- Initialize_Conversation。このコールは、CPI-C に、会話の特性を設定するよう要求します。

  Initialize_Conversation コールは、CS/AIX 構成内の CPI-C サイド情報エントリーに関連付けられているシンボリック宛先名を指定します。このエントリーには、パートナー・プログラム、パートナー LU、モード、およびセキュリティー情報が指定されています。
- Allocate。このコールは、呼び出し側プログラムと呼び出し対象プログラムの間に会話を確立するよう、CS/AIX に要求します。

呼び出し対象プログラムは、Accept_Conversation コールを発行します。このコールは、呼び出し対象プログラムが呼び出し側プログラムとの会話を開始する準備ができていることを CS/AIX に通知します。

### データの送信

Send_Data コールは、1 つのデータ・レコード (伝送するアプリケーション・データが入っています) を、ローカル LU の送信バッファーに入れます。この送信バッファーには、すでに割り振り要求が入っています。次のイベントのいずれかが発生するまでは、パートナー・プログラムへのデータの伝送は行われません。

- 送信バッファーが満ぱいになる
- バッファーのフラッシュ (およびパートナー・プログラムへのデータ送信) を CS/AIX に強制するコールをプログラムが発行する

Deallocate コールは、割り振り要求とデータをパートナー・プログラムに送信している送信バッファーをフラッシュします。

## データの受信

Receive コールは、データ・レコードと状況情報をパートナー・プログラムから受信します。現在受信可能なデータまたは状況情報がない場合、デフォルトでは、ローカル・プログラムはデータの到着を待ちます。

Receive コールの *data_received* パラメーターは、データを受信したかどうか、および受信したデータが完全かどうかをプログラムに通知します。

## 会話の終了

会話を終了するには、プログラムの一方が Deallocate コールを発行します。それによって、CS/AIX は、2 つのプログラムの間の会話の割り振りを解除します。

# 確認処理

プログラムは、パートナー・プログラムにデータを送信するときに、データの受信に成功したかどうかの確認をパートナー・プログラムに求めることもできます。受信側プログラムは、データの受信を確認するか、またはエラーが起きたことを通知する必要があります。2 つのプログラムは、確認の要求と応答を交換するたびに同期化されます。この様子を表 4 に示します。

*表 4. 確認処理*

| 呼び出し側プログラム | | 呼び出し対象プログラム |
|---|---|---|
| Initialize_Conversation<br>Set_Sync_Level<br>(*sync_level*=CM_CONFIRM)<br>Allocate<br>Send_Data<br>Confirm | → | |
| | | Accept_Conversation<br>Receive<br>(*data_received*=CM_COMPLETE_DATA_RECEIVED)<br>(*status_received*=CM_CONFIRM_RECEIVED)<br>Confirmed |
| | ← | |
| (*return_code*=CM_OK)<br>Send_Data<br>Deallocate | → | |
| | | Receive<br>(*status_received*=CM_CONFIRM_DEALLOC_RECEIVED)<br>Confirmed |
| | ← | |
| (*return_code*=CM_OK) | | |

### 同期レベルの設定

同期レベルは、会話の特性の 1 つです。設定できる同期レベルは次の 2 つです。

- CM_NONE。これはデフォルトです。この場合は、確認処理は行われません。
- CM_CONFIRM。この場合は、プログラムはデータ受信の確認を要求し、その受信の確認要求に応答できます。

デフォルトの同期レベルは CM_NONE です。これは、Set_Sync_Level コールを使用して変更できます。

### 確認要求の送信

Confirm コールを発行すると、次の処理が実行されます。

- ローカル LU の送信バッファーがフラッシュされます (これにより、バッファー内に入っていたデータはすべてパートナー・プログラムに送信されます)。
- 確認要求が送信されます。パートナー・プログラムは、Receive コールの *status_received* パラメーターを介してこの確認要求を受信します。

Confirm コールを発行した後、呼び出し側プログラムは、呼び出し対象プログラムからの確認が届くのを待ちます。

### 確認要求の受信

Receive コールの *status_received* パラメーターは、ローカル・プログラムが実行する必要がある後続のアクションを指示します。

前述の例では、最初の Received コールに、CM_CONFIRM_RECEIVED の *status_received* があり、パートナー・プログラムが処理を続行するためには確認が必要なことを示しています。

### 確認要求への応答

呼び出し対象プログラムは、Confirmed コールを発行して、データの受信を確認します。これによって、呼び出し側プログラムは解放され、処理を再開することができます。

### 会話の割り振り解除

会話の同期レベルが CM_CONFIRM に設定されているので、Deallocate コールは、バッファーからフラッシュされたデータと共に確認要求を送信します。

2 番目の Receive コールでは、*status_received* が CM_CONFIRM_DEALLOC_RECEIVED であり、パートナー・プログラムが Confirmed コールにより生成された確認を戻してからでないと、この会話の割り振りを解除できないことを示しています。

## 会話状態

会話の状態により、プログラムがどの CPI-C コールを発行できるかが制御されます。たとえば、会話が送信状態または送信 - 保留状態になっていないときは、プログラムは Send_Data コールを発行できません。起こり得る会話状態の要約を次に示します。

**リセット**
会話が開始していません。または会話が終了しました。

**初期化** 会話は正常に初期化されました。

**送信** プログラムは、パートナー・プログラムにデータを送信し、確認を要求することができます。会話が送信状態にあるときは、プログラムはデータの受信を開始することもできます。その場合、状態が受信に変わります。

**送信 - 保留**
プログラムは、Receive コールを発行し、データを受信すると同時に、プログラムがデータの送信を開始できることを示す送信インディケーター (*status_received* = CM_SEND_RECEIVED) を受信しました。この状態は、プログラムがエラーを報告する追加情報 (受信データまたはプログラム自体の処理でエラーを検出したかどうかを示す) を提供できることを除けば、送信状態と同じです。

**受信** プログラムは、アプリケーション・データと状況情報をパートナー・プログラムから受信できます。会話が受信状態にあるときは、プログラムはエラー情報を送信したり、データ送信のための許可を要求したりすることもできます。

**確認** プログラムは、データ受信の確認要求を受信しました。パートナー・プログラムに肯定応答またはエラー情報を送信する必要があります。

**確認 - 割り振り解除**
プログラムは、確認要求を受信しました。肯定応答またはエラー情報を送信する必要があります。プログラムが肯定応答を送ると、パートナー・プログラムは会話の割り振りを解除します。

**確認 - 送信**
プログラムは確認要求を受信しました。肯定応答またはエラー情報を送信する必要があります。応答すると、プログラムはデータの送信を開始できます。

AIX、LINUX

**初期化 - 着呼**
プログラムは正常に Initialize_For_Incoming を発行し、会話 ID を取得しました。現在、着呼会話を受け入れるための Accept_Incoming を発行できる状態になっています。

WINDOWS

**保留 - 通知**
プログラムは、正常に非ブロッキング・モードで受信コールを送出しました。コールが未解決であっても、この会話に限定された範囲の CPI-C コールを送出したり、他の会話に CPI-C コールを送出したり、あるいは他の処理を継続したりすることができます。

各 CPI-C コールの説明には、そのコールを発行できる会話状態についての情報も含まれています。各会話状態で発行できる verb の一覧表については、 197 ページの『付録 C. 会話状態の変化』を参照してください。

## プログラム側から見た会話

特定の状態になるのは、プログラムでなく会話の方です。プログラムは、それぞれに状態の異なる複数の会話を実行することができます。会話が送信状態にあるという場合、これはローカル・プログラムの側から見た表現です。パートナー・プログラムにとっては、会話は別の状態 (受信状態など) です。

## 状態の変化

次のいずれかを行った結果として、会話状態が変化することがあります。

- ローカル・プログラムが発行したコール
- パートナー・プログラムが発行したコール
- エラー状態

## 状態チェック

状態チェックは、プログラムが CPI-C コールを発行し、会話が適切な状態になっていない場合に実行されます。たとえば、会話が受信状態にあるときにプログラムが Send_Data コールを発行すると、状態チェックが起こります。状態チェックが起こると、CPI-C はコールを実行しないで、*return_code* パラメーターを介して状態チェック情報を戻します。

# 会話状態の変更

表 5 では、会話状態が左端と右端に表示されています。 この表は、CPI-C コールが会話状態を送信から受信に、および受信から送信に変更する方法を示しています。

*表 5. 会話状態の変更*

| 状態 | 呼び出し側プログラム | 呼び出し対象プログラム | 状態 |
|---|---|---|---|
| リセット | | | |
| | Initialize_Conversation | | |
| 初期化 | | | |
| | Set_Sync_Level<br>(*sync_level*=CM_CONFIRM) | | |
| | Allocate | | |
| 送信 | | | |
| | Send_Data | | |
| | Prepare_To_Receive | | リセット |
| | | → | |
| | | Accept_Conversation | |
| | | | 受信 |
| | | Receive<br>(*status_received*=CM_CONFIRM_SEND_RECEIVED) | |
| | | | 確認 - 送信 |
| | | Confirmed | |
| | | ← | |
| | | | 送信 |
| | (*return_code*=CM_OK) | | |

*表 5. 会話状態の変更 (続き)*

| 状態 | 呼び出し側プログラム | 呼び出し対象プログラム | 状態 |
|---|---|---|---|
| 受信 | | | |
| | | Send_Data<br>Confirm<br>← | |
| | Receive<br>(*status_received*=<br><br>CM_CONFIRM_RECEIVED) | | |
| 確認 | | | |
| | Request_To_Send<br>Confirmed | | |
| | | → | |
| 受信 | | | |
| | | (*return_code*=CM_OK)<br>(*request_to_send_received*=<br><br>CM_REQ_TO_SEND_RECEIVED)<br>Prepare_To_Receive<br>← | |
| | Receive<br>(*status_received*=<br><br>CM_CONFIRM_SEND_RECEIVED) | | |
| 確認 - 送信 | | | |
| | Confirmed | | |
| | | → | |
| 送信 | | | |
| | | (*return_code*=CM_OK) | |
| | | | 受信 |
| | Send_Data<br>Deallocate | | |
| | | →<br>Receive<br>(*status_received*=<br><br>CM_CONFIRM_DEALLOC_RECEIVED) | |
| | | | 確認 - 割り振り解除 |
| | | Confirmed<br>← | |
| | | | リセット |
| | (*return_code*=CM_OK) | | |
| リセット | | | |

## 初期状態

会話が割り振られる前は、両方のプログラムはリセット状態です。

会話が割り振られたあとは、呼び出し側プログラムの初期状態は送信で、呼び出し対象プログラムの初期状態は受信です。

## 受信状態への変更

Prepare_To_Receive コールにより、プログラムは、会話を送信状態から受信状態に変更します。このコールにより、次の処理が実行されます。

- ローカル LU の送信バッファーがフラッシュされます。
- 同期レベルが CM_CONFIRM に設定されている場合は、Prepare_To_Receive コールは、Receive コールの *status_received* パラメーターを介して、パートナー・プログラムに CM_CONFIRM_SEND インディケーターを送信します。このインディケーターは、パートナー・プログラムに、パートナー・プログラムがデータを送信するには、その前に Confirmed 応答を送信する必要があることを知らせます。

### 送信状態への変更

パートナー・プログラムが (Prepare_To_Receive コールを発行して) データを受信し始めると、プログラムの会話状態は、受信から送信に変わります。

ローカル・プログラム (このプログラムにとっては会話は受信状態) は、Request_To_Send コールを発行することによって、データを送信したいことをパートナー・プログラムに知らせることができます。この要求は、*request_to_send_received* パラメーターを介してパートナー・プログラムに伝達されます。(前述の例では、このパラメーターは Confirm コールの上に示されています。このパラメーターは、Send_Data およびその他のコールに対しても戻されます。)

Request_To_Send コールを発行しても、会話の状態は変わりません。これは、パートナー・プログラムがこのコールを無視できるからです。パートナー・プログラムが Prepare_To_Receive コールを発行すると、パートナー・プログラムの場合は会話状態が受信に変化します。ローカル・プログラムは、後続の RECEIVE verb の SEND 指示を受信すると、データを送信できる状態になります。

## サイド情報

2 つのプログラムが通信するために必要な情報は、CS/AIX 構成ファイル内の CPI-C サイド情報エントリーに保管されています。各サイド情報エントリーは、シンボリック宛先名、つまり Initialize_Conversation コールで指定される *sym_dest_name* パラメーターによって識別されます。パラメーター *sym_dest_name* は、8 バイトの ASCII 文字ストリングで、表示可能な文字はすべて使用することができます。このパラメーターには次のフィールドがあります。

- パートナー LU 名
- パートナー・プログラムのタイプと名前
- モード名
- 会話セキュリティー・タイプ (13 ページの『複数会話』を参照)
- パートナー・プログラムにアクセスするために必要なセキュリティー・ユーザー ID とパスワード

また CPI-C には、次のように、構成済みのサイド情報エントリーをアプリケーションが変更するための 2 つのメカニズムも備わっています。これらのメカニズムを両方とも適用できるのは、アプリケーション自体がこの情報を使用する場合に限られ、構成ファイルに保管されているオリジナル・バージョンが変更されることはありません。

- アプリケーションは、Set_CPIC_Side_Information コール、Extract_CPIC_Side_Information コール、および Delete_CPIC_Side_Information コー

ルを使用して、アプリケーション専用の、すべてのサイド情報エントリーのローカル・コピーを管理することができる。 (これらの関数は Java CPI-C では使用できません。)

- アプリケーションは、CPI-C Set_* 関数 (Set_Partner_LU_Name など) を使用して、会話の割り振り前にサイド情報の中のパラメーターを変更することができる。

詳しくは、 34 ページの『サイド情報』を参照してください。

# 基本会話

基本会話は、一般にサービス・プログラムが使用します。サービス・プログラムは、他のローカル・プログラムにサービスを提供するプログラムです。基本会話はマップ式会話より複雑ですが、熟練した LU 6.2 プログラマーは、基本会話を使用して、データの伝送と処理を大幅に制御できます。ここでは、基本会話の特性について簡単に説明します。

## 論理レコード

基本会話では、データは論理レコードの形式で送信されます。論理レコードとは、ここで説明する汎用データ・ストリーム (GDS) 構文を持つレコードです。GDS 構文についての詳細は、「*IBM Systems Network Architecture: Formats*」を参照してください。

送信側 TP はデータを複数の論理レコードに形式設定し、受信側 TP は論理レコードを使用可能データにデコードする必要があります。

論理レコードが単一レコードの場合、次のフィールドで構成されています。

- 2 バイトのレコード長 (LL) フィールド
- 2 バイトの GDS 識別子 (ID) フィールド (たとえば、値が `0x12FF` であるデータは、アプリケーション・データとして識別されます)
- 長さが 0 ～ 32,763 バイトの範囲のデータ・フィールド

先頭の 4 バイトを LLID と呼びます。

論理レコードに複数の部分がある場合、最初の部分は単一レコードと同じ形式になり、後続の部分はすべて、次のフィールドで構成されています。

- 2 バイトのレコード長 (LL) フィールド
- 長さが 0 ～ 32,765 バイトの範囲のデータ・フィールド

LL フィールドに記録される長さには、LL フィールド用の 2 バイト (および、ID フィールドがある場合は ID フィールド用の 2 バイト) が含まれます。たとえば、データの入っていない単一部分の GDS では、LL フィールドの値は `0x0004` になります。LL フィールドは、バイト・スワップ形式でなく、高低形式であることが必要です。たとえば、長さ 230 バイトは、`0xE600` でなく `0x00E6` として表します。

LL のバイト 0 のビット 0 (最上位ビット) は、長さの連結 (セグメント化) を示すために使用されます。次の例は、3 つの GDS セグメントに分割された 10 バイトのデータ (各データ・バイトの値は `DD` です) を示しています。第 1 と第 2 のセグ

メントには、それぞれ 4 バイトのデータが含まれ、最後のセグメントには 2 バイトのデータが含まれています。

8008 12FF DDDD DDDD
8006 DDDD DDDD
0004 DDDD

LL フィールドでは、次の値は有効ではありません。

- 0x0000
- 0x0001
- 0x8000
- 0x8001

### エラー・ログ・データ

基本会話にエラーまたはアベンドが起きた場合、プログラムは、汎用データ・ストリーム (GDS) エラー・ログ変数の形式で、パートナー LU にエラー・メッセージを送信できます。

## 複数会話

1 つのプログラムで同時に複数の会話を取り扱うことができます。それぞれの会話ごとに、LU-LU セッションが 1 つずつ必要です。アプリケーションで従属 LU を使用する場合は、複数会話はサポートされません (詳細については、 38 ページの『ローカル LU の指定』を参照してください)。

複数会話の一般的な使用法は、呼び出し対象プログラムが別のプログラムを呼び出し、そのプログラムがさらに別のプログラムを呼び出し、以降同様に呼び出す形になります。下のダイアグラムでは、プログラム A がプログラム B を呼び出し、プログラム B がプログラム C を呼び出します。

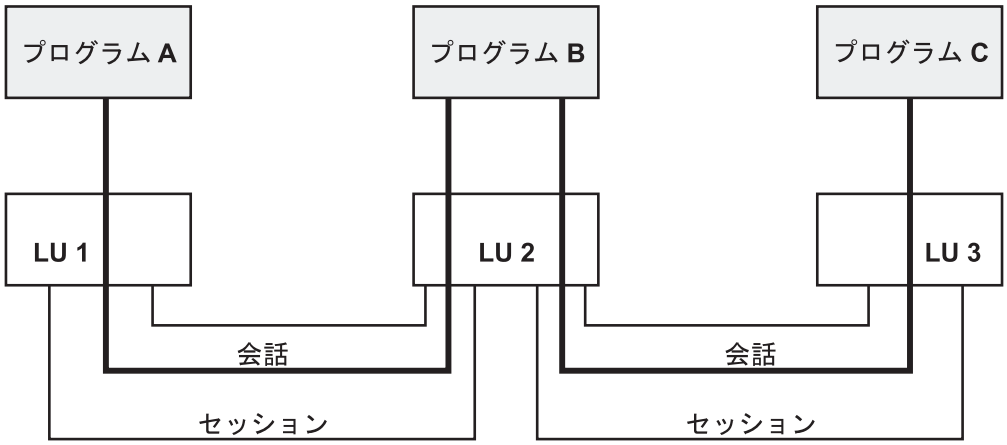


*図 2. 複数会話*

CPI-C 会話セキュリティーが複数の会話でどのように動作するのかの詳細については、 14 ページの『会話セキュリティーの概要』を参照してください。

# 会話セキュリティーの概要

会話セキュリティーを使用すると、呼び出し対象の TP との会話を CPI-C が割り振る前に、ユーザー ID とパスワードを提供するよう呼び出し側プログラムに要求することができます。

呼び出し対象の TP を構成する際に、システム管理者は会話セキュリティーを使用するかどうかを指定します。会話セキュリティーを使用する場合、呼び出し側の TP は呼び出し対象プログラムとの会話を割り振るのに、ユーザー ID とパスワードを提供する必要があります。このユーザー ID とパスワードは、サイド情報から取り出されるか、または呼び出し側プログラムが明示的に指定するもので、呼び出し対象プログラムの構成情報内で指定されているユーザー ID とパスワードに一致していなければなりません。

また CS/AIX は LU-LU セッション・セキュリティーもサポートしています。LU-LU セッション・セキュリティーは、ローカル LU とリモート LU との間のセッションを開始する際に、セキュリティー検査を行います。 LU-LU セッション・セキュリティーは構成時に指定され、CPI-C プログラムでのアクションは要求しません。詳しくは、「*Communications Server for AIX 管理ガイド*」を参照してください。

## 複数会話の会話セキュリティー

13 ページの『複数会話』で示した例では、プログラム A がプログラム B を呼び出し、B が A との会話の結果として C を呼び出した場合、C の構成情報は、「検査済み」セキュリティー指示を受け入れることを示しています。この例では、A が提供するユーザー ID とパスワードを、B の構成情報に照らして検査する必要があります。しかし、B が C を呼び出すときは、B は *security_type* 会話特性を「same」に設定します。そうすると、CPI-C は、A から提供されたユーザー ID と、セキュリティーがすでに検査済みであることを示す指示を C に送信します。詳しくは、130 ページの『Set_Conversation_Security_Type (cmscst)』を参照してください。

AIX、LINUX

この方法で着呼会話と発信会話の複数のペアを取り扱う場合は、発信会話にユーザー ID を指定するのはどの着呼会話かを、プログラムで指示する必要があります。そのために、CPI-C は、各会話を特定の「コンテキスト ID」に関連付けます。コンテキスト ID は、次のようにして割り当てられ、使用されます。

- プログラムが Accept_Conversation または Accept_Incoming を正常に発行するたびに、CPI-C は、新しいコンテキスト ID を会話に割り当てます。プログラムは、該当の会話 ID と共に Extract_Conversation_Context を発行することにより、このコンテキスト ID の値を決定します。
- プログラムの「現行コンテキスト」は、通常、最新の Accept_Conversation または Accept_Incoming に関連付けられているコンテキスト ID です。プログラムは、Set_Conversation_Context を使用して、(次に説明する制約に従って) 現行コンテキストを着呼会話の別のコンテキストのコンテキスト ID に設定できます。

- すべての Allocate コールはプログラムの現行コンテキスト内で発行されます。これは、会話セキュリティーのタイプが「same」なら、現行コンテキスト ID に関連付けられた着呼会話からのユーザー ID がパートナー・プログラムに送信されるということです。

前述の例では、プログラム B は、Allocate コールをプログラム C に発行する前に、現行コンテキストがプログラム A からの着呼会話に関連付けられたコンテキストであることを確認する必要があります。このことを確認しておくと、A のユーザー ID が割り振り要求と共に確実にプログラム C に送信されます。A からの会話を受け入れたあとで B が別の Accept_Conversation コール、Accept_Incoming コール、または Set_Conversation_Context コールを発行しない限り、通常は現行コンテキストは正しいものになります。

プログラムが Set_Conversation_Context を使用して現行コンテキストを変更すると、直前のコンテキストに関連するアクティブな会話が 1 つでも残っている場合を除いて、CS/AIX は直前のコンテキストからの情報を保存しません。これは、B が、A との会話を終了したあとで別のプログラムと通信するために現行コンテキストを変更すると、最初のコンテキスト ID に戻って C との会話を割り振ることはできないということです。C に会話を割り振る前に A との会話を終了する必要がある場合は、B は、現行コンテキストを別の値に変更する前に、C に会話を割り振る必要があります。

## 検査済み会話セキュリティー

AIX、LINUX

場合により、プログラム自身が別のプログラムから呼び出されたことがなくても、別の手段により (たとえば、ユーザーがログオン・シーケンスでユーザー ID とパスワードを入力することにより) 適切なセキュリティー情報を取得して検査したことがある場合には、プログラムは、「検査済み」セキュリティーを示す必要があります。CS/AIX はこれを次のようにサポートしています。

- 「検査済み」を指定するプログラムが別のプログラムから呼び出された場合は、14 ページの『複数会話の会話セキュリティー』で示すように、CPI-C は現在の会話コンテキストからユーザー ID を送信します。
- それ以外の場合は、CPI-C は、プログラムの実行時の AIX または Linux のユーザー名を受け取り、必要であればそれを 10 文字に切り捨て、会話セキュリティーのユーザー ID として使用します。この名前は必ず、有効な AE ストリング文字で構成され、呼び出されるプログラムで有効なユーザー名になるようにしてください。
- アプリケーションが別の方法を使用してセキュリティー情報を取得する場合 (たとえば、アプリケーションがユーザーに対して、AIX または Linux のシステム・セキュリティーに依存しないで、ユーザー ID とパスワードを明示的に指定するよう要求する場合)、アプリケーションは CPI-C 関数の

Set_Conversation_Security_User_ID または Set_CPIC_Side_Information のいずれかを使用して、この *user_id* を CPI-C に指定してから会話を割り振ることができます。

## 非ブロッキング操作

AIX、LINUX

この節は、Java CPI-C には適用されません。Java CPI-C 関数は常にブロッキング・モードで動作します。すなわちこのモードでは、Java CPI-C 関数は、要求された処理が完了するまでアプリケーションに制御を戻しません。

デフォルトでは、CPI-C の関数はブロッキング・モードで動作します。このモードでは、要求された処理が完了するまでは関数からアプリケーションに制御が戻りません。たとえば、Confirm 関数は、CPI-C がパートナー・アプリケーションに確認要求を送信し、OK またはエラー応答をパートナー・プログラムから受信するまでは、戻りません。

CPI-C の関数は、非ブロッキング・モードでも動作します。このモードでは、要求された処理がまだ完了していなくても、関数は即時にアプリケーションに制御権を戻します。したがって、アプリケーションは、この会話とは関係のない他の処理を続けることができ、verb の処理結果を後続のステージで取得できます。

AIX、LINUX

アプリケーションは、関数 Check_For_Completion を使用して、直前の非ブロッキング関数がすでに完了しているかどうか確認する、または Wait_For_Conversation を使用してその完了を待つことができます。表 6 は、非ブロッキング・モードの使用例を示しています。

*表 6. 非ブロッキング操作*

| 呼び出し側プログラム | | 呼び出し対象プログラム |
|---|---|---|
| Initialize_Conversation<br>Allocate<br>Send_Data<br>Set_Processing Mode (CM_NON_BLOCKING)<br>Confirm | → | |
| (*return_code*=CM_OPERATION_INCOMPLETE)<br>[アプリケーションは、この会話に関係のない他の処理を実行できる。] | | Accept_conversation<br><br>Receive |

*表 6. 非ブロッキング操作 (続き)*

| 呼び出し側プログラム | | 呼び出し対象プログラム |
|---|---|---|
| | | (*data_received*=CM_COMPLETE_DATA_RECEIVED)<br>(*status_received*=CM_CONFIRM_RECEIVED) |
| Wait_For_Conversation<br>[直前の Confirm の処理が完了するまで、アプリケーションは中断される。] | | |
| | | Confirmed |
| | ← | |
| (Wait_For_Conversation が戻る。<br>*return_code*=CM_OK、<br>conversation_return_code=CM_OK)<br>Send_Data<br>Deallocate | | |
| | → | |
| (*return_code*=CM_OPERATION_INCOMPLETE)<br>[アプリケーションは、この会話に関係のない他の処理を実行する。] | | |
| | | Receive<br>(*status_received*=<br>CM_CONFIRM_DEALLOC_RECEIVED)<br>Confirmed |
| | ← | |
| Check_for_Completion<br>(*return_code*=CM_OK)<br>Wait_For_Conversation<br>(*return_code*=CM_OK,<br>conversation_return_code=CM_OK)<br>[会話が割り振り解除される。] | | |

次のステップでは、上の例に示されている処理を説明します。

1. 呼び出し側プログラムは、会話を割り振り、データを送信したあとで、St_Processing_Mode を発行して処理モードを CM_NON_BLOCKING に設定します。これは、この会話の後続の関数が非ブロッキング・モードで動作できることを示します。
2. 呼び出し側プログラムは、次に Confirm を発行します。この Confirm から CM_OPERATION_INCOMPLETE が戻されます。これは、関数が正常に発行され、非ブロッキング・モードで動作中であることを示します。
3. プログラムは、他の会話での CPI-C 関数の発行など、この会話に関係のない別の処理を実行できます。またこの会話についても、限定された範囲の CPI-C 関数 (Extract_* 関数など) を発行できます。この点が IBM CPI-C 2.0 の仕様とは異なります。IBM CPI-C 2.0 では、プログラムがこの会話に対して発行できるのは、Wait_For_Conversation または Cancel_Conversation 関数のみです。
4. しばらくすると、プログラムは Wait_For_Conversation を発行して、直前の非ブロッキング関数の完了を待ちます。パートナー・プログラムがまだ Confirmed を発行していないので、直前の Confirm 関数の処理はまだ完了していません。したがって、呼び出し側プログラムは中断されます。
5. パートナー・プログラムが Confirmed を発行すると、それによって呼び出し側プログラムの Confirm 関数の処理が完了します。このあと、Wait_For_Conversation 関数から制御権が戻ります。 *return_code* が CM_OK の場

合、Wait_For_Conversation が正常に完了したことを示します。
conversation_*return_code* が CM_OK の場合、(完了を待っていた) Confirm 関数が正常に完了したことを示します。

6. 呼び出し側プログラムは、追加データを送信したあとで Deallocate を発行します。この Deallocate から CM_OPERATION_INCOMPLETE が戻されます。これは、関数が正常に発行され、非ブロッキング・モードで動作中であることを示します。前と同様に、プログラムはこの会話に関係のない他の処理を実行できますが、この会話に対してはほとんどの CPI-C 関数は発行できません。
7. パートナー・プログラムは、Deallocate 要求を受信し、応答として Confirmed を発行します。これによって、Deallocate 関数の処理が完了します。
8. 呼び出し側プログラムは、会話に対する前の非ブロッキング関数の中に完了したものがあるかどうかを判断するために、Check_For_Completion を発行します。Deallocate 処理はすでに完了しているため、Check_For_Completion から、この会話の *conversation_ID* が戻されます。
9. プログラムは、次に Wait_For_Conversation を発行して、Deallocate 処理の結果を取得します。Deallocate 処理はすでに完了しているため、この関数からは即時に制御権が戻ります。

WINDOWS

アプリケーションは、非ブロッキング・モードで verb を発行する前に、Specify_Windows_Handle 関数を使用する必要があります。この関数は、verb の処理の完了時に CPI-C がメッセージを送信する Windows ハンドルを指定します。このメッセージは、アプリケーションに verb が完了したことを通知し、これにより、アプリケーションが verb 処理の結果を待つための追加のコールを発行する必要がありません。

CPI-C は別の方法を使用して、verb が完了したことを知らせることもできます。イベント・ハンドルのシグナリングです。アプリケーションが、WinCPICSetEvent を使用した会話にイベントを登録する場合、 そのアプリケーションは、Win32 関数の WaitForSingleObject または WaitForMultipleObjects を使用して、verb の完了通知を待つことができます。

未解決のコールが受信コールの場合、アプリケーションは受信が未解決である間、以下のコールを発行することができます。

- Request_To_Send
- Send_Error
- Test_Request_to_Send_Received
- Cancel_Conversation
- Deallocate

前述したように、Specify_Windows_Handle または WinCPICSetEvent を使用する代わりに、アプリケーションは AIX システムについては Wait_For_Conversation を使用することができます。この関数は、Windows システムが、他のオペレーティング・システム環境からのアプリケーションの移行を支援するために用意されています。ただし、Windows 環境における Wait_For_Conversation などのブロッキング機能の

使用は、おやめください。 Windows 環境に限定される新規のアプリケーションを作成している場合、 Specify_Windows_Handle を使用し、Wait_For_Conversation は使用しないでください。

**注:**

- Check_For_Completion は、AIX または Linux システム向けに述べたように、Windows システムではサポートされません。
- アプリケーションが、受信が未解決の間に非ブロッキング・モードで、上記のリストのコールのいずれかを使用する場合、 Specify_Windows_Handle を使用しなければならない。受信に加えて他のコールも未解決である場合、Wait_For_Conversation を発行することはできません。同一の会話で複数のコールが未解決である場合、このコールの結果は未定義になります。

## CPI-C と LU 6.2

CPI-C アプリケーションは、APPC などの CPI-C 以外の LU 6.2 アプリケーションと通信できます。

インプリメントされた LU 6.2 に次の機能が組み込まれていても、CPI-C ではサポートされません。

- 同期点 / バックアウト処理
- PIP データ
- LOCKS=LONG
- MAP_NAME
- FMH_DATA

CPI-C での通信の対象にする LU 6.2 アプリケーションでは、これらの機能は使用しないでください。

# 第 2 章 CPI-C アプリケーションの作成

この章には、CPI-C アプリケーション・プログラムを作成するために必要な情報が記載されています。この章で説明するトピックは次のとおりです。

- CPI-C コールの要約
- 初期会話特性
- サイド情報
- 構成
- CPI-C プログラムの TP 名とローカル LU 名の指定
- プログラムの開始方法

AIX、LINUX

- AIX または Linux に関する考慮事項
- Java CPI-C に関する考慮事項

WINDOWS

- Windows に関する考慮事項

- 移植可能なアプリケーションの作成

## CPI-C コールの要約

この節では、各 CPI-C コールについて簡単に説明します。コールは、機能別に分類してあります。特定のコールの詳細な説明については、53 ページの『第 3 章 CPI-C コール』を参照してください。

コールの「名前」として示したものは、便宜上付けてある呼称です。実際の C 関数名は、呼称の後ろの括弧内に示してあります。たとえば、Initialize_Conversation は、あるコールの呼称です。実際の関数名は cminit です。

プログラムで使用するローカル TP 名とローカル LU 名を設定しなければならない場合もあります。この設定の詳細については、37 ページの『ローカル TP 名の指定』と 38 ページの『ローカル LU の指定』を参照してください。

### 会話の開始

次のコールは、2 つのプログラム間の会話を開始するために使用します。この操作の詳細については、40 ページの『プログラムの開始方法』を参照してください。

プログラムで使用するローカル TP 名とローカル LU 名を設定しなければならない場合もあります。この設定の詳細については、37 ページの『ローカル TP 名の指定』と 38 ページの『ローカル LU の指定』を参照してください。

## WinCPICStartup

WINDOWS

このコールは、アプリケーションを Windows CPI-C アプリケーションとして登録し、 CPI-C ソフトウェアがそのアプリケーションに必要な関数のレベルをサポートするかどうかを判別します。 Windows CPI-C アプリケーションは、他の CPI-C コールを発行する前に、このコールを使用する必要があります。

## Initialize_Conversation (cminit)

このコールは、会話 ID を取得し、会話の特性の初期値を設定するために、呼び出し側プログラムが発行します。初期値は、シンボリック宛先名に関連付けられたサイド情報から抽出されるか、CPI-C のデフォルト値が使用されます。

## Initialize_For_Incoming (cminic)

このコールは、あとで Accept_Incoming を使用して受け入れる着呼会話の会話 ID を取得するために、呼び出し対象プログラムが使用します。このコールにより、プログラムは、常にブロッキング・モードで動作する Accept_Conversation を使用する代わりに、必要な場合には非ブロッキング・モードで Accept_Incoming を使用できるようになります。

## 初期会話特性を変更する Set_* コール

呼び出し側プログラムは、Initialize_Conversation コールを発行したあとで、表 7 に示したコールを発行して初期会話特性を変更できます。これらのコールを発行できるのは、初期化状態のときだけです。

*表 7. 初期会話特性を変更する Set_* コール*

| コール | 設定される項目 |
|---|---|
| Set_Conversation_Type (cmsct) | 会話タイプ |
| Set_Mode_Name (cmsmn) | モード名 |
| Set_Partner_LU_Name (cmspln) | パートナー LU 名 |
| Set_TP_Name (cmstpn) | パートナー・プログラムの TP 名 |
| Set_Return_Control (cmsrc) | 戻り制御 |
| Set_Sync_Level (cmssl) | 同期レベル |
| AIX、LINUX | |
| Set_Conversation_Context (cmsctx) | 会話コンテキスト (この会話を直前の会話と同じグループにします) |
| | |
| Set_Conversation_Security_Type (cmscst) | 会話セキュリティー・タイプ |
| Set_Conversation_Security_User_ID (cmscsu) | セキュリティー・ユーザー ID |

表 7. 初期会話特性を変更する Set_* コール (続き)

| コール | 設定される項目 |
|---|---|
| Set_Conversation_Security_Password (cmscsp) | セキュリティー・パスワード |

### Allocate (cmallc)

このコールは、現行の会話特性を使用してパートナー・プログラムとの会話を割り振るために、呼び出し側プログラムが発行します。割り振られる会話のタイプは、会話タイプの特性 (マップ式または基本) に応じて決まります。

### Accept_Conversation (cmaccp)

このコールは、着呼会話を受け入れ、特定の会話特性を設定するために、呼び出し対象プログラムが発行します。このコールが正常に実行されると、CPI-C が会話 ID を生成して戻します。Accept_Conversation は常にブロッキング・モードで動作します。

### Accept_Incoming (cmacci)

AIX、LINUX

このコールは、前に Initialize_For_Incoming を発行した対象の着呼会話を受け入れるために、呼び出し対象プログラムが発行します。これは Accept_Conversation に似ていますが、必要な場合には非ブロッキング・モードで動作できます (Accept_Conversation は常にブロッキング・モードで動作します)。

## データの送信

次のコールは、パートナー・プログラムにデータを送信するために使用します。

### Set_Send_Type (cmsst)

このコールは、会話の送信タイプを設定します。送信タイプは、Send_Data コールでデータをどのように送信するかを指定します。Send_Data コールに、Flush、Confirm、Prepare_To_Receive、または Deallocate コールの機能を組み込む (Send_Data に続いて他のコールを発行するのと同じ) ことも、他の機能を実行せずに単にデータを送信することもできます。指定した送信タイプ値は、後続のすべての Send_Data コールに影響を与えます。値を変更するには、Set_Send_Type コールを再発行します。

### Send_Data (cmsend)

このコールは、パートナー・プログラムに伝送するためのデータをローカル LU の送信バッファーに入れます。

送信タイプ (Set_Send_Type コールにより指定された) に Flush、Confirm、Prepare_To_Receive、または Deallocate コールの機能が組み込まれている場合は、データはパートナー LU (およびパートナー・プログラム) に即時に伝送されます。そ

のようなコールの機能が組み込まれていない場合は、データはローカル LU の送信バッファーに累積され、次のどちらかの状態が発生したときに送信されます。

- 送信バッファーが満ぱいになる
- ローカル・プログラムが次のコールのいずれかを発行し、それによって LU の送信バッファーがフラッシュされる。
  - Flush
  - Confirm
  - Deallocate
  - Prepare_To_Receive
  - Receive (受信タイプが CM_RECEIVE_AND_WAIT に設定されたもの)

### Flush (cmflus)

このコールは、ローカル LU の送信バッファーの内容をパートナー LU (およびプログラム) に送信します。送信バッファーが空のときは、アクションは何も実行されません。

### Confirm (cmcfm)

このコールは、ローカル LU の送信バッファーの内容と確認要求をパートナー・プログラムに送信し、確認を待ちます。

### Request_To_Send (cmrts)

このコールは、ローカル・プログラムがデータの送信を待っていることをパートナー・プログラムに通知します。パートナー・プログラムはこの要求に対して、受信状態に変更することにより、ローカル・プログラムを送信状態に変更することも、あるいはこの要求を無視することもできます。

## データの受信

プログラムは、次のコールを使用して、パートナー・プログラムからデータを受信することができます。

### Set_Prepare_To_Receive_Type (cmsptr)

このコールは、会話の受信準備タイプを設定します。このタイプにより、後続の Prepare_To_Receive コールに Flush または Confirm の機能が含まれるかどうかが指定されます。受信準備タイプは、後続のすべての Prepare_To_Receive コールに影響を与えます。このタイプを変更するには、Set_Prepare_To_Receive_Type コールを再発行します。

### Prepare_To_Receive (cmptr)

このコールは、ローカル・プログラムの会話の状態を送信から受信に変更して、ローカル・プログラムがデータの受信を開始できるようにします。このコールは、会話状態を変更する前までは、Flush コール、または Confirm コールと同じ操作を実行します。

### Set_Receive_Type (cmsrt)

このコールは、会話の受信タイプを設定します。これによって、Receive コールを発行するプログラムが、データが有効でない場合にデータの到着を待つかどうかが決

まります。指定した受信タイプ値は、後続のすべての Receive コールに影響を与えます。値を変更するには、Set_Receive_Type コールを再発行します。

### Receive (cmrcv)

会話が受信状態のときにこのコールを発行すると、ローカル・プログラムは現在受信可能なすべてのデータをパートナー・プログラムから受信します。受信可能なデータがない場合、受信タイプが CM_RECEIVE_AND_WAIT に設定されていれば、ローカル・プログラムはデータが到着するまで待機します。受信タイプが CM_RECEIVE_IMMEDIATE に設定されている場合は、ローカル・プログラムはデータの到着を待ちません。

会話が送信状態または送信 - 保留状態のときにこのコールを発行できるのは、受信タイプが CM_RECEIVE_AND_WAIT に設定されている場合だけです。このコールにより、LU の送信バッファーがフラッシュされ、会話状態が受信に変わります。これで、ローカル・プログラムはデータの受信を開始します。

### Set_Fill (cmsf)

このコールは、会話の充てんタイプを設定します。このタイプにより、プログラムがデータを論理レコードの形式で受信するか、指定長のデータとして受信するかが決まります。このコールが有効なのは、基本会話の場合だけです。充てん値は、後続のすべての Receive コールに影響を与えます。値を変更するには、Set_Fill コールを再発行します。

## ASCII と EBCDIC 間のデータ変換

プログラムは、次のコールを使用して、ローカル・データをパートナー・プログラムに送信する前に ASCII から EBCDIC に変換したり、パートナー・プログラムから受信したデータを EBCDIC から ASCII に変換することができます。プログラムがこれらの機能を使用する必要があるのは、パートナー・プログラムでデータが EBCDIC であることが必要な場合だけです。

### Convert_Incoming (cmcnvi)

このコールは、EBCDIC データ・ストリングを ASCII に変換します。

### Convert_Outgoing (cmcnvo)

このコールは、ASCII データ・ストリングを EBCDIC に変換します。

WINDOWS

プログラムは、CSV CONVERT verb を使用して、ASCII と EBCDIC 間のデータの変換をすることもできます。詳しくは、「*Communications Server for AIX Common Service Verb プログラマーズ・ガイド*」を参照してください。

## データ受信の確認とエラーの報告

次のコールは、データの受信を確認する、またはエラーを報告します。

### Confirmed (cmcfmd)

このコールは、パートナー・プログラムからの確認要求への応答を送ります。このコールは、パートナー・プログラムに対して、ローカル・プログラムが受信データ内にエラーを検出しなかったことを通知します。確認要求を発行したプログラムは確認を待つので、Confirmed コールにより、2 つのプログラムの処理が同期化されます。

### Set_Error_Direction (cmsed)

このコールは、プログラムがエラーを検出したのが、データの受信中であるのか、データの送信準備中であるのかを示します。このエラーの通信情報は、プログラムが送信 - 保留状態で Send_Error コールを発行したときにだけ有効です。

### Set_Log_Data (cmsld)

このコールは、パートナー LU に送信するログ・メッセージ (ログ・データ) とその長さを指定します。このコールが有効なのは、基本会話の場合のみです。Send_Error コールが発行されたとき、または会話が突然割り振り解除されたときに、ログ・データがあればそれが送信されます。ログ・データの送信が終わると、CPI-C がログ・データをヌルに、ログ・データ長を 0 にリセットします。

### Send_Error (cmserr)

このコールは、ローカル・プログラムが、パートナー・プログラムにアプリケーション・レベルのエラーを検出したことを通知します。ローカル・プログラムは、Send_Error コールを使用して、受信データにエラーが見つかったことをパートナー・プログラムに通知したり、確認要求をリジェクトしたり、または送信中の不完全な論理レコードを切り捨てることができます。

## 非ブロッキング・モードでのコールの発行

AIX、LINUX

この節は、Java CPI-C には適用されません。Java CPI-C 関数は常にブロッキング・モードで動作します。すなわちこのモードでは、Java CPI-C 関数は、要求された処理が完了するまでアプリケーションに制御を戻しません。この節で説明した関数は Java CPI-C では使用できません。

プログラムは、次のコールを使用して、後続の CPI-C コールが非ブロッキング・モードで動作できることを指定する、直前の非ブロッキング・コールが完了したかどうかを検査する、または非ブロッキング・コールの完了を待つことができます。

非ブロッキング・モードの詳細については、41 ページの『AIX または Linux に関する考慮事項』 と 46 ページの『Windows に関する考慮事項』 を参照してください。( 29 ページの『Cancel_Conversation (cmcanc)』も参照してください。このコールは、直前の非ブロッキング・コールを取り消し、会話を割り振り解除します。)

### Set_Processing_Mode (cmspm)

このコールは、会話の処理モードをブロッキング (処理が完了するまでコールが戻らない) または非ブロッキング (処理が完了していなくてもコールは即時に戻ることができる) に設定します。

### Check_For_Completion (cmchck)

AIX、LINUX

このコールは、処理が完了したプログラムの会話の中に、未解決の非ブロッキング関数の対象になっているものがあるかどうかを検査します。そのような関数がある場合は、該当する会話の会話 ID が戻されます。この場合、プログラムは、Wait_For_Conversation を呼び出して、非ブロッキング関数の結果を受け取ることができます。このコールを使用することによって、プログラムは中断しないで非ブロッキング関数の完了を検査できます (Wait_For_Conversation の場合は、関数が完了するまで中断されます)。Check_For_Completion は、直前のコールの結果を戻しません。結果を入手するには、この会話に対して後続のコールを発行する前に、プログラムで Wait_For_Conversation を使用する必要があります。

### Wait_For_Conversation (cmwait)

このコールは、直前の非ブロッキング関数の処理が完了するまで待機します。プログラムが複数同時会話にかかわっている場合は、このコールは、すべての会話に対して働き、すべての会話で関数が完了すると戻ります。

WINDOWS

Wait_For_Conversation コールは、Windows システム上での他の Windows CPI-C インプリメンテーションとの互換性をサポートします。ただし、新規の Windows アプリケーションは、このコールではなく、Specify_Windows_Handle を使用してください。

### Specify_Windows_Handle (xchwnd)

このコールは、CPI-C が非ブロッキング関数の結果を通知する Windows ハンドルを指定します。アプリケーションは、非ブロッキング関数の完了時に CPI-C からこの Windows ハンドルに送信されたメッセージを受信します。
Wait_For_Conversation を使用して、verb 完了結果を取得する必要はありません。

## ブロッキング・モードでのコールの発行

以下のコールにより、Windows プログラムは、後続の CPI-C コールのブロッキング・モードでの動作の仕方を管理することができます。(『Set_Processing_Mode (cmspm)』 も参照してください。) これは、後続のコールがブロッキング・モードまたは非ブロッキング・モードのどちらで動作するかを指定します。) ブロッキング・コールについての詳細は、 48 ページの『ブロッキング・コール』を参照してください。

**WinCPICIsBlocking**
このアプリケーションについて未解決のブロッキング CPI-C コールがあるかどうかを確認します。

**WinCPICSetBlockingHook**
CPI-C がブロッキング・コールを処理中に使用する、ブロッキング・プロシージャーを指定します。これは、CPI-C のデフォルトのブロッキング・プロシージャーを置き換えます。ブロッキング・プロシージャーは、 CPI-C がコールの処理を完了するまで、繰り返し呼び出されます。

**WinCPICUnhookBlockingHook**
直前の WinCPICSetBlockingHook コールにより指定されたブロッキング・プロシージャーを未登録にし、CPI-C がデフォルトのブロッキング・プロシージャーを使用するように戻します。

## 情報の取得

次のコールは、プログラムに情報を提供します。

### Extract_* コール

表 8 に示した Extract_* コールは、指定の会話の特性に関する情報を検索します。

*表 8. Extract_* コールとそのアクション*

| コール | 検索される項目 |
|---|---|
| Extract_Conversation_Security_Type (xcecst)(Java CPI-C では使用できません) | セキュリティー・タイプ |
| Extract_Conversation_State (cmecs) | 会話状態 |
| Extract_Conversation_Type (cmect) | 会話タイプ |
| AIX、LINUX | |
| Extract_Conversation_Context (cmectx) | 会話コンテキスト |
| Extract_Max_Buffer_Size (cmembs) | Send_Data コールと Receive コールに使用されるデータ・バッファーの最大サイズ |
| Extract_Security_User_ID (cmesui) | セキュリティー・ユーザー ID |
| WINDOWS | |
| Extract_Conversation_Security_User_ID (cmecsu) | セキュリティー・ユーザー ID |
| | |
| Extract_Mode_Name (cmemn) | モード名 |
| Extract_Partner_LU_Name (cmepln) | パートナー LU 名 |
| Extract_TP_Name (cmetpn) | 着呼 Allocate 要求に指定されている TP 名 |
| Extract_Sync_Level (cmesl) | 同期レベル |

### Test_Request_to_Send_Received (cmtrts)

このコールは、パートナー・プログラムから送信要求 (RS) 通知を受信したかどうかを判別します。

## 会話の終了

次のコールは会話を終了させます。

### Set_Deallocate_Type (cmsdt)

この呼び出しは、会話の割り振りを解除する方法を指定します。このコールにより指定された割り振り解除命令が有効になるのは、Deallocate コールが発行されたとき、または送信タイプが CM_SEND_AND_DEALLOCATE に設定されていて Send_Data コールが発行されたときです。

### Deallocate (cmdeal)

このコールは、2 つのプログラム間の会話の割り振りを解除します。このコールは、会話の割り振りを解除する前に、現在の会話同期レベルと割り振り解除のタイプに応じて、Flush コール、または Confirm コールと同じ操作を実行します。

### Cancel_Conversation (cmcanc)

このコールは、会話に対する不完全コールを取り消し、その会話の割り振りを解除します (不完全コールとは、非ブロッキング・モードのときに発行され、CM_OPERATION_INCOMPLETE を戻したコールです)。

Java CPI-C では非ブロッキング・コールがサポートされていないので、不完全コールが未解決になることはあり得ません。Cancel_Conversation は、ログ・データをローカル・エラー・ログに書き込まない点以外は Deallocate と同じです。

### WinCPICCleanup

WINDOWS

このコールは、アプリケーションが CPI-C コールの発行を完了した後に、そのアプリケーションを Windows CPI-C アプリケーションとして未登録にします。Windows CPI-C アプリケーションは、終了前にこのコールを使用する必要があり、このコールを発行後に他の CPI-C コールを発行することはできません。

## サイド情報の管理

これらの関数は Java CPI-C では使用できません。

表 9 に要約して示したコールは、サイド情報エントリーを追加、置換、検索、または削除するために CPI-C アプリケーションで使用できるコールです。

*表 9. サイド情報の追加、置換、検索、または削除のためのコール*

| コール | アクション |
|---|---|
| Set_CPIC_Side_Information (xcmssi) | サイド情報エントリーを追加または置換します。 |
| Extract_CPIC_Side_Information (xcmesi) | サイド情報エントリーを検索します。 |
| Delete_CPIC_Side_Information (xcmdsi) | サイド情報エントリーを削除します。 |

# 初期会話特性

CPI-C は、それぞれの会話ごとに、特性と呼ばれる一連の内部値を保持しています。特性のなかには、会話タイプのように、会話の全体的な動作に影響を与えるものもあります。また、受信タイプのように、特定のコールの動作のみに影響を与えるものもあります。

これらの特性の多くは、最初は CS/AIX 構成ファイルに保管されているサイド情報から取り出されます。 34 ページの『サイド情報』を参照してください。 Initialize_Conversation コールは、必要なサイド情報テーブル・エントリーに関連付けられているシンボリック宛先名 (*sym_dest_name* パラメーター) を指定します。

表 10 は、会話特性、次の会話開始コールによってその特性がどのように設定または変更されるか、および特定の値を変更できるのはどのコールかを示します。

- Initialize_Conversation
- Accept_Conversation
- Initialize_For_Incoming
- Accept_Incoming

AIX、LINUX

Initialize_For_Incoming コールと Accept_Incoming コールは、常に一緒に使用します。つまり、1 つの特性は、通常、この 2 つのコールの一方で設定され、他方のコールでその特性が変更されることはありません。

WINDOWS

Initialize_For_Incoming および Accept_Incoming コールは、Windows システム上ではサポートされていません。これらのコールに対するすべての参照は、Windows システムでは無視されます。

特性の詳細な説明については、 53 ページの『第 3 章 CPI-C コール』に記載されている、特性に関連した Set_* コールに関する説明を参照してください。 たとえば、会話タイプの説明は、Set_Conversation_Type コールの節にあります。

*表 10. 初期会話特性の変更*

| 会話状態 | |
|---|---|
| Initialize_Conversation で設定: | CM_INITIALIZE_STATE |
| Accept_Conversation で設定: | CM_RECEIVE_STATE |
| Initialize_For_Incoming で設定: | CM_INITIALIZE_INCOMING_STATE |
| Accept_Incoming で設定: | CM_RECEIVE_STATE |

*表 10. 初期会話特性の変更 (続き)*

| | |
|---|---|
| 変更するために使用できるコール: | 多数の CPI-C コール。このコールから生じる状態変更については、 53 ページの『第 3 章 CPI-C コール』の各 CPI-C コールの説明の末尾にある、 *状態変更*セクションを参照してください。 |
| 会話タイプ | |
| Initialize_Conversation で設定: | CM_MAPPED_CONVERSATION |
| Accept_Conversation で設定: | 呼び出し側プログラムにより指定された値 |
| Initialize_For_Incoming で設定: | (設定されません) |
| Accept_Incoming で設定: | 呼び出し側プログラムにより指定された値 |
| 変更するために使用できるコール: | Set_Conversation_Type |
| 割り振り解除タイプ | |
| Initialize_Conversation で設定: | CM_DEALLOCATE_SYNC_LEVEL |
| Accept_Conversation で設定: | CM_DEALLOCATE_SYNC_LEVEL |
| Initialize_For_Incoming で設定: | CM_DEALLOCATE_SYNC_LEVEL |
| Accept_Incoming で設定: | (変更されません) |
| 変更するために使用できるコール: | Set_Deallocate_Type |
| エラー時の通信方向 | |
| Initialize_Conversation で設定: | CM_RECEIVE_ERROR |
| Accept_Conversation で設定: | CM_RECEIVE_ERROR |
| Initialize_For_Incoming で設定: | CM_RECEIVE_ERROR |
| Accept_Incoming で設定: | (変更されません) |
| 変更するために使用できるコール: | Set_Error_Direction |
| 充てん | |
| Initialize_Conversation で設定: | CM_FILL_LL |
| Accept_Conversation で設定: | CM_FILL_LL |
| Initialize_For_Incoming で設定: | CM_FILL_LL |
| Accept_Incoming で設定: | (変更されません) |
| 変更するために使用できるコール: | Set_Fill |
| ログ・データ | |
| Initialize_Conversation で設定: | ヌル・ストリング |
| Accept_Conversation で設定: | ヌル・ストリング |
| Initialize_For_Incoming で設定: | ヌル・ストリング |
| Accept_Incoming で設定: | (変更されません) |
| 変更するために使用できるコール: | Set_Log_Data |
| ローカル LU 名 | |
| Initialize_Conversation で設定: | 多数の異なるソースの 1 つからのローカル LU の別名 ( 38 ページの『ローカル LU の指定』を参照) |
| Accept_Conversation で設定: | 会話開始要求が到着したセッションの LU の別名 |
| Initialize_For_Incoming で設定: | (設定されません) |
| Accept_Incoming で設定: | 会話開始要求が到着したセッションの LU の別名 |
| 変更するために使用できるコール: | Set_Local_LU_Name |
| モード名 | |

*表 10. 初期会話特性の変更 (続き)*

| | |
|---|---|
| Initialize_Conversation で設定: | サイド情報からのモード名、または、*sym_dest_name* が指定されていない場合はヌル・ストリング |
| Accept_Conversation で設定: | 会話開始要求が到着したセッションのモード名 |
| Initialize_For_Incoming で設定: | (設定されません) |
| Accept_Incoming で設定: | 会話開始要求が到着したセッションのモード名 |
| 変更するために使用できるコール: | Set_Mode_Name |
| **パートナー LU 名** | |
| Initialize_Conversation で設定: | サイド情報からのパートナー LU 名、または *sym_dest_name* が指定されていない場合は 1 個のブランク |
| Accept_Conversation で設定: | 会話開始要求が到着したセッションのパートナー LU 名 |
| Initialize_For_Incoming で設定: | (設定されません) |
| Accept_Incoming で設定: | 会話開始要求が到着したセッションのパートナー LU 名 |
| 変更するために使用できるコール: | Set_Partner_LU_Name |
| **受信準備タイプ** | |
| Initialize_Conversation で設定: | CM_PREP_TO_RECEIVE_SYNC_LEVEL |
| Accept_Conversation で設定: | CM_PREP_TO_RECEIVE_SYNC_LEVEL |
| Initialize_For_Incoming で設定: | CM_PREP_TO_RECEIVE_SYNC_LEVEL |
| Accept_Incoming で設定: | (変更されません) |
| 変更するために使用できるコール: | Set_Prepare_To_Receive_Type |
| **処理モード (ブロッキングまたは非ブロッキング)** | |
| Initialize_Conversation で設定: | CM_BLOCKING |
| Accept_Conversation で設定: | CM_BLOCKING |
| Initialize_For_Incoming で設定: | CM_BLOCKING |
| Accept_Incoming で設定: | (変更されません) |
| 変更するために使用できるコール: | Set_Processing_Mode |
| **受信タイプ** | |
| Initialize_Conversation で設定: | CM_RECEIVE_AND_WAIT |
| Accept_Conversation で設定: | CM_RECEIVE_AND_WAIT |
| Initialize_For_Incoming で設定: | CM_RECEIVE_AND_WAIT |
| Accept_Incoming で設定: | (変更されません) |
| 変更するために使用できるコール: | Set_Receive_Type |
| **戻り制御** | |
| Initialize_Conversation で設定: | CM_WHEN_SESSION_ALLOCATED |
| Accept_Conversation で設定: | (適用されません) |
| Initialize_For_Incoming で設定: | (適用されません) |
| Accept_Incoming で設定: | (適用されません) |
| 変更するために使用できるコール: | Set_Return_Control |
| **セキュリティー・パスワード** | |
| Initialize_Conversation で設定: | サイド情報に含まれているパスワード、または *sym_dest_name* が指定されていない場合は 1 個のブランク |

*表 10. 初期会話特性の変更 (続き)*

| | |
|---|---|
| Accept_Conversation で設定: | (適用されません) |
| Initialize_For_Incoming で設定: | (適用されません) |
| Accept_Incoming で設定: | (適用されません) |
| 変更するために使用できるコール: | Set_Conversation_Security_Password |
| **セキュリティー・タイプ** | |
| Initialize_Conversation で設定: | サイド情報に含まれているセキュリティー・タイプ、または *sym_dest_name* が指定されていない場合は `CM_SECURITY_SAME` |
| Accept_Conversation で設定: | (適用されません) |
| Initialize_For_Incoming で設定: | (適用されません) |
| Accept_Incoming で設定: | (適用されません) |
| 変更するために使用できるコール: | Set_Conversation_Security_Type |
| **セキュリティー・ユーザー ID** | |
| Initialize_Conversation で設定: | サイド情報に含まれているユーザー ID、または *sym_dest_name* が指定されていない場合は 1 個のブランク |
| Accept_Conversation で設定: | 呼び出し側プログラムにより指定された値 |
| Initialize_For_Incoming で設定: | (設定されません) |
| Accept_Incoming で設定: | 呼び出し側プログラムにより指定された値 |
| 変更するために使用できるコール: | Set_Conversation_Security_User_ID |
| **送信タイプ** | |
| Initialize_Conversation で設定: | `CM_BUFFER_DATA` |
| Accept_Conversation で設定: | `CM_BUFFER_DATA` |
| Initialize_For_Incoming で設定: | `CM_BUFFER_DATA` |
| Accept_Incoming で設定: | (変更されません) |
| 変更するために使用できるコール: | Set_Send_Type |
| **同期レベル** | |
| Initialize_Conversation で設定: | `CM_NONE` |
| Accept_Conversation で設定: | 呼び出し側プログラムにより指定された値 |
| Initialize_For_Incoming で設定: | (設定されません) |
| Accept_Incoming で設定: | 呼び出し側プログラムにより指定された値 |
| 変更するために使用できるコール: | Set_Sync_Level |
| **呼び出し対象プログラムの TP 名 (呼び出し側プログラムから見た場合)** | |
| Initialize_Conversation で設定: | サイド情報に含まれている TP 名、または *sym_dest_name* が指定されていない場合は 1 個のブランク |
| Accept_Conversation で設定: | (適用されません) |
| Initialize_For_Incoming で設定: | (適用されません) |
| Accept_Incoming で設定: | (適用されません) |
| 変更するために使用できるコール: | Set_TP_Name |
| **呼び出し対象プログラムの TP 名 (呼び出し対象プログラムから見た場合)** | |
| Initialize_Conversation で設定: | (適用されません) |
| Accept_Conversation で設定: | 呼び出し側プログラムにより指定された値 |
| Initialize_For_Incoming で設定: | (設定されません) |
| Accept_Incoming で設定: | 呼び出し側プログラムにより指定された値 |
| 変更するために使用できるコール: | Specify_Local_TP_Name (着呼割り振りを受け入れる 1 つ以上の名前を指示するため) |

# サイド情報

2 つのプログラムが通信するために必要な情報は、CS/AIX 構成ファイル内の CPI-C サイド情報エントリーに保管されています。システム管理者に相談して、必要な情報がこの構成ファイルに確実に含まれるようにする必要があります。構成に関する追加情報については、「*Communications Server for AIX 管理ガイド*」を参照してください。

各サイド情報エントリーは、シンボリック宛先名、つまり Initialize_Conversation コールで指定される *sym_dest_name* パラメーターによって識別されます。パラメーター *sym_dest_name* は、8 バイトの ASCII 文字ストリングで、表示可能な文字はすべて使用することができます。

商用プログラム、または企業内の複数のマシンにインストールするプログラムを開発している場合は、プログラムのコピーごとに異なる *sym_dest_name* を使用するための論理を組み込むことができます。

ここのサイド情報エントリーには、次のフィールドがあります。

- ローカル LU 別名
- パートナー LU 名
- パートナー・プログラムのタイプと名前
- モード名
- 会話セキュリティー・タイプ
- セキュリティー・ユーザー ID とパスワード
- アプリケーション指定のサイド情報

## ローカル LU 別名

これは、会話を割り振るために使用されるローカル LU の別名です。この別名は最大 8 文字の ASCII 文字で構成されます。指定できる文字については、 146 ページの『Set_Local_LU_Name (cmslln)』を参照してください。

## パートナー LU 名

これは、ローカル・プログラムがパートナー LU を識別するために使用する名前です。これは、最大 8 文字の ASCII 文字の別名、または最大 17 文字の完全修飾ネットワーク名です。指定できる文字については、 152 ページの『Set_Partner_LU_Name (cmspln)』を参照してください。

## パートナー・プログラムのタイプと名前

これらのフィールドは、パートナー・プログラムがアプリケーション・プログラムであるか SNA サービス・プログラムであるかを示し、パートナー・プログラムの名前も示します。アプリケーション・プログラム名には、最大 64 文字の ASCII 文字を使用できます。サービス・プログラムには最大 4 文字を使用できます。指定できる文字については、 164 ページの『Set_TP_Name (cmstpn)』を参照してください。

## モード名

この名前は、LU-LU セッションで使用する一連の特性を表します。モード名には、最大 8 文字の ASCII 文字を使用できます。指定できる文字については、 150 ページの『Set_Mode_Name (cmsmn)』を参照してください。

## 会話セキュリティー・タイプ

このフィールドは、セキュリティーを使用するかどうか、使用する場合はどのタイプかを示します。セキュリティー・タイプで、呼び出し対象プログラムとの会話を割り振るときに、CPI-C がユーザー ID とパスワードを送信する必要があることを指定できます。呼び出し対象プログラムがさらに別のプログラムを呼び出す場合は、セキュリティー・タイプにより、2 番目の呼び出し対象プログラムに、セキュリティーが検査済みであることを通知できます。

会話セキュリティーの詳細については、 130 ページの『Set_Conversation_Security_Type (cmscst)』を参照してください。

## セキュリティー・ユーザー ID とパスワード

リモート・プログラムが会話セキュリティーを使用していて、「検査済み」の指示を受け入れない場合、呼び出し対象プログラムにアクセスするには、有効な組み合わせのユーザー ID とパスワードが必要です。ユーザー ID とパスワードには、最大 10 文字の ASCII 文字を使用できます。指定できる文字については、 132 ページの『Set_Conversation_Security_User_ID (cmscsu)』および 127 ページの『Set_Conversation_Security_Password (cmscsp)』を参照してください。

## アプリケーション指定のサイド情報

AIX、LINUX

**注:** この節で説明した関数は Java CPI-C では使用できません。Java CPI-C アプリケーションは、アプリケーション自身の CPI-C サイド情報エントリーを保守することはできません。ただし、Java CPI-C アプリケーションは、要求されているパラメーターごとに Set_* 関数または Extract_* 関数を使用することによって、サイド情報の個々のパラメーターを変更したり、その値を決定することはできます。

アプリケーションは、次のコールを使用して、構成ファイルに保管されているサイド情報を変更し、独自のサイド情報エントリーを保持することができます。

- Set_CPIC_Side_Information (指定の *sym_dest_name* に関連するサイド情報エントリーを定義します。*sym_dest_name* がすでに構成ファイルに定義されている場合は、構成ファイルは新しい情報に置き換えられます。)
- Delete_CPIC_Side_Information (アプリケーションにより定義されたエントリー、または構成ファイル内に定義されているエントリーは、このアプリケーションでは使用できなくなったことを示します。)

- Extract_CPIC_Side_Information (サイド情報エントリーの内容を戻します。このエントリーは、アプリケーションにより定義されているか、構成ファイル内に定義されています。)

変更後の情報は、このアプリケーションだけに適用されます。他のアプリケーションに影響を与えたり、構成ファイルの内容を変更したりすることはありません。変更後の情報は、アプリケーションの終了時に破棄されます。

これらのコールは IBM CPI-C 2.0 の一部ではありませんが、X/Open CPI-C との互換性を確保するために提供されています。また、これらのコールで使用されるサイド情報構造では、ユーザー ID とパスワードは 10 文字 (IBM CPI-C 2.0 の場合のように) でなく、8 文字として (X/Open CPI-C の場合のように) 定義されます。そのため、次のような制約があります。

- パートナー・アプリケーションで 8 文字を超えるユーザー ID またはパスワードが必要な場合は、Set_CPIC_Side_Information を使用して指定することはできません。構成ファイル内に定義されているサイド情報エントリーを使用するか、または Set_CPIC_Side_Information を使用して新規に定義し、次に Set_Conversation_Security_User_ID コール、または Set_Conversation_Security_Password コールを使用して構成ファイル内のユーザー ID またはパスワードを変更する必要があります。
- 構成ファイル内のサイド情報エントリーに 8 文字を超えるユーザー ID が入っている場合は、Extract_CPIC_Side_Information を使用してその ID を取り出すことはできません。Extract_Security_User_ID コールを使用する必要があります。(これはパスワードには適用されません。なぜなら、CPI-C ではアプリケーションでパスワードを取り出すことはできないからです。)

## 構成

CS/AIX を構成する場合の考慮事項は次のとおりです。

- CPI-C アプリケーションが CS/AIX の LU 6.2 サービスを使用できるようにするには、システム管理者は、サイド情報 (*sym_dest_name* により指定されたもの) を保持するほかに、構成時に次のエンティティーを定義する必要があります。
  - モード
  - ローカル LU
  - パートナー LU
  - 呼び出し可能 TP
  - セキュリティー・ユーザー ID とパスワード

  詳細については、「*Communications Server for AIX 管理ガイド*」を参照してください。
- 自動開始セッションを使用可能にする場合は、モードに *auto_act* パラメーターを設定します。モードの定義についての詳細は、「*Communications Server for AIX 管理ガイド*」を参照してください。

# ローカル TP 名の指定

プログラムが Initialize_Conversation コール、Initialize_Conversation_For_Incoming コール、または Accept_Conversation コールを発行すると、CPI-C ライブラリーでトランザクション・プログラム (TP) のインスタンスが 1 つ生成されます。この TP の名前は、以下に示しているように、さまざまな方法で指定することができます。

これらの方法は優先順位に従ってリストしています。最初の方法を使用して名前を指定すると、CPI-C ライブラリーはこの名前を使用し、2 番目以降の方法で指定した名前をすべて無視します。最初の方法を使用せず、2 番目の方法で名前を指定すれば、CPI-C ライブラリーはこの名前を使用し、3 番目以降の方法で指定した名前はすべて無視します。以下同様です。

- 呼び出し側プログラムでは、TP 名はログ・ファイルおよびトレース・ファイルの中で ID として使用されるだけです。
- オペレーターによって開始された呼び出し対象プログラムでは、インバウンド割り振り要求を適切なプログラムあてに経路指定するために TP 名が使用されるので、この値を正確に設定する必要があります。Accept_Conversation コール、または呼び出し対象プログラムからの Accept_Incoming コールは、この TP 名あてのインバウンド割り振り要求が到着したときに完了します。
- 自動的に開始した呼び出し対象プログラムの場合は、TP 名はインバウンド割り振り要求から取得されるので、TP 名を指定する必要はありません。

**注:** ローカル TP 名は、Set_TP_Name コールで設定されるパートナー TP 名とは別のものです。

## Specify_Local_TP_Name

プログラムでは、このコールを使用して TP 名を指定できます。

## コンテキスト

コンテキストをコピーする元になる別の TP がある場合は、TP 名はその別の TP からとられます。コンテキストの詳細については、13 ページの『複数会話』を参照してください。

## APPCTPN 環境変数

TP 名は、APPCTPN 環境変数を使用して指定できます。

AIX、LINUX

AIX または Linux システム上では、TP 名は APPCTPN 環境変数に指定します。この環境変数は次の方法で設定できます。

- プログラムから putenv コールを発行する。
- AIX または Linux シェルで設定する。たとえば、Korn シェル内で次のコマンドを発行できます。

```
export APPCTPN=MYTP
```

- 自動開始により呼び出される TP を使用している場合は、CS/AIX で呼び出し可能な TP データ・ファイルの環境フィールドを使用して設定する。

WINDOWS

Windows システム上では、TP 名は APPCTPN 環境変数またはレジストリーで指定することができます。 CPI-C は、最初に環境変数をチェックして、指定されていればこの名前を使用します。環境変数が指定されていない場合に限り、レジストリー項目を使用します。 Windows 端末サーバーを使用していて、異なるローカルの LU を使用する同一のアプリケーションの複数コピーを実行する必要がある場合には、環境変数を使用する必要があると思われます。

レジストリー・キーは、

¥¥HKEY_LOCAL_MACHINE¥SOFTWARE¥SNA Client¥SxClient¥Parameters¥MyExeName

ここで、MyExeName は **.exe** 拡張子なしのプログラムのファイル名です。

このレジストリー・キーにおける APPCTPN 値は、TP 名を示します。

## デフォルト値

上記の方法で TP 名を設定しなかった場合は、TP 名はデフォルト値 CPIC_DEFAULT_TPNAME に設定されます。

# ローカル LU の指定

呼び出し側の CPI-C TP が使用するローカル LU は、次のような方法で指定できます。

**注:** 呼び出し対象 TP のローカル LU は下記の方法では指定できません。割り振り要求に指定されたパートナー LU 値によって定義されます。

指定された LU が従属 LU の場合は、複数同時会話はサポートされません (従属 LU は複数セッションをサポートしないため)。

以下のセクションでは、ローカル LU の別名の設定に使用できるさまざまな方法を説明します。これらの方法は優先順位に従ってリストしています。つまり、最初の方法を使用してローカル LU の別名を指定すると、CPI-C ライブラリーはこの名前を使用し、2 番目以降の方法で指定した名前はすべて無視します。最初の方法を使用せず、2 番目の方法でローカル LU の別名を指定すれば、CPI-C ライブラリーはこの別名を使用し、3 番目以降の方法で指定した別名はすべて無視します。以下同様です。

## Set_Local_LU_Name

プログラムは、Initialize_Conversation コールの完了後にこのコールを発行して、ローカル LU の別名を指定できます。このコールは発行元の TP だけに影響を与えます。構成ファイルに保管されているサイド情報が変更されることはありません。

**注:** このコールは標準 CPI-C 仕様の一部ではありません。他のインプリメンテーションでは使用できない場合があります。アプリケーションを他の CPI-C インプリメンテーションで使用できるようにする場合は、この関数は使用しないか、簡単に変更できる数個の特定のルーチンに限定する必要があります。

## コンテキスト

コンテキストをコピーする元になる別の TP がある場合は、ローカル LU 名はその別の TP からとられます。コンテキストの詳細については、13 ページの『複数会話』を参照してください。

## APPCLLU 環境変数

ローカル LU の別名は、APPCLLU 環境変数を使用して指定できます。

AIX、LINUX

AIX または Linux システムでは、環境変数は次の方法で設定できます。

- プログラムから putenv コールを発行する。
- AIX または Linux シェルで設定する。たとえば、Korn シェル内で次のコマンドを発行できます。

  **export APPCLLU=MYLU**

WINDOWS

Windows システムでは、ローカル LU の別名は、 APPCLLU 環境変数の使用するか、またはレジストリーのどちらかで指定できます。CPI-C は、最初に環境変数をチェックして、指定されていればこの別名を使用します。環境変数が指定されていない場合に限り、レジストリー項目を使用します。 Windows 端末サーバーを使用していて、異なるローカルの LU を使用する同一のアプリケーションの複数コピーを実行する必要がある場合には、環境変数を使用する必要があると思われます。

レジストリー・キーは、

¥¥HKEY_LOCAL_MACHINE¥SOFTWARE¥SNA Client¥SxClient¥Parameters¥MyExeName

ここで、MyExeName は **.exe** 拡張子なしのプログラムのファイル名です。

このレジストリー・キーにおける APPCLLU 値は、ローカル LU 別名を示します。

### サイド情報

ローカル LU の別名は、シンボリック宛先名ごとに構成されたサイド情報の一部です。TP は、そのどれを Initialize_Conversation コールで使用するかを選択します。

**注:** サイド情報はプログラムで変更できます。詳しくは、29 ページの『サイド情報の管理』を参照してください。

### デフォルトのローカル LU

ローカル LU は、APPC LU のデフォルト・プールの一部として構成できます。ローカル LU の別名をほかに何も指定しなかった場合は、このプールの中から適切な LU が使用されます。

### 制御点 LU

CS/AIX では、通常、各ノードに 1 つずつの制御点 (CP) LU が定義されています。ローカル LU の別名を他に何も指定しなかった場合は、CP LU が使用されます。

## プログラムの開始方法

会話は、呼び出し側プログラムと呼び出し対象プログラムの間に発生します。呼び出し側プログラムは、コマンドを入力したユーザー、またはバッチ・コマンドによって開始されます。呼び出し対象プログラムは、ユーザーが手操作で開始するか、または CS/AIX で自動的に開始することができます。

### 呼び出し対象プログラム: 自動開始の場合

呼び出し対象プログラムは、次のいずれかの条件のもとで、自動的に開始されるように構成できます。

- 呼び出し対象プログラムにサービスを提供する LU が、最初にインバウンド割り振り要求を受信したとき。この方法で開始されたプログラムを、自動開始待機プログラム (または自動開始待機 TP) といいます。

  呼び出し対象プログラムが実行中でない場合は、最初のインバウンド割り振り要求がそのプログラムを開始します。割り振り要求に対する応答は、呼び出し対象プログラム内の Accept_Conversation コール、または Accept_Incoming コールが実行されるまで保留されます。

  呼び出し対象プログラムがすでに実行中の場合は、インバウンド割り振り要求は、呼び出し対象プログラムが次の Accept_Conversation コール、または Accept_Incoming コールを発行するまで待つか、または呼び出し対象プログラムが実行を終了し再始動されるまで待機します。
- 呼び出し対象プログラムにサービスを提供する LU がインバウンド割り振り要求を受信するたびに、プログラムの新規インスタンスがロードされ開始されます。この方法で開始されたプログラムを自動開始非待機プログラムといいます。

一般に、インバウンド割り振り要求は、呼び出し対象プログラムが開始されて Accept_Conversation コール、または Accept_Incoming コールを発行するまで待機します。ただし、呼び出し対象プログラムのローカル LU の定義にタイムアウト値を

組み込んで、呼び出し対象プログラムが Accept_Conversation コール、または Accept_Incoming コールを発行する前にタイムアウトに達した場合は、インバウンド割り振り要求が失敗するように設定しておく必要があります。

呼び出し対象 TP の定義 (CS/AIX の呼び出し可能 TP データ・ファイルにあります) に、第 2 のタイムアウト値を組み込みます。このタイムアウト値により、Accept_Conversation コール、または Accept_Incoming コールがインバウンド割り振り要求を待つ時間が決まります。インバウンド割り振り要求を受信する前にこのタイムアウトに達すると、コールは失敗します。このタイムアウト値は、非待機プログラムには適用されません。この種のプログラムは常にインバウンド割り振り要求に応答して開始されるものであり、常に 1 つが保留になっています。

### 呼び出し対象プログラム: ユーザー開始の場合

呼び出し対象プログラムがユーザーが開始するように構成されていれば、その呼び出し対象プログラムは、呼び出し側プログラムの前でも後でも、ユーザーが開始できます。この方法で開始されるプログラムを、オペレーター開始待機プログラムといいます。

呼び出し対象プログラムが開始される前にユーザーが呼び出し側プログラムを開始した場合は、呼び出し対象プログラムに対するインバウンド割り振り要求は、呼び出し対象プログラムが開始されて Accept_Conversation コール、または Accept_Incoming コールを発行するまで待機します。ただし、呼び出し対象プログラムのローカル LU の定義にタイムアウト値を組み込んで、呼び出し対象プログラムが開始され、Accept_Conversation コール、または Accept_Incoming コールを発行する前にタイムアウトに達した場合は、インバウンド割り振り要求が失敗するように設定しておく必要があります。

呼び出し側プログラムが Allocate コールを発行する前にユーザーが呼び出し対象プログラムを開始した場合は、呼び出し対象プログラムから発行された Accept_Conversation コール、または Accept_Incoming コールは、インバウンド割り振り要求が届くのを待ちます。呼び出し対象 TP の定義 (CS/AIX の呼び出し可能 TP データ・ファイルにあります) に、第 2 のタイムアウト値を組み込みます。このタイムアウト値により、Accept_Conversation コール、または Accept_Incoming コールがインバウンド割り振り要求を待つ時間が決まります。インバウンド割り振り要求を受信する前にこのタイムアウトに達すると、コールは失敗します。

## AIX または Linux に関する考慮事項

AIX、LINUX

このセクションでは、AIX または Linux システム用の CPI-C アプリケーションの作成時に、考慮する必要のある情報を要約しています。

Java CPI-C アプリケーションを作成する場合は、43 ページの『Java CPI-C に関する考慮事項』を参照してください。

## CPI-C ヘッダー・ファイル

CPI-C アプリケーションで使用するヘッダー・ファイルは、**cmc.h** です。このファイルには、すべての CPI-C エントリー・ポイントの定義が含まれています。また、共通インターフェース・ヘッダー・ファイル **values_c.h** も含まれています。これらの 2 つのファイルには、CPI-C インターフェースでの指定パラメーターおよび戻りパラメーターの値に対して定義される、すべての定数が含まれます。両ファイルとも **/usr/include/sna** (AIX) または **/opt/ibm/sna/include** (Linux) に保管されます。

## マルチプロセス

会話を開始したプロセスが fork して子プロセスを作成する場合、その子プロセスは、親プロセスに戻された conversation_ID を使用することはできません。ただし、子プロセスは独自の Initialize_Conversation コール、 Initialize_For_Incoming コール、または Accept_Conversation コールを発行して、専用の conversation_ID を取得できます。

同一プログラムの複数のインスタンスを、異なるプロセスとして実行することができますが、各インスタンスには、それぞれ固有の conversation_ID が割り当てられます。

1 つのプロセスに複数の会話が入っていて、各会話が専用の conversation_ID を持っているアプリケーションを作成することができます。ただし、そのようなアプリケーションは慎重に設計して、「デッドロック」状況、つまり同じプロセス内の他の会話の状況が原因で CPI-C コールが完了できなくなってしまう状況を回避するようにしてください。この状況が起こるのは、プログラムの中のある会話が、情報待ちの状態でその情報が届いてからでなければデータを戻すことができず、同じプロセスの別の会話が、このデータを受け取ってからでないと最初の会話が本来必要としている情報を送信できないような場合です。このような状況は、それぞれの会話ごとに別個のプロセスを使用することによって、ある程度回避できます。

## CPI-C アプリケーションのコンパイルとリンク

### AIX アプリケーション

32 ビット・アプリケーションをコンパイルおよびリンクするには、次のオプションを使用します。

**-bimport:/usr/lib/sna/cpic_r.exp -I**
**/usr/include/sna**

64 ビット・アプリケーションをコンパイルおよびリンクするには、次のオプションを使用します。

**-bimport:/usr/lib/sna/cpic_r64_5.exp -I**
**/usr/include/sna**

### Linux アプリケーション

CPI-C アプリケーションのコンパイルおよびリンクの前に、共用ライブラリーが保管されているディレクトリーを指定して、アプリケーションが実行時に共用ライブ

ラリーを検出できるようにします。このためには、環境変数 LD_RUN_PATH を **/opt/ibm/sna/lib** に、64 ビット・アプリケーションをコンパイルしている場合には **/opt/ibm/sna/lib64** に設定します。

32 ビット・アプリケーションをコンパイルおよびリンクするには、次のオプションを使用します。

**-I /opt/ibm/sna/include -L**
**/opt/ibm/sna/lib -lcpic -lappc -lsna_r -lpthread**

64 ビット・アプリケーションをコンパイルおよびリンクするには、次のオプションを使用します。

**-I /opt/ibm/sna/include -L**
**/opt/ibm/sna/lib64 -lcpic -lappc -lsna_r -lpthread**

# Java CPI-C に関する考慮事項

この節では、Java CPI-C アプリケーションを作成する場合に考慮に入れなければならない事項について簡単に説明します。

## Java CPI-C クラスの使用

Java CPI-C パッケージの名前は **COM.ibm.eNetwork.cpic** です。このパッケージは、以下の項目を含む Java クラスで構成されています。

- サポートされている各 CPI-C コールのメソッド
- これらのコールに渡すパラメーターとして使用するためのクラス

CPIC クラスを使用する Java プログラムを作成する場合は、Java のソースで次の import ステートメントを使用して CPIC パッケージをインポートしてください。

**import COM.ibm.eNetwork.cpic.*;**

### 定数値

Java CPI-C クラスは、特定の CPI-C パラメーターの最大バイト長に対して多くの定数値を定義しています。これらの定数を表 11 に示します。プログラムでは長さを明示的に指定するのではなく、これらの定数を使用してください。

*表 11. Java CPI-C の定数*

| パラメーターの長さ | Java CPI-C の定数 |
|---|---|
| 会話 ID の長さ | CM_CID_SIZE |
| コンテキスト ID の長さ | CM_CTX_SIZE |
| ログ・データ・サイズ | CM_LD_SIZE |
| モード名の長さ | CM_MN_SIZE |
| パートナー LU 名の長さ | CM_PLN_SIZE |
| セキュリティー・パスワードの長さ | CM_PW_SIZE |
| セキュリティー・ユーザー ID の長さ | CM_UID_SIZE |
| シンボリック宛先名の長さ | CM_SDN_SIZE |
| トランザクション・プログラム (TP) 名の長さ | CM_TPN_SIZE |

### パラメーターのタイプ・クラス

CPI-C 関数で使用される多くのパラメーターは、2 つ以上の定義値のセットの 1 つを取ります。Java CPI-C パッケージでは、これらの各パラメーター・タイプが、有効な値を含むクラスとして定義されています。たとえば、CPICSyncLevel クラスは、関数 Set_Sync_Level (cmssl) および Extract_Sync_Level (cmesl) で使用され、CM_NONE または CM_CONFIRM のいずれかの値を取ることができます。

53 ページの『第 3 章 CPI-C コール』の各 CPI-C 関数の説明では、適切な CPI-C パラメーターのクラス・タイプと有効な値をリストしています。たとえば、Set_Sync_Level (cmssl) では、*sync_level* パラメーターは CPICSyncLevel タイプとしてリストされており、この関数のパラメーターの説明では、CM_NONE または CM_CONFIRM を有効な値としてリストしています。

Java クラスに関連付けられる定数値はクラス内に定義されるため、定数値にアクセスするには、特定の値だけでなくクラスも参照する必要があります。たとえば、確認による同期を指定しない場合は、Set_Sync_Level 関数の *sync_level* パラメーターを `CPICSyncLevel.CM_NONE` に設定します。

これらのクラスはそれぞれ、コンストラクターの他に次のメソッドを持っています。

**int intValue()**
: オブジェクトに保管されている値を戻します。

**int intValue(***int_value***)**
: オブジェクトに保管されている値を、指定された整数値 *int_value* に設定し、同じ値を戻します。

  また、オブジェクトの作成時に、そのオブジェクトのコンストラクターへのパラメーターとしてこの値を渡すと、そのオブジェクトに保管される値を設定することもできます。

**boolean equals(***int_value***)**
: オブジェクトに保管されている値が、指定された値 *int_value* と等しければ `true` を戻します。

**boolean equals(***supplied_object***)**
: オブジェクトに保管されている値が、指定パラメーター *supplied_object* に保管されている値に等しければ、`true` を戻します。*supplied_object* は、それ自体が Java CPI-C パラメーター・クラスのうちの 1 つのインスタンスである必要があります。

クラス `CPICReturnCode` は次の追加メソッドを持っています。

**boolean isOK()**
: `CPICReturnCode` オブジェクトに保管されている値が `CM_OK` かどうかを判別するには、アプリケーションはこのメソッドを呼び出す必要があります。保管されている値が `CM_OK` でない場合は、このクラスは例外を生成します。

## 使用例

以下に示した例は、Java プログラムをセットアップして Java CPI-C クラスを使用する方法と、個々の CPI-C コールを行う方法です。

Java CPI-C パッケージをインポートするには、プログラムのソース・コードの先頭に以下を組み込みます。

```
import COM.ibm.eNetwork.cpic.*;
```

プログラムで Java CPI-C を使用するには、次のように Java CPI-C クラスのインスタンスを作成します。

```
CPIC cpicObject = new CPIC();
```

次のステップは、Initialize_Conversation (cminit) 関数を例として、各 Java CPI-C 関数を呼び出す方法を示したものです。

1. 次のように関数のパラメーターを作成し、初期化する。

   ```
   byte[] bConversationId = new byte[cpicObject.CM_CID_SIZE];
   String sSymbolicDestination = "testprog";
   CPICReturnCode cpicReturn = new CPICReturnCode(0);
   ```

   会話 ID 用のバイト配列のサイズを設定するための定数 CM_CID_SIZE の使用法と、このパラメーターの初期値をゼロに設定するための CPICReturnCode クラスの使用法に注意してください。この例の最後の行は、次のように 2 行に分割することもできます。

   ```
   CPICReturnCode cpicReturn = new CPICReturnCode();
   cpicReturn.intValue(0);
   ```

2. 次のように関数コールを発行する。

   ```
   cpicObject.cminit(bConversationId,
                     sSymbolicDestination,
                     cpicReturn);
   ```

3. 次のように戻りコードを特定の値と照合してテストする。

   ```
   if (cpicReturn.intValue() != CPICReturnCode.CM_PARAMETER_ERROR)
   . . .
   ```

   あるいは、次のように戻りコードが CM_OK かどうかを検査する。

   ```
   try
   {
         cpicReturn.isOK();
   }
   catch(CPICReturncode c)
   {
   . . . // cpicReturn is not set to CM_OK
   }
   ```

## Java CPI-C アプリケーションのコンパイルとリンク

Java CPI-C アプリケーションをコンパイルしてリンクする前に、Java クラスを保管しているディレクトリーを指定します。そのためには、環境変数 CLASSPATH を **/usr/lib/sna/java/cpic.jar:.** (AIX) または **/opt/ibm/sna/java/cpic.jar:.** (Linux) に設定し、エクスポートします。

Java コンパイラー **javac** を通常の方法で使用して、アプリケーションをコンパイルしてリンクします。

## Java CPI-C アプリケーションの実行

アプリケーションを実行する前に、ライブラリーが保管されているディレクトリーを指定します。これによりアプリケーションが実行時にライブラリーを見つけ出すことができます。

そのためには、該当する環境変数を次のように設定し、エクスポートします。

**export LD_LIBRARY_PATH=/usr/lib/sna**

また、 37 ページの『ローカル TP 名の指定』で説明しているように、APPCTPN 環境変数を設定しエクスポートして、アプリケーションのローカル TP 名を指定する必要があります。

Java インタープリター **java** を通常の方法で使用して、アプリケーションを実行します。

# Windows に関する考慮事項

WINDOWS

このセクションでは、Windows の Remote API Client 上でプログラムを開発する際に、認識する必要のある考慮事項の処理を要約しています。

## Windows CPI-C ファイル

Windows CPI-C アプリケーションで使用するヘッダー・ファイルは、 **wincpic.h** です。このファイルには、すべての CPI-C エントリー・ポイントの定義と、Windows CPI-C インターフェースでの指定および戻りパラメーターの値に対する定義済みの定数が含まれます。このファイルは、Windows ソフトウェア上の Remote API Client をインストールしたディレクトリー内のサブディレクトリー **/sdk** にインストールされます。

Windows CPI-C アプリケーションのリンクに使用されるライブラリーは、**wcpic32.lib** です。

## 関数のプロトタイプ

53 ページの『第 3 章 CPI-C コール』に記載されている CPI-C コールの関数のプロトタイプは、 AIX または Linux システムで使用される形式になっています。Windows システム用には、各コールの void *functionname* を void WINAPI *functionname* に置き換えてください。

## 複数プロセスと複数会話

複数のプロセスが同一の会話 ID を持つことはできません。 Initialize_Conversation または Accept_Conversation コールを発行するプロセスだけが、コールにより戻される会話 ID を使用することができます。CPI-C を使用する必要のある別のプロセスは、 Initialize_Conversation または Accept_Conversationcall を発行して、固有の会話 ID を取得しなければなりません。

1 つのプログラムは、最大 64 の同時会話を処理することができます。

## Windows 関数コール

標準の CPI-C 関数コールと Windows 固有の CPI-C 関数コール Specify_Windows_Handle に加えて、 Windows アプリケーションは、次の関数を使用することもできます。

**WinCPICStartup**
このコールは、アプリケーションを Windows CPI-C ユーザーとして登録し、 CPI-C ソフトウェアがそのアプリケーションに必要な関数のレベルをサポートするかどうかを判別します。

**WinCPICCleanup**
CPI-C の使用を完了したときに、アプリケーションを未登録にします。

**WinCPICIsBlocking**
このアプリケーションに未解決のブロッキング・コールがあるかどうかを確認します。このコールが必要になる環境についての詳細は、 48 ページの『ブロッキング・コール』を参照してください。

**WinCPICSetBlockingHook**
CPI-C がブロッキング・コールを処理中に使用する、ブロッキング・プロシージャーを指定します。これは、CPI-C のデフォルトのブロッキング・プロシージャーを置き換えます。ブロッキング・プロシージャーは、ブロッキング・コールの処理が完了するまで、繰り返し呼び出されます。詳しくは、48 ページの『ブロッキング・コール』を参照してください。

**WinCPICUnhookBlockingHook**
直前の WinCPICSetBlockingHook コールにより指定されたブロッキング・プロシージャーを未登録にし、CPI-C がデフォルトのブロッキング・プロシージャーを使用するように戻します。

**WinCPICExtractEvent**
アプリケーションが CPI-C 会話に使用される Win32 イベント・ハンドルを決定する方法を提供します。

**WinCPICSetEvent**
Win32 イベント・ハンドルを CPI-C 会話の verb 完了に関連付けます。

アプリケーションは、CPI-C コールの発行を試みる前に、WinCPICStartup を呼び出す必要があります。

『ブロッキング・コール』には、Windows 環境においてブロッキング・コールがどのように動作するかということと、またアプリケーションが WinCPICIsBlocking、WinCPICSetBlockingHook、および WinCPICUnhookBlockingHook コールをどのように使用しなければならないかということについて、詳細な情報が記載されています。

アプリケーションが CPI-C コールの発行を完了したとき、終了する前に WinCPICCleanup を呼び出す必要があります。 WinCPICCleanup を呼び出した後は、CPI-C コールを発行することはできません。

Windows 関数コールについては、53 ページの『第 3 章 CPI-C コール』 の末尾に記載されています。

## ブロッキング・コール

このセクションでは、ブロッキング CPI-C コール (CM_BLOCKING に設定された会話の処理モードで発行されたコール) が、呼び出しアプリケーションが単一スレッドの場合に Win32 環境でどのように動作するかを説明します。 (通常、 Win32 アプリケーションは、マルチスレッドを使用して、ブロッキング verb のプログラムがアプリケーション全体をブロックするのを回避します。)

このセクションでは、ブロッキング・コールを使用するアプリケーションを作成する際に認識する必要のある情報も記載します。

Remote API Client は、Windows システム上のブロッキング・コールをサポートして、他のオペレーティング・システム環境からのアプリケーションの移行を支援します。ただし、Windows 環境におけるブロッキング・コールの使用は、おやめください。 Windows に固有な新規のアプリケーションを作成している場合、以下を行う必要があります。

- Specify_Windows_Handle 関数を使用して、CPI-C がコール完了の結果を通知する、 Windows ハンドルを指定する
- すべての CPI-C コールを非ブロッキング・モードで発行する

ブロッキング・コールは、CPI-C がコールの処理を完了するまでアプリケーションを中断するように見えますが、 CPI-C ライブラリーは、CS/AIX が処理を完了するのを待つ間、他のプロセスが稼働できるように、システムの制御を放棄する必要があります。そのために、「ブロッキング関数」を使用し、ライブラリーが待機中にこの関数が繰り返し呼び出されます。この関数により、Windows メッセージを他のプロセスに送信することができます。この関数についての詳細は、 49 ページの『デフォルトのブロッキング関数』を参照してください。

ブロッキング関数は、元のブロッキング・コールを発行したアプリケーションにメッセージを送信することができます。この場合、アプリケーションは、未解決のブロッキング・コールがあっても再入することができます。これらの環境では、アプリケーションは CPI-C コールの発行に関連しない他の処理を継続することができます。ただし、最初のコールが未解決である間は、他のブロッキング・コールを発行することはできません。

アプリケーションは、 53 ページの『第 3 章 CPI-C コール』に記載されている WinCPICIsBlocking 関数を使用して、ブロッキング・コールが未解決であるか (すなわち、コールが未解決である間に受信したメッセージの結果として再入されたかどうか) を確認することができます。この関数が、ブロッキング・コールが未解決であることを示す場合、アプリケーションは、さらにブロッキング CPI-C コールを発行すべきではありません。ただし、以下を行うことができます。

- 他の処理を継続する
- 処理モードが CM_NON_BLOCKING の他の会話に、CPI-C コールを発行する

### デフォルトのブロッキング関数

Windows CPI-C ライブラリーで使用される標準のブロッキング関数は、次の通りです。

```
BOOL DefaultBlockingHook (void) {
       MSG msg;
       /* get the next message if any */
       if ( PeekMessage (&msg,0,0,PM_NOREMOVE) ) {
             if ( msg.message == WM_QUIT )
                    return FALSE; // let app process WM_QUIT
       PeekMessage (&msg,0,0,PM_REMOVE);
       TranslateMessage (&msg);
       DispatchMessage (&msg);
}
/* TRUE if no WM_QUIT received */
return TRUE;
}
```

アプリケーションが、ブロッキング関数の一部として実行される他の処理を含む必要がある場合、固有のブロッキング関数を指定して、CPI-C が提供するデフォルトのものと置き換えることができます。これを行うためには、 53 ページの『第 3 章 CPI-C コール』に記載されている WinCPICSetBlockingHook コールを使用します。

ブロッキング関数は、WM_QUIT メッセージを受け取ると、FALSE を戻す必要があります。これは、CPI-C はアプリケーションに制御を戻し、アプリケーションはメッセージを処理して終了できるということです。そうでない場合、関数はTRUE を戻す必要があります。

## アプリケーションの終了

CPI-C は、Windows でアプリケーションが終了するタイミングを識別できません。このため、アプリケーションをクローズする必要のある場合 (例えば、WM_CLOSE メッセージを受け取った場合)、アプリケーションは、WinCPICCleanup コールを発行する必要があります。この関数を発行しないと、システムは不確定な状態になります。ただし、CPI-C がアプリケーションが終了したことを検出すると、可能な限りのクリーンアップが行われます。

## CPI-C アプリケーションのコンパイルとリンク

このセクションでは、Windows システムの CPI-C アプリケーションのコンパイルとリンクについての情報を記載します。

### 構造パッキング用コンパイラー・オプション

一部の CPI-C コールで指定され、戻される構造はパックされていません。このパッキング・メソッドを変更するコンパイラー・オプションを使用しないでください。

BYTE パラメーターは BYTE 境界に、WORD パラメーターは WORD 境界に、DWORD パラメーターは DWORD 境界にあります。

### ヘッダー・ファイル

Windows CPI-C アプリケーションに組み込まれるヘッダー・ファイルは、 指定された **wincpic.h** です。このファイルは、Windows クライアント・ソフトウェアをインストールしたディレクトリー内のサブディレクトリー **/sdk** にインストールされます。

### ロード時リンク

ロード時にアプリケーションを CPI-C にリンクするには、アプリケーションをライブラリー **wincpic32.lib** にリンクします。

### 実行時リンク

実行時にアプリケーションを CPI-C にリンクするには、アプリケーションに以下のコールを組み込みます。

- CPI-C ダイナミック・リンク・ライブラリー **wincpic32.dll** をロードする `LoadLibrary`
- `WinCPIC` をダイナミック・リンク・ライブラリーへのエントリー・ポイントとして指定する `GetProcAddress`
- ライブラリーが必要なくなった場合に `FreeLibrary`

## 移植可能なアプリケーションの作成

以下に示したガイドラインは、他のオペレーティング・システム環境あるいは他の CPI-C インプリメンテーションに移植可能な CPI-C アプリケーションを作成するためのものです。

- パス名に接頭部を付けずに CPI-C ヘッダー・ファイルを組み込みます。ファイルを見つけるには、コンパイラーで組み込みオプションを使用します (この章の前の方で述べている、ご使用のオペレーティング・システムの該当するセクションを参照してください)。これにより、アプリケーションを異なるファイル・システムを持つ環境で使用することができます。
- パラメーター値と戻りコードには、ヘッダー・ファイルに示されている数値ではなく、記号定数名を使用します。これにより、値のメモリー内での保管方法には関係なく、確実に正しい値を使用することができます。
- 現在ご使用のオペレーティング・システムに適用できるもの以外の戻りコードの検査を組み込み (例えば、switch ステートメントで「デフォルトの」ケースを使用する)、適切な診断を行います。
- CS/AIX が提供する CPI-C 関数の中には、X/Open CPI-C との互換性のために組み込まれた拡張関数であったり、CPI-C の標準仕様には含まれていなかったりするものがあり、それらは他のインプリメンテーションでは使用できない場合があります。これらの拡張関数はそれぞれ、53 ページの『第 3 章 CPI-C コール』の関数説明の概要の部分で注記されています。

– X/Open 関数が組み込まれているのは、CS/AIX で X/Open CPI-C 用に作成された既存のアプリケーションを使用できるようにするためです。新しいアプリケーションを作成する場合は、これらの関数は使用しないでください。
– アプリケーションで拡張関数を使用する場合は、他の環境でも使用できるように、アプリケーションのいくつかのセクションを作成し直す必要がある場合もあります。これらの関数を数個の特定のルーチンに限定して使用すれば、変更が簡単になります。

AIX、LINUX

以下に示したガイドラインは、Java CPI-C アプリケーションに適用されます。

- 3 つの関数 Extract_Conversation_Context、Set_Conversation_Context、および Set_Local_LU_Name は、CPI-C の標準仕様には含まれておらず、 IBM の Java CPI-C for CS/Windows ではサポートされません。Java CPI-C アプリケーションでこれらの関数を使用する場合は、他の Java CPI-C 環境でも使用できるように、アプリケーションのいくつかのセクションを作成し直す必要がある場合もあります。これらの関数を数個の特定のルーチンに限定して使用すれば、変更が簡単になります。
- Java CPI-C クラスにはこの資料で説明していない CPI-C 関数がいくつか組み込まれています。これらの関数は Java クラスの一部として定義されていますが、サポートはされていません。 アプリケーションでこれらの非サポート関数を使用する場合、コンパイルは正常に行われる場合もありますが、アプリケーションがこれらの関数を使用すると、これらの関数はエラーを示す戻りコード (CM_CALL_NOT_SUPPORTED) を戻します。

# 第 3 章 CPI-C コール

この章では、CPI-C 関数コールおよび CPI-C アプリケーションで使用される追加の Windows 固有の関数について説明します。次の情報が記載されています。

- コールに対して提供される情報の説明
- コールの説明

## CPI-C コールに対して提供される情報

この章で説明するそれぞれの CPI-C コールについては、次の情報が用意されています。

- コールの呼称と、括弧で囲んだ実際の C 関数名 (この情報が各節の見出しになっています)。
- コールの関数プロトタイプ。これには、コールが使用するパラメーターとそれぞれのパラメーターのデータ・タイプが含まれています。各関数のプロトタイプ宣言は、ファイル **cmc.h** (AIX または Linux システム) または **wincpic.h** (Windows システム) にあります。

  WINDOWS

  『第 3 章 CPI-C コール』に記載されている CPI-C コールの関数のプロトタイプは、 AIX または Linux システムで使用される形式になっています。Windows システム用には、各コールの void *functionname* を void WINAPI *functionname* に置き換えてください。

  [redacted]

- CPI-C 関数の Java メソッド定義 (その関数が Java CPI-C でサポートされている場合)。
- 個々の指定パラメーターと戻りパラメーターの説明。パラメーター名は便宜上の呼称です。これらのパラメーターの実際の変数名は、アプリケーション・プログラムで宣言します。説明は、パラメーターとして使用できる値も含んでいます。
- コールを発行できる会話状態。
- コールからの戻り時に会話状態が変わる場合に、変化後の会話状態。パラメーター・チェックや状態チェックなど、対話の変化の原因にならない条件は示されていません。
- コールの使用法についての追加説明。

### データ・タイプ

Java CPI-C アプリケーションにおけるデータ・タイプについては、 43 ページの『Java CPI-C に関する考慮事項』を参照してください。

CPI-C アプリケーションの移植性向上のために、CPI-C に対して指定するパラメーターおよび CPI-C から戻されるパラメーターのデータ・タイプは、CPI-C ヘッダー・ファイル内の #define ステートメントにより記号定数として設定されています。たとえば、CM_INT32 は 32 ビットの整数タイプを表し、CM_PTR はポインター・タイプを表します。

この章では、指定パラメーターおよび戻りパラメーターのデータ・タイプを識別するために、これらの記号定数を使用しています。アプリケーションを作成するときは、実データ・タイプでなくこれらの記号定数を使用することをお勧めします。

## データ構造

Java CPI-C でサポートされている CPI-C 関数は、いずれもデータ構造を使用しないので、この節は Java CPI-C アプリケーションには適用されません。

いくつかの CPI-C コールでは、アプリケーションがデータ構造を提供し、そこに CS/AIX がアプリケーションに戻すパラメーターを保管します。これらのデータ構造には、「予約済み」としてマークされたパラメーターが含まれることがあります。これらの予約済みパラメーターには、CS/AIX ソフトウェアにより内部的に使用されるものがありますが、その他にも、このバージョンでは使用されなくても将来のバージョンで使用されるものもあります。アプリケーションでは、これらの予約済みパラメーターに決してアクセスしないでください。verb によって使用される他のパラメーターをアプリケーションが設定する前に、データ構造の内容全体をゼロに設定して、これらのパラメーターすべてを確実にゼロに設定しておく必要があります。このようにすると、CS/AIX がその内部使用パラメーターを誤って解釈することはありません。またこれにより、今後の CS/AIX のバージョンで、これらのパラメーターを使って新しい機能を引き続き使用することができるようになります。

データ構造の内容をゼロに設定するには、次のように `memset` を使用します。

```
memset(my_struct, 0, sizeof(my_struct));
```

## 記号定数

Java CPI-C アプリケーションにおける記号定数値については、43 ページの『Java CPI-C に関する考慮事項』を参照してください。

CPI-C に対する指定パラメーターと CPI-C からの戻りパラメーターのほとんどは、32 ビットの整数です。コーディングを単純化するために、これらのパラメーターの値は、ヘッダー・ファイルの #define ステートメントにより設定される、意味の分かりやすい記号定数で表してあります。たとえば、値 `CM_MAPPED_CONVERSATION` は整数 1 を表します。移植性と可読性を確保するために、プログラムを書くときは記号定数だけを使用してください。

## ストリング

CPI-C インターフェースを介して渡される場合、ストリングはすべて ASCII 形式です。

### 戻りパラメーターの妥当性

CPI-C から戻されるパラメーターが有効なのは、CPI-C コールが正常に実行されたことが戻りコード CM_OK により示されている場合だけです。

## Windows 関数コールに対して提供される情報

WINDOWS

以下の情報は、この章で説明する Windows 固有の関数コールがそれぞれ述べられています。

- コールの名前。CPI-C 関数コールと異なり、これらのコールにはスードニム (pseudonym) はありません。
- コールの説明。
- コールの関数プロトタイプ。これには、コールが使用するパラメーターとそれぞれのパラメーターのデータ・タイプが含まれています。各関数のプロトタイプ宣言は、ファイル **wincpic.h** にあります。
- 個々の指定パラメーターと戻りパラメーターの説明。パラメーター名は便宜上の呼称です。これらのパラメーターの実際の変数名は、アプリケーション・プログラムで宣言します。説明は、パラメーターとして使用できる値も含んでいます。
- コールの使用法についての追加説明。

## Accept_Conversation (cmaccp)

Accept_Conversation コールは、着呼会話を受け入れ、特定の会話特性を設定するために、呼び出し対象プログラムによって発行されます。初期会話特性のリストについては、21 ページの『第 2 章 CPI-C アプリケーションの作成』を参照してください。

このコールが正常に実行されると、CPI-C は 8 バイトの会話 ID を生成します。この ID は、この会話の間に呼び出し対象プログラムが発行する他のすべての CPI-C コールの必須パラメーターです。

### 関数コール

```
void cmaccp (
             unsigned char CM_PTR      conversation_ID,
             CM_RETURN_CODE CM_PTR     return_code
            );
```

### Java CPI-C の関数コール

AIX、LINUX

```
public native void cmaccp (
                            byte[]          conversation_ID,
                            CPICReturnCode  return_code
                           );
```

## 指定パラメーター

このコールには、指定パラメーターはありません。

## 戻りパラメーター

コールの実行後に、CS/AIX は、処理が正常に実行されたかどうかを示すパラメーターと、失敗した場合はその理由を示すパラメーターを戻します。

*conversation_ID*
このパラメーターは会話の ID です。この ID は、後続の CPI-C コールにより使用されます。

*return_code*
値は次のとおりです。

**CM_OK** コールは正常に実行されました。

**CM_PROGRAM_STATE_CHECK**
この値は次の状態のいずれかを示します。

- 構成内容に指定されているタイムアウト期間内に、着呼 Allocate 要求を受信しなかった。
- アプリケーションがローカル TP 名を指定しなかった (または、AIX または Linux システムの場合、すべての指定名を解放した)。このコールを発行する前に、アプリケーションは少なくとも 1 つのローカル TP 名を持っていなければなりません。ローカル TP 名の指定の詳細については、37 ページの『ローカル TP 名の指定』を参照してください。
- アプリケーションが手操作で開始されたが、呼び出し可能な TP データ・ファイルに非待機として定義されている。非待機 TP は、会話要求 (着呼 Attach) に応答して CS/AIX により自動的に開始されます。手操作で開始しようとしても、このアプリケーションを待っている着呼 Attach がないので、Accept_Conversation コールは失敗します。

**CM_PRODUCT_SPECIFIC_ERROR**
191 ページの『付録 B. 共通な戻りコード』を参照してください。

## 発行時の状態

会話はリセット状態になっていなければなりません。

## 状態の変化

コールが成功した場合は、会話が受信状態に変わります。コールが失敗した場合は、会話は元の状態のままです。

## 使用上の注意

TP 名は、さまざまな方法で指定できます。ローカル TP 名の指定の詳細については、37 ページの『ローカル TP 名の指定』を参照してください。プログラムは、Accept_Conversation を発行する前に、着呼 Allocate を受け付ける 1 つ以上の TP 名を指定するための Specify_Local_TP_Name を発行できます (これらの名前は、APPCTPN 環境変数などの他の方法で定義された名前に追加して使用されます)。この方法で複数の TP 名を指定した場合は、Extract_TP_Name コールを (Accept_Conversation から戻ったあとで) 使用して、呼び出し側プログラムがどの TP 名を使用したかを判別できます。

AIX、LINUX

Accept_Conversation から CM_OK が戻された場合は、この会話用の新規の会話コンテキストが作成され、それがプログラムの現行コンテキストになります。

Accept_Conversation は、常にブロッキング・モードで動作します。つまり、このコールは常に、着呼 Allocate 要求を受信するまでは延期されます。次の方法を使用すれば、不必要な遅延を避けることができます。

- このアプリケーションの呼び出し可能な TP 構成に小さいタイムアウト値が指定され、着呼 Allocate 要求がない場合に Accept_Conversation コールが (*return_code* CM_PROGRAM_STATE_CHECK を伴って) 短時間で戻り、あとでアプリケーションが Accept_Conversation を再試行するようになっていることを確認してください。タイムアウト値は、起動可能な TP データ・ファイル内に指定されます。詳しくは、「*Communications Server for AIX 管理ガイド*」を参照してください。
- Accept_Conversation を使用する代わりに、非ブロッキング・モードで動作可能な Accept_Incoming を使用します。次のコール・シーケンスを使用してください。
  - Initialize_For_Incoming (着呼会話の会話 ID を取得する)
  - Set_Processing_Mode (この会話 ID の *processing_mode* を CM_NON_BLOCKING に設定する)
  - Accept_Incoming

  詳細については、各コールの説明を参照してください。

# Accept_Incoming (cmacci)

AIX、LINUX

Accept_Incoming コールは、Initialize_For_Incoming により初期化済みの着呼会話を受け入れ、特定の会話特性を設定するために、呼び出し対象プログラムによって発行されます。初期会話特性のリストについては、30 ページの『初期会話特性』を参照してください。

プログラムは、このコールを発行する前に、Set_Processing_Mode を発行して会話の処理モードを CM_NON_BLOCKING に設定することができます。これによって、Accept_Incoming コール、および後続のすべての CPI-C コールが非ブロッキング・モードで発行されます。

## 関数コール

```
void cmacci (
             unsigned char CM_PTR      conversation_ID,
             CM_RETURN_CODE CM_PTR     return_code
            );
```

## Java CPI-C の関数コール

```
public native void cmacci (
                           byte[]          conversation_ID,
                           CPICReturnCode  return_code
                          );
```

## 指定パラメーター

指定パラメーターは次のとおりです。

*conversation_ID*

このパラメーターは、直前の Initialize_For_Incoming コールで戻された会話の ID です。この ID は、この会話の後続の CPI-C コールを識別するために使用されます。

## 戻りパラメーター

verb の実行後に、CS/AIX は、処理が正常に実行されたかどうかを示すパラメーターと、失敗した場合はその理由を示すパラメーターを戻します。

*return_code*

値は次のとおりです。

**CM_OK** コールは正常に実行されました。

**CM_PROGRAM_PARAMETER_CHECK**

*conversation_ID* に指定された値は無効です。

**CM_PROGRAM_STATE_CHECK**

次のいずれかの状態が発生しました。

- *conversation_ID* に示された会話が初期化 - 着呼状態でない。
- 構成内容に指定されているタイムアウト期間内に、着呼 Allocate 要求を受信しなかった。
- アプリケーションが (たとえば APPCTPN 環境変数に) 指定されているローカル TP 名を解除し、追加のローカル TP 名を何も指定していない。このコールを発行する前に、アプリケーションは少なくとも 1 つのローカル TP 名を持っていなければなりません。ローカル TP 名の指定の詳細については、37 ページの『ローカル TP 名の指定』を参照してください。
- アプリケーションが手操作で開始されたが、呼び出し可能な TP データ・ファイルに非待機として定義されている。非待機 TP は、着呼 Attach に応答して CS/AIX により自動的に開始されま

す。手操作で開始しようとしても、アプリケーションを待っている着呼 Attach がないので、Accept_Incoming コールは失敗します。

次の戻りコードの説明については、191 ページの『付録 B. 共通な戻りコード』を参照してください。

CM_OPERATION_INCOMPLETE
CM_OPERATION_NOT_ACCEPTED
CM_PRODUCT_SPECIFIC_ERROR

## 発行時の状態

会話は初期化 - 着呼状態になっていなければなりません。

## 状態の変化

コールが成功した場合は、会話が受信状態に変わります。コールが失敗した場合は、会話は元の状態のままです。

## 使用上の注意

Initialize_For_Incoming に続けて Accept_Incoming を発行するのは、Accept_Conversation を発行するのと同じです。会話を受け付けるこれらの 2 つの方法の違いは、Accept_Conversation が常にブロッキング・モードで動作するのに対して、Accept_Incoming は非ブロッキング・モードでも動作できる点にあります。非ブロッキング・モードで会話を受け付けるには、プログラムから次のコール・シーケンスを発行します。

Initialize_For_Incoming (着呼会話の会話 ID を取得する)
Set_Processing_Mode (この会話 ID の *processing_mode* を CM_NON_BLOCKING に設定する)
Accept_Incoming

APPCTPN 環境変数で指定される TP 名は、通常、着呼 Allocate をこのプログラムと突き合わせるために使用されます。プログラムは、Accept_Incoming を発行する前に、着呼 Allocate を受け入れる 1 つ以上の TP 名を示すための Specify_Local_TP_Name を発行できます (これらの名前は、APPCTPN 内の名前と置き換わります)。この方法で複数の TP 名を指定した場合は、Extract_TP_Name コールを (Accept_Incoming から戻ったあとで) 使用して、呼び出し側プログラムがどの TP 名を使用したかを判別できます。ローカル TP 名の指定の詳細については、37 ページの『ローカル TP 名の指定』を参照してください。

Accept_Incoming から CM_OK が戻された場合は、この会話用の新規の会話コンテキストが作成され、それがプログラムの現行コンテキストになります。
Accept_Incoming から CM_OPERATION_INCOMPLETE が戻され、後続の Wait_For_Conversation から Accept_Incoming の完了を示す CM_OK が戻された場合は、会話用の新規の会話コンテキストは作成されますが、プログラムの現行コンテキストは変更されません。新規コンテキストを使用するには、プログラムは、この *conversation_ID* のための Extract_Conversation_Context を発行して会話のコンテキストの値を取得し、Set_Conversation_Context を発行してプログラムの現行コンテキストをこの値に設定する必要があります。

# Allocate (cmallc)

Allocate コールは、現行の会話特性を使用してパートナー・プログラムとの会話を割り振るために、呼び出し側プログラムが発行します。ローカル LU とパートナー LU との間に既存のセッションがない場合は、CPI-C がこのセッションを割り振ることもできます。

割り振られる会話のタイプは、会話タイプの特性 (マップ式または基本) に基づいて決まります。

このコールで会話が割り振られたあとでは、次の会話特性は変更できません。

- 会話タイプ
- モード名
- パートナー LU 名
- パートナー・プログラム名
- 戻り制御
- 同期レベル
- 会話セキュリティー
- ユーザー ID
- パスワード

## 関数コール

```
void cmallc (
             unsigned char CM_PTR      conversation_ID,
             CM_RETURN_CODE CM_PTR     return_code
            );
```

## Java CPI-C の関数コール

AIX、LINUX

```
public native void cmallc (
                           byte[]          conversation_ID,
                           CPICReturnCode  return_code
                          );
```

## 指定パラメーター

指定パラメーターは次のとおりです。

*conversation_ID*
このパラメーターは会話 ID です。このパラメーターの値は、Initialize_Conversation コールから戻されます。

## 戻りパラメーター

verb の実行後に、CS/AIX は、処理が正常に実行されたかどうかを示すパラメーターと、失敗した場合はその理由を示すパラメーターを戻します。

*return_code*
値は次のとおりです。

**CM_OK** コールは正常に実行されました。

**CM_PARAMETER_ERROR**
次のいずれかの状態が発生しました。

- サイド情報から取り出されたモード名、または Set_Mode_Name により設定されたモード名が有効でない。
- モード名は、SNA の内部使用のために予約された名前の 1 つ (たとえば SNASVCMG) であり、アプリケーションがこの名前を使用できない。

**CM_PROGRAM_PARAMETER_CHECK**
*conversation_ID* に指定された値は無効です。

**CM_PROGRAM_STATE_CHECK**
会話は初期化状態ではありません。

**CM_UNSUCCESSFUL**
会話の戻り制御特性が CM_IMMEDIATE に設定されており、ローカル LU には使用可能なコンテンション勝者セッションがありません。

次の戻りコードの説明については、191 ページの『付録 B. 共通な戻りコード』を参照してください。

CM_ALLOCATE_FAILURE_NO_RETRY
CM_ALLOCATE_FAILURE_RETRY
CM_OPERATION_INCOMPLETE
CM_OPERATION_NOT_ACCEPTED
CM_PRODUCT_SPECIFIC_ERROR

## 発行時の状態

会話は初期化状態になっていなければなりません。

## 状態の変化

表 12 に要約して示した状態の変化は、*return_code* パラメーターの値に基づくものです。

*表 12. Allocate コールによる状態の変化*

| *return_code* | **新しい状態** |
|---|---|
| CM_OK | 送信 |
| CM_ALLOCATE_FAILURE_NO_RETRY | リセット |
| CM_ALLOCATE_FAILURE_RETRY | |
| 上記以外 | 変化なし |

### 使用上の注意

割り振り要求を即時に送信する場合は、呼び出し側プログラムは Allocate コールの直後に Flush コール、または Confirm コールを発行できます。それ以外の場合は、割り振り要求は、バッファーが満ぱいになるまで、ローカル LU の送信バッファーに他のデータと共に累積されます。

割り振り要求はバッファーに入れられ、すぐには送信されないため、Allocate コールからは CM_OK が戻されても、その後パートナー LU が Allocate コールにより生成された割り振り要求をリジェクトする可能性があります。このエラーは、後続のコールのときに呼び出し側プログラムに戻されます。

会話の同期レベルが CM_CONFIRM に設定されていれば、呼び出し側プログラムは、Allocate コールのあとで Confirm コールを発行することによって、割り振りが成功したかどうかを即時に判別できます。

AIX、LINUX

Allocate コールを発行した時点でのプログラムの現行コンテキストが、Allocate から CM_OK が戻されたときの新規会話のコンテキストになります。プログラムが複数のコンテキストを (複数の会話を受け入れた結果として) 使用している場合は、Allocate コールを発行する前に、プログラムで現行コンテキストを適切な値に設定する必要があります。

## Cancel_Conversation (cmcanc)

Cancel_Conversation コールは、指定の会話を終了し、この会話についての不完全操作 (CM_OPERATION_INCOMPLETE を伴って戻った直前のコール) をすべて取り消し、会話が使用していたセッションを終了します。これは、次の相違点を除くと、Deallocate コールの *deallocate_type* パラメーターを CM_DEALLOCATE_ABEND に設定して発行した場合と同じです。

- Deallocate は、操作が不完全な間は使用できません。Cancel_Conversation はこの場合も使用でき、未解決のコールを取り消します。
- Deallocate は、ログ・データがあれば、それをローカル・エラー・ログに書き込みます。Cancel_Conversation にはこの機能はありません。

未解決のコールの結果は未定義であり、アプリケーションには戻されません。たとえば、未解決の Send_Data コールを取り消すために Cancel_Conversation を使用した場合、データは一部しか送信されていないこともあり、全部送信されていることもあります。Send_Error を取り消すために Cancel_Conversation を使用した場合は、エラー通知がパートナー・プログラムに送信されている場合も、送信されていない場合もあります。

Java CPI-C では非ブロッキング・コールがサポートされていないので、不完全コールが未解決になることはあり得ません。Cancel_Conversation は、ログ・データをローカル・エラー・ログに書き込まない点以外は Deallocate と同じです。

## 関数コール

```
void cmcanc (
             unsigned char CM_PTR      conversation_ID,
             CM_RETURN_CODE CM_PTR     return_code
            );
```

## Java CPI-C の関数コール

AIX、LINUX

```
public native void cmcanc (
                           byte[]          conversation_ID,
                           CPICReturnCode  return_code
                          );
```

## 指定パラメーター

指定パラメーターは次のとおりです。

*conversation_ID*
このパラメーターは会話の ID です。このパラメーターの値は、Initialize_Conversation コール、Initialize_For_Incoming コール、または Accept_Conversation コールから戻されます。

## 戻りパラメーター

verb の実行後に、CS/AIX は、処理が正常に実行されたかどうかを示すパラメーターと、失敗した場合はその理由を示すパラメーターを戻します。

*return_code*
値は次のとおりです。

**CM_OK** コールは正常に実行されました。指定の会話は割り振りを解除され、この会話の未解決のコールはすべて取り消されました。

**CM_PROGRAM_PARAMETER_CHECK**
*conversation_ID* に指定された値は無効です。

**CM_PRODUCT_SPECIFIC_ERROR**
191 ページの『付録 B. 共通な戻りコード』を参照してください。

## 発行時の状態

会話は、リセット以外であればどの状態にあっても構いません。

## 状態の変化

戻りコードが CM_OK の場合は、会話状態はリセットに変化します。

## 使用上の注意

戻りコード CM_DEALLOCATED_ABEND によって、会話の終了がパートナー・プログラムに通知されます。

# Check_For_Completion (cmchck)

AIX、LINUX

この関数は Java CPI-C では使用できません。

Check_For_Completion コールは、CM_OPERATION_INCOMPLETE コードで戻った直前のコールが、その後完了したかどうかを検査します。直前のコールが完了しているかどうかにかかわらず、このコールはすぐに戻ります。したがって、アプリケーションは、直前のコールがまだ完了していなければ他の処理を続けることができ、完了していれば、Wait_For_Conversation を呼び出して直前のコールの結果を取得できます。

アプリケーションが複数の会話に関係している場合は、このコールはそのすべての会話に対して働き、そのうちのいずれかの会話で直前のコールが完了すると、「成功」の戻りコードを戻します。

このコールは標準 CPI-C 仕様の一部ではありません。他のインプリメンテーションでは使用できない場合があります。未解決のコールの結果を取得するための標準的な方法は、Wait_For_Conversation を発行することです。このコールはブロッキング・モードで動作し、他のコールが完了するまで待機します。

## 関数コール

```
void cmchck (
             unsigned char CM_PTR      conversation_ID,
             CM_RETURN_CODE CM_PTR     return_code
            );
```

## 指定パラメーター

このコールには、指定パラメーターはありません。

## 戻りパラメーター

verb の実行後に、CS/AIX は、処理が正常に実行されたかどうかを示すパラメーターと、失敗した場合はその理由を示すパラメーターを戻します。

*conversation_ID*

直前の未解決のコールが完了した会話の ID。詳しくは、65 ページの『使用上の注意』を参照してください。

この値が関係するのは、*return_code* パラメーターが CM_OK に設定されている場合だけです。

*return_code*

値は次のとおりです。

**CM_OK** コールは正常に実行されました。*conversation_ID* により指定された会話で前に未解決になっていたコールが完了しました。

**CM_PROGRAM_STATE_CHECK**

前に未解決になっている不完全コールはありません。アプリケーションは、CM_OPERATION_INCOMPLETE を戻したコールは発行していな

いか、そのようなすべてのコールの結果を取得するための Wait_For_Conversation をすでに発行しています。

**CM_UNSUCCESSFUL**
前に未解決になっている不完全コールが 1 つ以上ありますが、そのどれもまだ完了していません。アプリケーションは、他の処理を続け、あとで Check_For_Completion を再試行する必要があります (この戻りコードは CM_PROGRAM_STATE_CHECK とは別のものです)。

## 発行時の状態

このコールは、特定の会話に関連付けられていません。したがって、会話状態は無関係です。ただしアプリケーションは、未解決の不完全操作を持つ会話を 1 つ以上持っていなければなりません。

## 状態の変化

状態には変化はありません。

## 使用上の注意

Check_For_Completion からの戻りコードが CM_OK の場合は、アプリケーションは Wait_For_Conversation をコールして未解決のコールの結果を取得する必要があります。

アプリケーションが前回 Check_For_Completion または Wait_For_Conversation を発行したあとで完了したコールが複数あった場合は、Check_For_Completion を複数回発行しても、必ずしも追加コールについての情報が戻されるとは限りません。単に、少なくとも 1 つのコールが完了したことが示されるだけです。したがって、後続の Wait_For_Conversation コールはブロックされずに即時に戻ります。各 Wait_For_Conversation コールは 1 つの不完全操作を戻します。複数の不完全操作が (異なる会話に) ある場合は、アプリケーションは、Wait_For_Conversation のあとでさらに別の Check_For_Completion を発行して、ほかのコールが完了したかどうかを調べることができます。

Wait_For_Conversation コールは、必ずしも Check_For_Completion が報告したものと同じコールについての情報を戻すわけではありません。

# Confirm (cmcfm)

Confirm コールは、ローカル LU の送信バッファーの内容と確認要求をパートナー・プログラムに送信し、確認を待ちます。

パートナー・プログラムは、通常、Confirm コールへの応答として Confirmed コールを発行して、データを受信しエラーもなかったことを確認します (エラーを検出した場合は、パートナー・プログラムは Send_Error コールを発行するか、Deallocate コールを使用して会話の割り振りを強制的に解除します)。

プログラムが Confirm コールを発行できるのは、会話の同期レベルが CM_CONFIRM のときだけです。

## 関数コール

```
void cmcfm (
            unsigned char CM_PTR                  conversation_ID,
            CM_Request_to_Send_Received CM_PTR    request_to_send_received,
            CM_RETURN_CODE CM_PTR                 return_code
           );
```

## Java CPI-C の関数コール

AIX、LINUX

```
public native void cmcfm (
                          byte[]                          conversation_ID,
                          CPICControlInformationReceived  request_to_send_received,
                          CPICReturnCode                  return_code
                         );
```

## 指定パラメーター

指定パラメーターは次のとおりです。

*conversation_ID*
このパラメーターは会話の ID です。このパラメーターの値は、Initialize_Conversation コール、Initialize_For_Incoming コール、または Accept_Conversation コールから戻されます。

## 戻りパラメーター

verb の実行後に、CS/AIX は、処理が正常に実行されたかどうかを示すパラメーターと、失敗した場合はその理由を示すパラメーターを戻します。

*request_to_send_received*
このパラメーターは、送信要求受信インディケーターです。値は次のとおりです。

**CM_REQ_TO_SEND_RECEIVED**
パートナー・プログラムが Request_To_Send コールを発行しました。このコールは、ローカル・プログラムに、会話を受信状態に変更するよう要求します。

**CM_REQ_TO_SEND_NOT_RECEIVED**
パートナー・プログラムは Request_To_Send コールを発行していません。

*return_code* パラメーターが次のどちらかに設定されている場合は、この値は無関係です。

- CM_PROGRAM_PARAMETER_CHECK

- CM_PROGRAM_STATE_CHECK

*return_code*
値は次のとおりです。

**CM_OK** コールは正常に実行されました。パートナー・プログラムが Confirmed コールを発行しました。

**CM_PROGRAM_PARAMETER_CHECK**
次のいずれかの状態が発生しました。

- *conversation_ID* に指定された値が有効でない。
- ローカル・プログラムは、同期レベルが CM_NONE の会話の中で Confirm コールを使おうとした。同期レベルは CM_CONFIRM でなければなりません。

**CM_PROGRAM_STATE_CHECK**
次のいずれかの状態が発生しました。

- 会話が送信状態または送信 - 保留状態でない。
- ローカル・プログラムの基本会話が送信状態で、そのローカル・プログラムが論理レコードを送信し終わっていない。

次の戻りコードの説明については、191 ページの『付録 B. 共通な戻りコード』を参照してください。

CM_CONVERSATION_TYPE_MISMATCH
CM_DEALLOCATED_ABEND
CM_DEALLOCATED_ABEND_SVC
CM_DEALLOCATED_ABEND_TIMER
CM_OPERATION_INCOMPLETE
CM_OPERATION_NOT_ACCEPTED
CM_PIP_NOT_SPECIFIED_CORRECTLY
CM_PRODUCT_SPECIFIC_ERROR
CM_PROGRAM_ERROR_PURGING
CM_RESOURCE_FAILURE_NO_RETRY
CM_RESOURCE_FAILURE_RETRY
CM_SECURITY_NOT_VALID
CM_SVC_ERROR_PURGING
CM_SYNC_LVL_NOT_SUPPORTED_PGM
CM_SYNC_LVL_NOT_SUPPORTED_LU
CM_TP_NOT_AVAILABLE_NO_RETRY
CM_TP_NOT_AVAILABLE_RETRY
CM_TPN_NOT_RECOGNIZED

## 発行時の状態

会話は、送信状態でも送信 - 保留状態でも構いません。

## 状態の変化

68 ページの表 13 に要約して示した状態の変化は、*return_code* パラメーターの値に基づくものです。

表 13. *Confirm* コールによる状態の変化

| *return_code* | 新しい状態 |
|---|---|
| CM_OK (送信状態でコールを発行) | 変化なし |
| CM_OK (送信 - 保留状態でコールを発行) | 送信 |
| CM_PROGRAM_ERROR_PURGING | 受信 |
| CM_SVC_ERROR_PURGING | |
| CM_CONVERSATION_TYPE_MISMATCH | リセット |
| CM_PIP_NOT_SPECIFIED_CORRECTLY | |
| CM_SECURITY_NOT_VALID | |
| CM_SYNC_LEVEL_NOT_SUPPORTED_PGM | |
| CM_SYNC_LEVEL_NOT_SUPPORTED_LU | |
| CM_TPN_NOT_RECOGNIZED | |
| CM_TP_NOT_AVAILABLE_NO_RETRY | |
| CM_TP_NOT_AVAILABLE_RETRY | |
| CM_RESOURCE_FAILURE_NO_RETRY | |
| CM_RESOURCE_FAILURE_RETRY | |
| CM_DEALLOCATED_ABEND | |
| CM_DEALLOCATED_ABEND_SVC | |
| CM_DEALLOCATED_ABEND_TIMER | |
| 上記以外 | 変化なし |

## 使用上の注意

Confirm コールは、パートナー・プログラムからの応答を待ちます。応答は、パートナー・プログラム内の次の CPI-C コールの 1 つによって生成されます。

- Confirmed
- Send_Error
- Deallocate (会話の割り振り解除タイプが CM_DEALLOCATE_ABEND に設定されている場合)

# Confirmed (cmcfmd)

Confirmed コールは、パートナー・プログラムからの確認要求への応答を送ります。このコールは、パートナー・プログラムに対して、ローカル・プログラムが受信データ内にエラーを検出しなかったことを通知します。

確認要求を発行したプログラムは確認を待つので、Confirmed コールにより、2 つのプログラムの処理が同期化されます。

## 関数コール

```
void cmcfmd (
             unsigned char CM_PTR      conversation_ID,
             CM_RETURN_CODE CM_PTR     return_code
            );
```

## Java CPI-C の関数コール

AIX、LINUX

```
public native void cmcfmd (
                            byte[]          conversation_ID,
                            CPICReturnCode  return_code
                          );
```

## 指定パラメーター

指定パラメーターは次のとおりです。

*conversation_ID*
このパラメーターは会話の ID です。このパラメーターの値は、Initialize_Conversation コール、Initialize_For_Incoming コール、または Accept_Conversation コールから戻されます。

## 戻りパラメーター

verb の実行後に、CS/AIX は、処理が正常に実行されたかどうかを示すパラメーターと、失敗した場合はその理由を示すパラメーターを戻します。

*return_code*
値は次のとおりです。

**CM_OK** コールは正常に実行されました。

**CM_PROGRAM_PARAMETER_CHECK**
*conversation_ID* に指定された値は無効です。

**CM_PROGRAM_STATE_CHECK**
プログラムがこのコールを発行したとき、会話が確認状態、確認 - 送信状態、または確認 - 割り振り解除状態になっていませんでした。

次の戻りコードの説明については、 191 ページの『付録 B. 共通な戻りコード』を参照してください。

CM_OPERATION_INCOMPLETE
CM_OPERATION_NOT_ACCEPTED
CM_PRODUCT_SPECIFIC_ERROR

## 発行時の状態

プログラムがこのコールを発行するとき、会話は次の状態のいずれかになっていなければなりません。

- 確認
- 確認 - 送信
- 確認 - 割り振り解除

## 状態の変化

新しい状態は元の状態、つまり、ローカル・プログラムが Confirmed コールを発行したときの会話の状態によって決まります。元の状態は、直前の Receive コールの *status_received* パラメーターの値により指定されています。 70 ページの表 14 は、*return_code* が CM_OK に設定されている場合の状態の変化を示しています。

*表 14. Confirmed コールによる状態の変化*

| 元の状態 | 新しい状態 |
|---|---|
| 確認 | 受信 |
| 確認 - 送信 | 送信 |
| 確認 - 割り振り解除 | リセット |

その他の戻りコードでは、状態は変化しません。

## 使用上の注意

Confirmed コールの使用上の注意の補足事項について、次の 2 つの項で説明します。

### 確認要求のソース

確認要求は、パートナー・プログラム内の次のコールのいずれかから発行されます。

- Confirm
- Prepare_To_Receive (受信準備タイプが CM_PREP_TO_RECEIVE_CONFIRM または CM_PREP_TO_RECEIVE_SYNC_LEVEL に設定されており、会話の同期レベルが CM_CONFIRM に設定されている場合)
- Deallocate (割り振り解除タイプが CM_DEALLOCATE_CONFIRM または CM_DEALLOCATE_SYNC_LEVEL に設定されており、会話の同期レベルが CM_CONFIRM に設定されている場合)
- Send_Data。これは次の状況の場合です。
  - 送信タイプが CM_SEND_AND_CONFIRM に設定されている。
  - 送信タイプが CM_SEND_AND_PREP_TO_RECEIVE に設定されており、受信準備タイプが CM_PREP_TO_RECEIVE_CONFIRM に設定されている。
  - 送信タイプが CM_SEND_AND_PREP_TO_RECEIVE に、受信準備タイプが CM_PREP_TO_RECEIVE_SYNC_LEVEL にそれぞれ設定されており、同期レベルが CM_CONFIRM に設定されている。
  - 送信タイプが CM_SEND_AND_DEALLOCATE に設定されており、割り振り解除タイプが CM_DEALLOCATE_CONFIRM に設定されている。
  - 送信タイプが CM_SEND_AND_DEALLOCATE に、割り振り解除タイプが CM_DEALLOCATE_SYNC_LEVEL にそれぞれ設定されており、同期レベルが CM_CONFIRM に設定されている。

### 確認要求の受信

ローカル・プログラムは、Receive コールの *status_received* パラメーターを介して確認要求を受信します。ローカル・プログラムが Confirmed コールを発行できるのは、*status_received* パラメーターが次の値のいずれかに設定されているときだけです。

- CM_CONFIRM_RECEIVED
- CM_CONFIRM_SEND_RECEIVED
- CM_CONFIRM_DEALLOC_RECEIVED

# Convert_Incoming (cmcnvi)

Convert_Incoming コールは、文字ストリングを EBCDIC から ASCII に変換します。パートナー・アプリケーションが EBCDIC 文字ストリングから成るデータを送信した場合、ローカル・アプリケーションは、Convert_Incoming を使ってこれらのストリングを ASCII に変換できます (*mode_name* や *TP_name* など、Send_Data コールおよび Receive コール内のデータ以外の CPI-C パラメーターは、常に ASCII で指定されるので変換は不要です)。

## 関数コール

```
void cmcnvi (
             unsigned char CM_PTR      string,
             CM_INT32 CM_PTR           string_length,
             CM_RETURN_CODE CM_PTR     return_code
            );
```

## Java CPI-C の関数コール

```
public native void cmcnvi (
                             byte[]          string,
                             CPICLength      string_length,
                             CPICReturnCode  return_code
                           );
```

## 指定パラメーター

指定パラメーターは次のとおりです。

*string* このパラメーターは、ASCII に変換する EBCDIC ストリングです。 CPI-C 仕様によれば、ストリングには以下の任意の文字 (文字セット 640) を使用することができます。

大文字の A〜Z、小文字の a〜z、0〜9、ピリオド (.)、スペース文字、および特殊文字 < + ( & * ) ; - / , % _ > ? : ' = "

さらに、CS/AIX CPI-C は以下の文字も受け入れます (他の CPI-C インプリメンテーションではサポートされていない場合があります)。

! # $ @ ¥ { } ~

` (逆引用符)

| (垂直バー)

¦ (分割垂直バー)

¬ (否定文字)

¢ (セント)

このストリングの内容が (*string_length* に指定されている最大文字数の範囲内で)、変換の結果 ASCII ストリングに置換されます。

*string_length*
このパラメーターは、変換される文字数です (1 〜 32,767)。

## 戻りパラメーター

verb の実行後に、CS/AIX は、処理が正常に実行されたかどうかを示すパラメーターと、失敗した場合はその理由を示すパラメーターを戻します。

*string*　このパラメーターは、変換の結果得られる ASCII ストリングです。*string_length* に指定されている文字数まで有効です。

*return_code*
値は次のとおりです。

**CM_OK**　コールは正常に実行されました。string パラメーターには変換済みの ASCII ストリングが入っています。

**CM_PROGRAM_PARAMETER_CHECK**
*buffer_length* パラメーターに、有効でない値が指定されていました。

**CM_PRODUCT_SPECIFIC_ERROR**
この戻りコードの説明については、191 ページの『付録 B. 共通な戻りコード』を参照してください。

## 発行時の状態

このコールは、会話には関連付けられていません。

## 状態の変化

状態には変化はありません。

## 使用上の注意

基本会話で (Set_Fill コールの指定に従って) バッファー形式でデータを受信している場合は、それぞれ 2 バイトまたは 4 バイトのヘッダー (LLID) と、それに続くデータで構成される複数の論理レコードがデータ・バッファーに入っている可能性があります。アプリケーションは、各データ・ストリング (ヘッダーは含まれない) を個別に抽出し変換する必要があります。バッファーの内容全体を 1 回の操作で変換しないでください。このような操作を行うと、ヘッダー値が無効になってしまいます。

---

# Convert_Outgoing (cmcnvo)

Convert_Outgoing コールは、文字ストリングを ASCII から EBCDIC に変換します。パートナー・アプリケーションが EBCDIC 文字ストリングから成るデータを必要としている場合、ローカル・アプリケーションは、Convert_Outgoing を使って、データを送信する前に ASCII から EBCDIC に変換できます (*mode_name* や *TP_name* など、Send_Data コールおよび Receive コール内のデータ以外の CPI-C パラメーターは、常に ASCII で指定されるので変換は不要です)。

## 関数コール

```
void cmcnvo (
             unsigned char CM_PTR      string,
             CM_INT32 CM_PTR           string_length,
             CM_RETURN_CODE CM_PTR     return_code
            );
```

## Java CPI-C の関数コール

```
public native void cmcnvo (
                            byte[]          string,
                            CPICLength      string_length,
                            CPICReturnCode  return_code
                          );
```

## 指定パラメーター

指定パラメーターは次のとおりです。

*string* このパラメーターは、EBCDIC に変換する ASCII ストリングです。CPI-C 仕様によれば、ストリングには以下の任意の文字 (文字セット 640) を使用することができます。

大文字の A〜Z、小文字の a〜z、0〜9、ピリオド (.)、スペース文字、および特殊文字 < + ( & * ) ; - / , % _ > ? : ' = "

さらに、CS/AIX CPI-C は以下の文字も受け入れます (他の CPI-C インプリメンテーションではサポートされていない場合があります)。

! # $ @ ¥ { } ~

` (逆引用符)

| (垂直バー)

| (分割垂直バー)

¬ (否定文字)

¢ (セント)

このストリングの内容が (*string_length* に指定されている最大文字数の範囲内で)、変換結果の EBCDIC ストリングに置換されます。

*string_length*
このパラメーターは、変換される文字数です (1 〜 32,767)。

## 戻りパラメーター

verb の実行後に、CS/AIX は、処理が正常に実行されたかどうかを示すパラメーターと、失敗した場合はその理由を示すパラメーターを戻します。

*string* このパラメーターは、変換の結果得られる EBCDIC ストリングです。*string_length* に指定されている文字数まで有効です。

*return_code*
値は次のとおりです。

**CM_OK** コールは正常に実行されました。string パラメーターには変換済みの EBCDIC ストリングが入っています。

**CM_PROGRAM_PARAMETER_CHECK**
*buffer_length* パラメーターに、有効でない値が指定されていました。

**CM_PRODUCT_SPECIFIC_ERROR**
この戻りコードの説明については、191 ページの『付録 B. 共通な戻りコード』を参照してください。

### 発行時の状態

このコールは、会話には関連付けられていません。

### 状態の変化

状態には変化はありません。

### 使用上の注意

基本会話で (Set_Fill コールの指定に従って) バッファー形式でデータを送信している場合は、それぞれ 2 バイトまたは 4 バイトのヘッダー (LLID) と、それに続くデータで構成される複数の論理レコードがデータ・バッファーに入っている可能性があります。アプリケーションは、各データ・ストリング (ヘッダーは含まれない) を個別に変換する必要があります。バッファーの内容全体を 1 回の操作で変換しないでください。このような操作を行うと、ヘッダー値が無効になってしまいます。

## Deallocate (cmdeal)

Deallocate コールは、2 つのプログラム間の会話の割り振りを解除します。

このコールは、会話の割り振りを解除する前に、現在の会話同期レベルと割り振り解除タイプに応じて、Flush コール、または Confirmed コールと同じ操作を実行します。割り振り解除タイプは、Set_Deallocate_Type コールにより設定されます。

パートナー・プログラムは、次のパラメーターのいずれかを介して割り振り解除通知を受け取ります。

- *status_received* = CM_CONFIRM_DEALLOC_RECEIVED
- *return_code* = CM_DEALLOCATED_NORMAL
- *return_code* = CM_DEALLOCATED_ABEND

このコールが正常に実行されたあとは、会話 ID は無効になります。

### 関数コール

```
void cmdeal (
             unsigned char CM_PTR      conversation_ID,
             CM_RETURN_CODE CM_PTR     return_code
            );
```

### Java CPI-C の関数コール

AIX、LINUX

```
public native void cmdeal (
                             byte[]          conversation_ID,
                             CPICReturnCode  return_code
                           );
```

## 指定パラメーター

指定パラメーターは次のとおりです。

*conversation_ID*
このパラメーターは会話の ID です。このパラメーターの値は、Initialize_Conversation コール、Initialize_For_Incoming コール、または Accept_Conversation コールから戻されます。

## 戻りパラメーター

verb の実行後に、CS/AIX は、処理が正常に実行されたかどうかを示すパラメーターと、失敗した場合はその理由を示すパラメーターを戻します。

*return_code*
値は次のとおりです。

**CM_OK** コールは正常に実行され、会話の割り振りが解除されました。

**CM_PROGRAM_PARAMETER_CHECK**
*conversation_ID* に指定された値は無効です。

**CM_PROGRAM_STATE_CHECK**
割り振り解除タイプに通常の割り振り解除 (CM_DEALLOCATE_SYNC_LEVEL、CM_DEALLOCATE_FLUSH、CM_DEALLOCATE_CONFIRM) が指定されている場合は、次の状態エラーが起きている可能性があります。

- 会話が送信状態または送信 - 保留状態でない。
- 会話は送信状態になっているが、プログラムが論理レコードを送信し終わっていない。

次の戻りコードの説明については、191 ページの『付録 B. 共通な戻りコード』を参照してください。

CM_OPERATION_INCOMPLETE
CM_OPERATION_NOT_ACCEPTED
CM_PRODUCT_SPECIFIC_ERROR

割り振り解除タイプが CM_DEALLOCATE_CONFIRM に設定されている場合、または割り振り解除タイプが CM_DEALLOCATE_SYNC_LEVEL に設定されていて会話の同期レベルが CM_CONFIRM に設定されている場合は、次の戻りコードが戻されます。これらの戻りコードの説明については、191 ページの『付録 B. 共通な戻りコード』 を参照してください。

CM_CONVERSATION_TYPE_MISMATCH
CM_DEALLOCATED_ABEND
CM_DEALLOCATED_ABEND_SVC
CM_DEALLOCATED_ABEND_TIMER
CM_PIP_NOT_SPECIFIED_CORRECTLY
CM_SECURITY_NOT_VALID
CM_SVC_ERROR_PURGING
CM_SYNC_LVL_NOT_SUPPORTED_PGM
CM_SYNC_LVL_NOT_SUPPORTED_LU
CM_TP_NOT_AVAILABLE_NO_RETRY
CM_TP_NOT_AVAILABLE_RETRY

CM_TPN_NOT_RECOGNIZED
CM_PROGRAM_ERROR_PURGING
CM_RESOURCE_FAILURE_NO_RETRY
CM_RESOURCE_FAILURE_RETRY

## 発行時の状態

プログラムが Deallocate コールを発行できる会話状態は、表 15 に示す状態のいずれかです。どの状態になるかは、Set_Deallocate_Type コールにより設定される会話の *deallocate_type* パラメーターの値に応じて決まります。

*表 15. Deallocate コールの発行時の会話状態*

| 割り振り解除タイプ | 発行可能な状態 |
|---|---|
| CM_DEALLOCATE_FLUSH CM_DEALLOCATE_CONFIRM CM_DEALLOCATE_SYNC_LEVEL | 送信または送信保留 |
| CM_DEALLOCATE_ABEND | リセット以外のすべて |

## 状態の変化

表 16 に要約して示した状態の変化は、*return_code* パラメーターの値に基づくものです。

*表 16. Deallocate コールによる状態の変化*

| *return_code* | 新しい状態 |
|---|---|
| CM_OK | リセット |
| CM_PROGRAM_ERROR_PURGING | 受信 |
| CM_SVC_ERROR_PURGING | |
| CM_CONVERSATION_TYPE_MISMATCH | リセット |
| CM_PIP_NOT_SPECIFIED_CORRECTLY | |
| CM_SECURITY_NOT_VALID | |
| CM_SYNC_LEVEL_NOT_SUPPORTED_PGM | |
| CM_SYNC_LEVEL_NOT_SUPPORTED_LU | |
| CM_TPN_NOT_RECOGNIZED | |
| CM_TP_NOT_AVAILABLE_NO_RETRY | |
| CM_TP_NOT_AVAILABLE_RETRY | |
| CM_RESOURCE_FAILURE_NO_RETRY | リセット |
| CM_RESOURCE_FAILURE_RETRY | |
| CM_DEALLOCATED_ABEND | リセット |
| CM_DEALLOCATED_ABEND_SVC | |
| CM_DEALLOCATED_ABEND_TIMER | |
| 上記以外 | 変化なし |

## 使用上の注意

会話の割り振り解除タイプが CM_DEALLOCATE_ABEND に設定されており、ログ・データ長が 0 (ゼロ) より大きい場合、ローカル LU はログ・データ (Set_Log_Data コールにより指定された) をローカル・エラー・ログ・ファイルおよびパートナー LU に書き込みます。ログ・データの詳細については、148 ページの『Set_Log_Data (cmsld)』を参照してください。

Deallocate コールが実行されたあとは、ログ・データ長が 0 (ゼロ) に設定され、ログ・データがヌルに設定されます。

## Delete_CPIC_Side_Information (xcmdsi)

この関数は Java CPI-C では使用できません。

Delete_CPIC_Side_Information コールは、以前にアプリケーションが Set_CPIC_Side_Information を使用して指定したサイド情報エントリーを削除します。または、構成ファイル内のエントリーがこのアプリケーションでこれ以上使用できないように指定します。 このエントリーは、シンボリック宛先名により識別されます。

このコールは、X/Open CPI-C および Windows CPI-C 仕様との互換性を確保するために提供されているもので、IBM CPI-C 2.0 には組み込まれていません。

### 関数コール

```
void xcmdsi (
             unsigned char CM_PTR      key,
             unsigned char CM_PTR      sym_dest_name,
             CM_RETURN_CODE CM_PTR     return_code
            );
```

### 指定パラメーター

指定パラメーターは次のとおりです。

*key* このパラメーターは無視されます。

*sym_dest_name*
このパラメーターは、削除するエントリーのシンボリック宛先名を示します。これは 8 バイトの ASCII 文字ストリングで、表示可能な文字はすべて使用することができます。

### 戻りパラメーター

verb の実行後に、CS/AIX は、処理が正常に実行されたかどうかを示すパラメーターと、失敗した場合はその理由を示すパラメーターを戻します。

*return_code*
値は次のとおりです。

**CM_OK** コールは正常に実行されました。

**CM_PROGRAM_PARAMETER_CHECK**
*sym_dest_name* パラメーターに、存在していないサイド情報エントリーが指定されています。

**CM_PRODUCT_SPECIFIC_ERROR**
この戻りコードの説明については、191 ページの『付録 B. 共通な戻りコード』を参照してください。

### 発行時の状態

このコールは、会話には関連付けられていません。

### 状態の変化

状態には変化はありません。

### 使用上の注意

このコールにより、構成ファイル内に保持されているサイド情報は変更されません。変更は、このアプリケーションだけに適用されます。CS/AIX は変更後の情報をこのオペレーティング・システム・プロセスに関連付けられたメモリーに保管します。そのプロセスが終了すると、変更内容は破棄されます。詳しくは、34 ページの『サイド情報』を参照してください。

## Extract_Conversation_Context (cmectx)

AIX、LINUX

Extract_Conversation_Context コールは、指定された会話のコンテキストを戻します。これによって、プログラムは、新規会話を開始する前に、(Set_Conversation_Context を使用して) 現行コンテキストを必要な値に設定して、新規会話で同じコンテキストを使うようにすることができます。

### 関数コール

```
void cmectx (
        unsigned char CM_PTR      conversation_ID,
        unsigned char CM_PTR      context_ID,
        CM_INT32 CM_PTR           context_ID_length,
        CM_RETURN_CODE CM_PTR     return_code
       );
```

### Java CPI-C の関数コール

```
public native void cmectx (
                            byte[]          conversation_ID,
                            byte[]          context_ID,
                            CPICLength      context_ID_length,
                            CPICReturnCode  return_code
                           );
```

### 指定パラメーター

指定パラメーターは次のとおりです。

*conversation_ID*

このパラメーターは会話の ID です。このパラメーターの値は、Initialize_Conversation コール、Initialize_For_Incoming コール、または Accept_Conversation コールから戻されます。

### 戻りパラメーター

verb の実行後に、CS/AIX は、処理が正常に実行されたかどうかを示すパラメーターと、失敗した場合はその理由を示すパラメーターを戻します。

*context_ID*
このパラメーターには、指定の会話のコンテキストが含まれています。この値が有効なのは、*return_code* パラメーターが CM_OK の場合のみです。

*context_ID_length*
このパラメーターには、*context_ID* の長さ (1〜32 バイト) が入っています。この値が有効なのは、*return_code* パラメーターが CM_OK の場合のみです。

*return_code*
値は次のとおりです。

**CM_OK** コールは正常に実行されました。

**CM_PROGRAM_PARAMETER_CHECK**
*conversation_ID* に指定された値は無効です。

**CM_PROGRAM_STATE_CHECK**
*conversation_ID* に指定された会話が、初期化状態または初期化 - 着呼状態です。

次の戻りコードの説明については、 191 ページの『付録 B. 共通な戻りコード』を参照してください。
CM_PRODUCT_SPECIFIC_ERROR

## 発行時の状態

会話は、リセット、初期化、初期化 - 着呼以外ならどの状態でも構いません。

## 状態の変化

状態には変化はありません。

## 使用上の注意

このコールは、プログラムの現行コンテキストを抽出値には設定しません。この設定をするには、プログラムで Set_Conversation_Context を呼び出す必要があります。

次の状況では、Extract_Conversation_Context とそれに続く Set_Conversation_Context をアプリケーションで使用します。

- アプリケーションで複数の会話を取り扱っていて、既存の会話と同じコンテキストを使用する新規会話を割り振る必要がある場合。
- 新規コンテキストを割り当てる CPI-C コールが非ブロッキング・モードで完了する場合。たとえば、Accept_Incoming が即時に完了し *return_code* CM_OK を戻した場合は、プログラムの現行コンテキストは新規会話のコンテキストに設定されます。しかし、Accept_Incoming が CM_OPERATION_INCOMPLETE を戻した場合は、後続の Wait_For_Conversation から Accept_Incoming の結果が戻されても、プログラムの現行コンテキストは変更されません。プログラムは、Extract_Conversation_Context および Set_Conversation_Context を使用して、現行コンテキストを正しい値に設定する必要があります。

# Extract_Conversation_Security_Type (xcecst)

この関数は Java CPI-C では使用できません。

Extract_Conversation_Security_Type コールは、指定された会話のセキュリティー・タイプを戻します。

このコールは、X/Open CPI-C および Windows CPI-C 仕様との互換性を確保するために提供されているもので、IBM CPI-C 2.0 には組み込まれていません。

## 関数コール

```
void xcecst (
             unsigned char CM_PTR                    conversation_ID,
             XC_CONVERSATION_SECURITY_TYPE CM_PTR    conversation_security_type,
             CM_RETURN_CODE CM_PTR                   return_code
            );
```

## 指定パラメーター

指定パラメーターは次のとおりです。

*conversation_ID*
このパラメーターは会話の ID です。このパラメーターの値は、Initialize_Conversation コール、Initialize_For_Incoming コール、または Accept_Conversation コールから戻されます。

## 戻りパラメーター

verb の実行後に、CS/AIX は次のパラメーターを戻します。

*conversation_security_type*
このパラメーターは、パートナー LU が呼び出し対象プログラムへのアクセスの妥当性検査をするのに必要な情報を指定します。値は次のとおりです。

AIX、LINUX

**CM_SECURITY_NONE**
呼び出し対象プログラムは会話セキュリティーを使用していません。

**CM_SECURITY_SAME**
この値は、有効なユーザー ID とパスワードを使用して呼び出された呼び出し対象プログラムが、別のプログラムを呼び出すときに使用されます (1 ページの『第 1 章 概念』の説明を参照)。プログラム A が有効なユーザー ID とパスワードを使用してプログラム B を呼び出し、プログラム B が次のプログラム C を呼び出す場合、プログラム B が値 CM_SECURITY_SAME を指定していれば、CPI-C は、プログラム C の LU に検査済みインディケーターを送ります。このインディケーターは、プログラム C に、パスワードが不要なことを知らせます (プログラム C が検査済みインディケーターを受け入れる構成になっている場合)。

**CM_SECURITY_PROGRAM**
呼び出し対象プログラムは会話セキュリティーを使用しているため、ユーザー ID とパスワードが必要です。

**CM_SECURITY_PROGRAM_STRONG**
ローカル・ノードはネットワークを介して平文形式でパスワードを送信してはいけない、という点以外は CM_SECURITY_PROGRAM と同じです。この値を使用できるのは、リモート・システムがパスワードの置換をサポートしている場合だけです。

WINDOWS

**XC_SECURITY_NONE**
CM_SECURITY_NONE と同等

**XC_SECURITY_SAME**
CM_SECURITY_SAME と同等

**XC_SECURITY_PROGRAM**
CM_SECURITY_PROGRAM と同等

*return_code*
値は次のとおりです。

**CM_OK** コールは正常に実行されました。

**CM_PROGRAM_PARAMETER_CHECK**
*conversation_ID* に指定された値は無効です。

## 発行時の状態

会話は、リセット以外であればどの状態にあっても構いません。

## 状態の変化

状態には変化はありません。

# Extract_Conversation_Security_User_ID (cmecsu)

WINDOWS

このコールは、AIX または Linux の CPI-C コール Extract_Security_User_ID (cmesui) と同等の Windows CPI-C です。この 2 つのコールは、名前が異なる点以外、まったく同じように使用されます。 Extract_Conversation_Security_User_ID についての詳細は、 92 ページの『Extract_Security_User_ID (cmesui または cmecsu)』を参照し、AIX または Linux 関数の名前とスードニム (pseudonym) を、指示通りに Windows 関数の名前とスードニム (pseudonym) に置き換えます。

## Extract_Conversation_Security_User_ID (xcecsu)

この関数は Java CPI-C では使用できません。

このコールは、指定の会話で使用中のユーザー ID を戻します。

このコールは、X/Open CPI-C 定義を使用するアプリケーションに互換性を提供します。これは、コール Extract_Security_User_ID (cmesui) としてすでに IBM CPI-C 2.0 に組み込まれています。作成するプログラムの他のプラットフォームへの移植性を高めるために、可能な限り cmesui を使用してください。

このコールのパラメーターは、cmesui コールのパラメーターと同一です。cmesui の詳細については、92 ページの『Extract_Security_User_ID (cmesui または cmecsu)』を参照してください。

## Extract_Conversation_State (cmecs)

Extract_Conversation_State コールは、指定された会話の状態を戻します。

### 関数コール

```
void cmecs (
            unsigned char CM_PTR             conversation_ID,
            CM_CONVERSATION_STATE CM_PTR     conversation_state,
            CM_RETURN_CODE CM_PTR            return_code
           );
```

### Java CPI-C の関数コール

AIX、LINUX

```
public native void cmecs (
                           byte[]                conversation_ID,
                           CPICConversationState  conversation_state,
                           CPICReturnCode         return_code
                         );
```

### 指定パラメーター

指定パラメーターは次のとおりです。

*conversation_ID*

このパラメーターは会話の ID です。このパラメーターの値は、Initialize_Conversation コール、Initialize_For_Incoming コール、または Accept_Conversation コールから戻されます。

## 戻りパラメーター

verb の実行後に、CS/AIX は、処理が正常に実行されたかどうかを示すパラメーターと、失敗した場合はその理由を示すパラメーターを戻します。

*conversation_state*
会話状態を示します。値は次のとおりです。

```
CM_INITIALIZE_STATE
CM_INITIALIZE_INCOMING_STATE
CM_SEND_STATE
CM_RECEIVE_STATE
CM_SEND_PENDING_STATE
CM_CONFIRM_STATE
CM_CONFIRM_SEND_STATE
CM_CONFIRM_DEALLOCATE_STATE
```

*return_code*
値は次のとおりです。

**CM_OK** コールは正常に実行されました。

**CM_PROGRAM_PARAMETER_CHECK**
*conversation_ID* に指定された値は無効です。

次の戻りコードの説明については、191 ページの『付録 B. 共通な戻りコード』を参照してください。

```
CM_OPERATION_INCOMPLETE
CM_PRODUCT_SPECIFIC_ERROR
```

## 発行時の状態

会話は、リセット以外であればどの状態にあっても構いません。

## 状態の変化

状態には変化はありません。

---

# Extract_Conversation_Type (cmect)

Extract_Conversation_Type コールは、指定された会話の会話タイプ (マップ式または基本) を戻します。

## 関数コール

```
void cmect (
           unsigned char CM_PTR             conversation_ID,
           CM_CONVERSATION_TYPE CM_PTR      conversation_type,
           CM_RETURN_CODE CM_PTR            return_code
          );
```

## Java CPI-C の関数コール

AIX、LINUX

```
public native void cmect (
                            byte[]                conversation_ID,
                            CPICConversationType  conversation_type,
                            CPICReturnCode        return_code
                          );
```

## 指定パラメーター

指定パラメーターは次のとおりです。

*conversation_ID*
このパラメーターは会話の ID です。このパラメーターの値は、Initialize_Conversation コール、Initialize_For_Incoming コール、または Accept_Conversation コールから戻されます。

## 戻りパラメーター

verb の実行後に、CS/AIX は、処理が正常に実行されたかどうかを示すパラメーターと、失敗した場合はその理由を示すパラメーターを戻します。

*conversation_type*
このパラメーターは、会話タイプを示します。値は次のとおりです。

```
CM_BASIC_CONVERSATION
CM_MAPPED_CONVERSATION
```

*return_code*
値は次のとおりです。

**CM_OK** コールは正常に実行されました。

**CM_PROGRAM_PARAMETER_CHECK**
*conversation_ID* に指定された値は無効です。

**CM_PRODUCT_SPECIFIC_ERROR**
この戻りコードの説明については、191 ページの『付録 B. 共通な戻りコード』を参照してください。

## 発行時の状態

会話は、リセット以外であればどの状態にあっても構いません。

## 状態の変化

状態には変化はありません。

---

# Extract_CPIC_Side_Information (xcmesi)

この関数は Java CPI-C では使用できません。

Extract_CPIC_Side_Information コールは、エントリー番号またはシンボリック宛先名についてのサイド情報を戻します。

このコールは、X/Open CPI-C および Windows CPI-C 仕様との互換性を確保するために提供されているもので、IBM CPI-C 2.0 には組み込まれていません。

## 関数コール

```
  void xcmesi  (
                CM_INT32 CM_PTR             entry_number,
                unsigned char CM_PTR        sym_dest_name,
                SIDE_INFO CM_PTR            side_info_entry,
                CM_INT32 CM_PTR             side_info_entry_length,
                CM_RETURN_CODE CM_PTR       return_code
               );
typedef struct side_info_entry
{
 unsigned char    sym_dest_name[8];           /* symbolic destination name   */
 unsigned char    partner_LU_name[17];        /* Fully qualified partner LU  */
                                              /*  name                       */
 unsigned char    reserved[3];                /* Reserved                    */
 XC_TP_NAME_TYPE  TP_name_type;               /* TP name type                */
 unsigned char    TP_name[64];                /* TP name                     */
 unsigned char    mode_name[8];               /* Mode name                   */
 XC_CONVERSATION_SECURITY_TYPE
                  conversation_security_type; /* Conversation security type  */
 unsigned char    security_user_ID[8];        /* User ID                     */
 unsigned char    security_password[8];       /* Password                    */
} SIDE_INFO;
```

## 指定パラメーター

指定パラメーターは次のとおりです。

*entry_number*
このパラメーターは無視されます。

*sym_dest_name*
このパラメーターは、検索するシンボリック宛先名を指定します。これは 8 バイトの ASCII 文字ストリングで、表示可能な文字はすべて使用することができます。

*side_info_entry_length*

AIX、LINUX

この値は常に `sizeof(SIDE_INFO)` に設定する必要があります。

WINDOWS

この値は常に `124` に設定する必要があります。

## 戻りパラメーター

verb の実行後に、CS/AIX は、処理が正常に実行されたかどうかを示すパラメーターと、失敗した場合はその理由を示すパラメーターを戻します。

*side_info_entry*
このパラメーターは、次のように、サイド情報エントリーの内容を示します。

*side_info_entry.sym_dest_name*
サイド情報エントリーを識別するシンボリック宛先名。パラメーター *sym_dest_name* は、8 バイトの ASCII 文字ストリングで、表示可能な文字はすべて使用することができます。

*side_info_entry.partner_LU_name*
パートナー LU の完全修飾名。この名前は、それぞれ 1〜8 バイトから成りドットで連結された、2 つの文字ストリングで構成されます。

*side_info_entry.TP_name_type*
ターゲット TP のタイプ (TP 名に対して有効な文字は、TP タイプによって決まります)。値は次のとおりです。

**XC_APPLICATION_TP**
アプリケーション TP。TP 名のすべての文字は、有効な ASCII 文字でなければなりません。

**XC_SNA_SERVICE_TP**
サービス TP。TP 名は、16 進数字 (4 文字) 2 つを表す 8 文字の ASCII ストリングとして指定しなければなりません。たとえば、名前の 16 進表示が 0x21F0F0F8 の場合は、*TP_name* パラメーターを 8 文字ストリング「21F0F0F8」に設定します。

最初の文字 (2 バイトで表されます) は、0x0E および 0x0F を除く、0x0〜0x3F の範囲の 16 進値でなければなりません。残りの文字 (それぞれ 2 バイトで表されます) は、有効な EBCDIC 文字でなければなりません。

*side_info_entry.TP_name*
ターゲット TP の TP 名。

*side_info_entry.mode_name*
ターゲット TP にアクセスするために使用するモードの名前。

*side_info_entry.conversation_security_type*
ターゲット TP が会話セキュリティーを使用するかどうかを示します。値は次のとおりです。

**XC_SECURITY_NONE**
ターゲット TP は会話セキュリティーを使用していません。

**XC_SECURITY_PROGRAM**
ターゲット TP は会話セキュリティーを使用しています。ターゲット TP へのアクセスには、以下に示す *security_user_ID* と *security_password* という 2 つのパラメーターが使用されます。

**XC_SECURITY_SAME**
ターゲット TP は会話セキュリティーを使用していて、ローカル TP からの「検査済み」インディケーターを受け入れることができます。(これは、ローカル TP 自体が別の TP から呼び出され、その TP から提供されたセキュリティー・ユーザー ID とパスワード

を検査したことを示しています)。ターゲット TP へのアクセスには、以下に示す *security_user_ID* パラメーターが使用されます。パスワードは不要です。

*side_info_entry.security_user_ID*
パートナー TP にアクセスするために使用するユーザー ID。*conversation_security_type* パラメーターが `XC_SECURITY_NONE` に設定されている場合は、このパラメーターは不要です。

セキュリティー・ユーザー ID には 10 文字まで指定できますが、X/Open CPI-C との互換性を確保するために、この verb はユーザー ID について 8 文字だけを戻します。完全なユーザー ID を取得できるようにするには、ここで戻される値に依存せずに、Extract_Security_User_ID コール (Windows システムの場合は Extract_Conversation_Security_User_ID) を使用して、ユーザー ID を明示的に抽出する必要があります。

*side_info_entry.security_password*
これは予約パラメーターです。アプリケーションにはパスワード情報は戻されません。

*return_code*
値は次のとおりです。

**CM_OK** コールは正常に実行されました。

**CM_PROGRAM_PARAMETER_CHECK**
次のいずれかの状態が発生しました。

- *sym_dest_name* パラメーターが有効でない。
- *side_info_entry_length* パラメーターが `sizeof(SIDE_INFO)` に設定されていない。

**CM_PRODUCT_SPECIFIC_ERROR**
この戻りコードの説明については、191 ページの『付録 B. 共通な戻りコード』を参照してください。

## 発行時の状態

このコールは、会話には関連付けられていません。

## 状態の変化

状態には変化はありません。

## 使用上の注意

サイド情報のセキュリティー・ユーザー ID が設定されていない場合は、セキュリティー・ユーザー ID フィールドにスペースが埋め込まれて戻されます。

# Extract_Local_LU_Name (cmelln)

Extract_Local_LU_Name コールは、指定された会話についてローカル LU の別名を戻します。

このコールは標準 CPI-C 仕様の一部ではありません。他のインプリメンテーションでは使用できない場合があります。特に、他の Java CPI-C インプリメンテーションではサポートされていません。

## 関数コール

```
void cmelln   (
               unsigned char CM_PTR         conversation_ID,
               unsigned char CM_PTR         lu_alias,
               CM_RETURN_CODE CM_PTR        return_code
              );
```

## Java CPI-C の関数コール

AIX、LINUX

```
public native void cmelln (
                             byte[]          conversation_ID,
                             byte[]          lu_alias,
                             CPICReturnCode  return_code
                           );
```

## 指定パラメーター

指定パラメーターは次のとおりです。

*conversation_ID*
このパラメーターは会話の ID です。このパラメーターの値は、Initialize_Conversation コール、Initialize_For_Incoming コール、または Accept_Conversation コールから戻されます。

## 戻りパラメーター

verb の実行後に、CS/AIX は、処理が正常に実行されたかどうかを示すパラメーターと、失敗した場合はその理由を示すパラメーターを戻します。

*lu_alias*
ローカル LU の LU 別名。

*return_code*
値は次のとおりです。

**CM_OK** コールは正常に実行されました。

**CM_PROGRAM_PARAMETER_CHECK**
*conversation_ID* に指定された値は無効です。

**CM_PRODUCT_SPECIFIC_ERROR**
この戻りコードの説明については、191 ページの『付録 B. 共通な戻りコード』を参照してください。

## 発行時の状態

会話は、リセット以外であればどの状態にあっても構いません。

## 状態の変化

状態には変化はありません。

## 使用上の注意

このコールから戻される LU の別名は、必ずしも 146 ページの『Set_Local_LU_Name (cmslln)』で説明する Set_Local_LU_Name を使用して設定する必要はありません。 38 ページの『ローカル LU の指定』で説明した方法であればどれでも使用できます。

---

# Extract_Maximum_Buffer_Size (cmembs)

AIX、LINUX

Extract_Maximum_Buffer_Size コールは、CPI-C データ・バッファーの最大サイズを戻します。これによって、1 回の Send_Data コールで送信できるデータまたは 1 回の Receive コールで受信できるデータの最大量が定義されます。

CS/AIX CPI-C は、データ・バッファー・サイズとして常に 32,767 バイトを使用します。しかし、他のインプリメンテーション形態の CPI-C (または CS/AIX の将来のバージョン) との互換性を確保するために、アプリケーションでは、この値に依存せずに、このコールを使用して使用可能な最大バッファー・サイズを判別するようにしてください。

## 関数コール

```
void cmembs (
             CM_INT32 CM_PTR           maximum_buffer_size,
             CM_RETURN_CODE CM_PTR     return_code
            );
```

## Java CPI-C の関数コール

```
public native void cmembs (
                             CPICLength      maximum_bufer_size,
                             CPICReturnCode  return_code
                           );
```

## 指定パラメーター

このコールには、指定パラメーターはありません。

## 戻りパラメーター

verb の実行後に、CS/AIX は、処理が正常に実行されたかどうかを示すパラメーターと、失敗した場合はその理由を示すパラメーターを戻します。

*maximum_buffer_size*
このパラメーターは、データ・バッファーの長さを示します。

*return_code*
値は次のとおりです。

**CM_OK** コールは正常に実行されました。

**CM_PRODUCT_SPECIFIC_ERROR**
この戻りコードの説明については、 191 ページの『付録 B. 共通な戻りコード』を参照してください。

### 発行時の状態

このコールは、どの会話にも関連付けられていません。

### 状態の変化

状態には変化はありません。

---

## Extract_Mode_Name (cmemn)

Extract_Mode_Name コールは、指定された会話のモード名とモード名の長さを戻します。

### 関数コール

```
void cmemn (
            unsigned char CM_PTR       conversation_ID,
            unsigned char CM_PTR       mode_name,
            CM_INT32 CM_PTR            mode_name_length,
            CM_RETURN_CODE CM_PTR      return_code
           );
```

### Java CPI-C の関数コール

AIX、LINUX

```
public native void cmemn (
                           byte[]          conversation_ID,
                           byte[]          mode_name,
                           CPICLength      mode_name_length,
                           CPICReturnCode  return_code
                         );
```

### 指定パラメーター

指定パラメーターは次のとおりです。

*conversation_ID*
このパラメーターは会話の ID です。このパラメーターの値は、Initialize_Conversation コール、Initialize_For_Incoming コール、または Accept_Conversation コールから戻されます。

## 戻りパラメーター

verb の実行後に、CS/AIX は、処理が正常に実行されたかどうかを示すパラメーターと、失敗した場合はその理由を示すパラメーターを戻します。

*mode_name*
このパラメーターは、モード名の開始アドレスを示します。

*mode_name_length*
このパラメーターは、モード名の長さを示します。

*return_code*
値は次のとおりです。

**CM_OK** コールは正常に実行されました。

**CM_PROGRAM_PARAMETER_CHECK**
*conversation_ID* に指定された値は無効です。

**CM_PRODUCT_SPECIFIC_ERROR**
この戻りコードの説明については、191 ページの『付録 B. 共通な戻りコード』を参照してください。

## 発行時の状態

会話は、リセット以外であればどの状態にあっても構いません。

## 状態の変化

状態には変化はありません。

---

# Extract_Partner_LU_Name (cmepln)

Extract_Partner_LU_Name コールは、指定された会話のパートナー LU 名とパートナー LU 名の長さを戻します。これは、最大 8 バイトの別名、または最大 17 バイトの完全修飾ネットワーク名です。

## 関数コール

```
void cmepln (
             unsigned char CM_PTR      conversation_ID,
             unsigned char CM_PTR      partner_LU_name,
             CM_INT32 CM_PTR           partner_LU_name_length,
             CM_RETURN_CODE CM_PTR     return_code
            );
```

## Java CPI-C の関数コール

AIX、LINUX

```
public native void cmepln (
                           byte[]          conversation_ID,
                           byte[]          partner_LU_name,
                           CPICLength      partner_LU_name_length,
                           CPICReturnCode  return_code
                          );
```

## 指定パラメーター

指定パラメーターは次のとおりです。

*conversation_ID*
このパラメーターは会話の ID です。このパラメーターの値は、Initialize_Conversation コール、Initialize_For_Incoming コール、または Accept_Conversation コールから戻されます。

## 戻りパラメーター

verb の実行後に、CS/AIX は、処理が正常に実行されたかどうかを示すパラメーターと、失敗した場合はその理由を示すパラメーターを戻します。

*partner_LU_name*
このパラメーターは、パートナー LU 名が入っている変数を示します (プログラムは、適切な変数を指すポインターを指定する必要があります)。

*partner_LU_name_length*
このパラメーターは、パートナー LU 名の長さを示します。

*return_code*
値は次のとおりです。

**CM_OK** コールは正常に実行されました。

**CM_PROGRAM_PARAMETER_CHECK**
*conversation_ID* に指定された値は無効です。

**CM_PROGRAM_STATE_CHECK**
*conversation_ID* に指定された会話が初期化 - 着呼状態です。

**CM_PRODUCT_SPECIFIC_ERROR**
この戻りコードの説明については、191 ページの『付録 B. 共通な戻りコード』を参照してください。

## 発行時の状態

会話は、リセット状態または初期化 - 着呼状態以外ならどの状態でも構いません。

## 状態の変化

状態には変化はありません。

---

# Extract_Security_User_ID (cmesui または cmecsu)

Extract_Security_User_ID コールは、指定された会話で使用中のユーザー ID を戻します。

WINDOWS

このコールは、Windows CPI-C インターフェースとの互換性を確保するため、スードニム (pseudonym) cmecsu を付けて、 指定された Extract_Conversation_Security_User_ID です。

## 関数コール

```
void cmesui (
             unsigned char CM_PTR      conversation_ID,
             unsigned char CM_PTR      security_user_ID,
             CM_INT32 CM_PTR           security_user_ID_length,
             CM_RETURN_CODE CM_PTR     return_code
            );
```

WINDOWS

Windows システムでは、cmesui を cmecsu に置き換えてください。

## Java CPI-C の関数コール

AIX、LINUX

```
public native void cmesui (
                           byte[]          conversation_ID,
                           byte[]          security_user_ID,
                           CPICLength      security_user_ID_length,
                           CPICReturnCode  return_code
                          );
```

## 指定パラメーター

指定パラメーターは次のとおりです。

*conversation_ID*
このパラメーターは会話の ID です。このパラメーターの値は、Initialize_Conversation コール、Initialize_For_Incoming コール、または Accept_Conversation コールから戻されます。

## 戻りパラメーター

verb の実行後に、CS/AIX は次のパラメーターを戻します。

*security_user_ID*
このパラメーターは、会話を確立するために使用するユーザー ID を示します。

*security_user_ID_length*
このパラメーターは、*security_user_ID* の長さを示します。

この値の範囲は、1 から 10 文字 (AIX または Linux システム)、または 1 から 8 文字 (Windows システム) です。 *security_user_ID_length* が 0 (ゼロ) に設定されている場合は、*security_user_ID_length* パラメーターは無視されます。これは、*security_user_ID* をヌル・ストリングに設定するのと同じです。

*return_code*
値は次のとおりです。

**CM_OK** コールは正常に実行されました。

**CM_PROGRAM_PARAMETER_CHECK**
*conversation_ID* に指定された値は無効です。

**CM_PRODUCT_SPECIFIC_ERROR**
この戻りコードの説明については、191 ページの『付録 B. 共通な戻りコード』を参照してください。

### 発行時の状態

会話は、リセット以外であればどの状態にあっても構いません。

### 状態の変化

状態には変化はありません。

### 使用上の注意

*security_user_ID* の値にはブランクは埋め込まれません。有効なのは、*security_user_ID_length* に指定された長さまでです。

## Extract_Sync_Level (cmesl)

Extract_Sync_Level コールは、指定された会話の同期レベルを戻します。

### 関数コール

```
void cmesl (
            unsigned char CM_PTR       conversation_ID,
            CM_INT32 CM_PTR            sync_level,
            CM_RETURN_CODE CM_PTR      return_code
           );
```

### Java CPI-C の関数コール

AIX、LINUX

```
public native void cmesl (
                           byte[]          conversation_ID,
                           CPICSyncLevel   sync_level,
                           CPICReturnCode  return_code
                         );
```

## 指定パラメーター

指定パラメーターは次のとおりです。

*conversation_ID*
このパラメーターは会話の ID です。このパラメーターの値は、Initialize_Conversation コール、Initialize_For_Incoming コール、または Accept_Conversation コールから戻されます。

## 戻りパラメーター

verb の実行後に、CS/AIX は、処理が正常に実行されたかどうかを示すパラメーターと、失敗した場合はその理由を示すパラメーターを戻します。

*sync_level*
このパラメーターは会話の同期レベルを示します。値は次のとおりです。

**CM_NONE**
プログラムは確認処理を実行しません。

**CM_CONFIRM**
プログラムは確認処理を実行できます。

*return_code*
値は次のとおりです。

**CM_OK** コールは正常に実行されました。

**CM_PROGRAM_PARAMETER_CHECK**
*conversation_ID* に指定された値は無効です。

**CM_PROGRAM_STATE_CHECK**
*conversation_ID* に指定された会話が初期化 - 着呼状態です。

**CM_PRODUCT_SPECIFIC_ERROR**
この戻りコードの説明については、191 ページの『付録 B. 共通な戻りコード』を参照してください。

## 発行時の状態

会話は、リセット状態または初期化 - 着呼状態以外ならどの状態でも構いません。

## 状態の変化

状態には変化はありません。

# Extract_TP_Name (cmetpn)

Extract_TP_Name コールは、指定された会話の呼び出し対象 TP の TP 名と TP 名の長さを戻します。

アプリケーションが、Specify_Local_TP_Name コールを使用して複数の TP 名を対象とした着呼 Allocate の受け付けを指定し、その後 Accept_Conversation または

Accept_Incoming を発行して着呼 Allocate を受け付けた場合、このコールを使用して、着呼 Allocate に指定された TP 名がどれかを判別することができます。

## 関数コール

```
void cmetpn (
             unsigned char CM_PTR      conversation_ID,
             unsigned char CM_PTR      TP_name,
             CM_INT32 CM_PTR           TP_name_length,
             CM_RETURN_CODE CM_PTR     return_code
            );
```

## Java CPI-C の関数コール

AIX、LINUX

```
public native void cmetpn (
                             byte[]          conversation_ID,
                             byte[]          TP_name,
                             CPICLength      TP_name_length,
                             CPICReturnCode  return_code
                           );
```

## 指定パラメーター

指定パラメーターは次のとおりです。

*conversation_ID*
このパラメーターは会話の ID です。このパラメーターの値は、Initialize_Conversation コール、Initialize_For_Incoming コール、または Accept_Conversation コールから戻されます。

## 戻りパラメーター

verb の実行後に、CS/AIX は、処理が正常に実行されたかどうかを示すパラメーターと、失敗した場合はその理由を示すパラメーターを戻します。

*TP_name*
このパラメーターは、TP 名の開始アドレスを示します。

*TP_name_length*
このパラメーターは、TP 名の長さを示します。

*return_code*
値は次のとおりです。

**CM_OK** コールは正常に実行されました。

**CM_PROGRAM_PARAMETER_CHECK**
*conversation_ID* に指定された値は無効です。

**CM_PROGRAM_STATE_CHECK**
会話がリセット状態または初期化 - 着呼状態になっています。

**CM_PRODUCT_SPECIFIC_ERROR**
この戻りコードの説明については、 191 ページの『付録 B. 共通な戻りコード』を参照してください。

### 発行時の状態

会話は、リセット状態または初期化 - 着呼状態以外ならどの状態でも構いません。

### 状態の変化

状態には変化はありません。

## Flush (cmflus)

Flush コールは、ローカル LU の送信バッファーの内容をパートナー LU (およびプログラム) に送信します。送信バッファーが空のときは、アクションは何も実行されません。

### バッファーに入れられるデータのソース

Send_Data コールにより処理されたデータは、次のどちらかの状態が発生するまで、ローカル LU の送信バッファーに入れられます。

- ローカル・プログラムが、Flush コールなど LU の送信バッファーをフラッシュするコールを発行した。(Set_Send_Type コールにより設定される送信タイプには、フラッシュ機能が備わっているものもあります。)
- バッファーが満ぱいになる。

Allocate コールにより生成される割り振り要求、および Send_Error コールにより生成されるエラー情報も、バッファーに入れられます。

### 関数コール

```
void cmflus (
             unsigned char CM_PTR      conversation_ID,
             CM_RETURN_CODE CM_PTR     return_code
            );
```

### Java CPI-C の関数コール

AIX、LINUX

```
public native void cmflus (
                             byte[]          conversation_ID,
                             CPICReturnCode  return_code
                           );
```

### 指定パラメーター

指定パラメーターは次のとおりです。

*conversation_ID*
このパラメーターは会話の ID です。

このパラメーターの値は、Initialize_Conversation コール、Initialize_For_Incoming コール、または Accept_Conversation コールから戻されます。

## 戻りパラメーター

verb の実行後に、CS/AIX は、処理が正常に実行されたかどうかを示すパラメーターと、失敗した場合はその理由を示すパラメーターを戻します。

*return_code*
値は次のとおりです。

**CM_OK** コールは正常に実行されました。

**CM_PROGRAM_PARAMETER_CHECK**
*conversation_ID* に指定された値は無効です。

**CM_PROGRAM_STATE_CHECK**
プログラムがこのコールを発行したとき、会話が送信状態または送信 - 保留状態になっていませんでした。

次の戻りコードの説明については、191 ページの『付録 B. 共通な戻りコード』を参照してください。
CM_OPERATION_INCOMPLETE
CM_OPERATION_NOT_ACCEPTED
CM_PRODUCT_SPECIFIC_ERROR

## 発行時の状態

会話は、送信状態または送信 - 保留状態になっていなければなりません。

## 状態の変化

コールが正常に完了する (*return_code* = CM_OK) と、会話は送信状態になります。

その他の戻りコードでは、状態は変化しません。

# Initialize_Conversation (cminit)

Initialize_Conversation コールは、8 バイトの会話 ID を取得し、その会話の特性に初期値を設定するために、呼び出し側プログラムが発行します。

初期値は、CPI-C のデフォルト値、またはシンボリック宛先名に関連付けられたサイド情報から抽出された値です。初期値とサイド情報の詳細については、21 ページの『第 2 章 CPI-C アプリケーションの作成』を参照してください。

このコールが正常に実行されると、CPI-C は会話 ID を生成します。この ID は、呼び出し側プログラムがこの会話について発行する他のすべての CPI-C コールの必須パラメーターです。

初期値は、Set_* コールを使用して変更できます。

## 関数コール

```
void cminit (
             unsigned char CM_PTR      conversation_ID,
             unsigned char CM_PTR      sym_dest_name,
             CM_RETURN_CODE CM_PTR     return_code
            );
```

## Java CPI-C の関数コール

AIX、LINUX

```
public native void cminit (
                            byte[]          conversation_ID,
                            String          sym_dest_name,
                            CPICReturnCode  return_code
                           );
```

## 指定パラメーター

指定パラメーターは次のとおりです。

*sym_dest_name*

このパラメーターは、シンボリック宛先名を指定します。シンボリック宛先名は、 CS/AIX 構成ファイルからロードされるサイド情報エントリーに関連付けられた名前、または Set_CPIC_Side_Information コールにより定義された名前です。

このパラメーターは 8 バイトの ASCII 文字ストリングであり、表示可能な文字はすべて使用することができます。このパラメーターには、スペース 8 個を設定することもできます。その場合は、呼び出し側プログラムは、Allocate コールを発行する前に次のコールを発行する必要があります。

- Set_Mode_Name
- Set_Partner_LU_Name
- Set_TP_Name

サイド情報エントリーの詳細については、 136 ページの『Set_CPIC_Side_Information (xcmssi)』を参照してください。

## 戻りパラメーター

verb の実行後に、CS/AIX は、処理が正常に実行されたかどうかを示すパラメーターと、失敗した場合はその理由を示すパラメーターを戻します。

*conversation_ID*

このパラメーターは会話の ID です。この ID は、後続の CPI-C コールにより使用されます。

*return_code*

値は次のとおりです。

**CM_OK** コールは正常に実行されました。

**CM_PROGRAM_PARAMETER_CHECK**
次のいずれかの状態が発生しました。

- *sym_dest_name* に指定された値が、構成ファイルに定義されているシンボリック宛先名、またはプログラムで Set_CPIC_Side_Information を使用して指定されているシンボリック宛先名に一致しない。
- *conversation_ID* パラメーターが有効でない。

**CM_PRODUCT_SPECIFIC_ERROR**
この戻りコードの説明については、191 ページの『付録 B. 共通な戻りコード』を参照してください。

## 発行時の状態

会話がリセット状態になっています。

## 状態の変化

*return_code* が CM_OK の場合は、会話が初期化状態に変化します。その他の戻りコードの場合は、会話の状態は変わりません。

## 使用上の注意

会話特性に有効でない値がサイド情報に含まれている場合、または Set_* コールにより有効でない値がサイド情報に設定されている場合は、Allocate コールの発行時にエラーが戻ります。

---

# Initialize_For_Incoming (cminic)

AIX、LINUX

Initialize_For_Incoming コールは、8 バイトの会話 ID を取得するために、呼び出し対象プログラムが発行します。次に、プログラムは、Accept_Incoming コールを使用して会話を受け入れます。

Initialize_For_Incoming に続けて Accept_Incoming を発行するのは、Accept_Conversation を発行するのと同じです。違いは、Set_Processing_Mode は Initialize_For_Incoming と Accept_Incoming の間で発行して Accept_Incoming が非ブロッキング・モードで動作するようにできるのに対して、Accept_Conversation は常にブロッキング・モードで動作する点です。

## 関数コール

```
void cminic (
             unsigned char CM_PTR      conversation_ID,
             CM_RETURN_CODE CM_PTR     return_code
            );
```

## Java CPI-C の関数コール

```
public native void cminic (
                            byte[]          conversation_ID,
                            CPICReturnCode  return_code
                          );
```

## 指定パラメーター

このコールには、指定パラメーターはありません。

## 戻りパラメーター

verb の実行後に、CS/AIX は次のパラメーターを戻します。

*conversation_ID*
このパラメーターは会話の ID です。この ID は、後続の CPI-C コールにより使用されます。

*return_code*
値は次のとおりです。

**CM_OK** コールは正常に実行されました。

**CM_PRODUCT_SPECIFIC_ERROR**
この戻りコードの説明については、191 ページの『付録 B. 共通な戻りコード』を参照してください。

## 発行時の状態

会話がリセット状態になっています。

## 状態の変化

*return_code* が CM_OK の場合は、会話が初期化状態に変化します。その他の場合は、会話の状態は変わりません。

# Prepare_To_Receive (cmptr)

Prepare_To_Receive コールは、ローカル・プログラムの会話の状態を送信から受信に変更します。会話状態を変更する前に、このコールは次のどちらかの処理を実行します。

- Flush コールと同様に、ローカル LU の送信バッファーの内容をパートナー LU (およびプログラム) に送信する。これは、次の条件のどちらかに該当する場合です。
  - 会話の受信準備タイプが CM_PREP_TO_RECEIVE_FLUSH に設定されている。
  - 受信準備タイプが CM_PREP_TO_RECEIVE_SYNC_LEVEL に設定されており、会話の同期レベルが CM_NONE に設定されている。
- Confirm コールと同様に、ローカル LU の送信バッファーの内容と確認要求をパートナー・プログラムに送信する。これは、次の条件のどちらかに該当する場合です。

– 会話の受信準備タイプが CM_PREP_TO_RECEIVE_CONFIRM に設定されている。
– 受信準備タイプが CM_PREP_TO_RECEIVE_SYNC_LEVEL に設定されており、会話の同期レベルが CM_CONFIRM に設定されている。

受信準備タイプは Set_Prepare_To_Receive_Type コールで設定されており、同期レベルは Set_Sync_Level コールで設定されています。

このコールが正常に実行されたあとは、ローカル・プログラムはデータを受信できます。

## 関数コール

```
void cmptr (
            unsigned char CM_PTR       conversation_ID,
            CM_RETURN_CODE CM_PTR      return_code
           );
```

## Java CPI-C の関数コール

AIX、LINUX

```
public native void cmptr (
                            byte[]          conversation_ID,
                            CPICReturnCode  return_code
                          );
```

## 指定パラメーター

指定パラメーターは次のとおりです。

*conversation_ID*
このパラメーターは会話の ID です。このパラメーターの値は、Initialize_Conversation コール、Initialize_For_Incoming コール、または Accept_Conversation コールから戻されます。

## 戻りパラメーター

verb の実行後に、CS/AIX は、処理が正常に実行されたかどうかを示すパラメーターと、失敗した場合はその理由を示すパラメーターを戻します。

*return_code*
値は次のとおりです。

**CM_OK** コールは正常に実行されました。

**CM_PROGRAM_PARAMETER_CHECK**
*conversation_ID* に指定された値は無効です。

**CM_PROGRAM_STATE_CHECK**
次のいずれかの状態が発生しました。

- 会話状態が送信または送信 - 保留ではない。

- 基本会話は送信状態になっているが、プログラムが論理レコードを送信し終わっていない。

次の戻りコードの説明については、 191 ページの『付録 B. 共通な戻りコード』を参照してください。

CM_OPERATION_INCOMPLETE
CM_OPERATION_NOT_ACCEPTED
CM_PRODUCT_SPECIFIC_ERROR

会話の受信準備タイプが CM_PREP_TO_RECEIVE_CONFIRM に設定されているか、受信準備タイプが CM_PREP_TO_RECEIVE_SYNC_LEVEL に設定されており、会話の同期レベルが CM_CONFIRM に設定されている場合は、次の戻りコードが戻されることがあります。これらの戻りコードの説明については、191 ページの『付録 B. 共通な戻りコード』を参照してください。

CM_CONVERSATION_TYPE_MISMATCH
CM_DEALLOCATED_ABEND
CM_DEALLOCATED_ABEND_SVC
CM_DEALLOCATED_ABEND_TIMER
CM_PIP_NOT_SPECIFIED_CORRECTLY
CM_PROGRAM_ERROR_PURGING
CM_RESOURCE_FAILURE_NO_RETRY
CM_RESOURCE_FAILURE_RETRY
CM_SECURITY_NOT_VALID
CM_SVC_ERROR_PURGING
CM_SYNC_LVL_NOT_SUPPORTED_PGM
CM_SYNC_LVL_NOT_SUPPORTED_LU
CM_TPN_NOT_RECOGNIZED
CM_TP_NOT_AVAILABLE_NO_RETRY
CM_TP_NOT_AVAILABLE_RETRY

## 発行時の状態

会話は、送信状態でも送信 - 保留状態でも構いません。

## 状態の変化

表 17 に要約して示した状態の変化は、*return_code* パラメーターの値に基づくものです。

*表 17. Prepare_To_Receive コールによる状態の変化*

| *return_code* | **新しい状態** |
|---|---|
| CM_OK | 受信 |
| CM_PROGRAM_ERROR_PURGING | 受信 |
| CM_SVC_ERROR_PURGING | |

表 17. *Prepare_To_Receive* コールによる状態の変化 (続き)

| *return_code* | 新しい状態 |
|---|---|
| CM_CONVERSATION_TYPE_MISMATCH | リセット |
| CM_PIP_NOT_SPECIFIED_CORRECTLY | |
| CM_SECURITY_NOT_VALID | |
| CM_SYNC_LEVEL_NOT_SUPPORTED_PGM | |
| CM_SYNC_LEVEL_NOT_SUPPORTED_LU | |
| CM_TPN_NOT_RECOGNIZED | |
| CM_TP_NOT_AVAILABLE_NO_RETRY | |
| CM_TP_NOT_AVAILABLE_RETRY | |
| CM_DEALLOCATED_ABEND | リセット |
| CM_RESOURCE_FAILURE_NO_RETRY | |
| CM_RESOURCE_FAILURE_RETRY | |
| CM_DEALLOCATED_ABEND_SVC | リセット |
| CM_DEALLOCATED_ABEND_TIMER | |
| 上記以外 | 変化なし |

### 使用上の注意

Receive コールの *status_received* パラメーターを介してパートナー・プログラムが次の値のいずれかを受信するまでは、パートナー・プログラムから見た会話は送信 (または送信保留) 状態に変化しません。

- CM_SEND_RECEIVED
- CM_CONFIRM_SEND_RECEIVED を受信し、Confirmed コール、または Send_Error コールで応答する。

## Receive (cmrcv)

Receive コールは、パートナー・プログラムから現在受信可能なデータをすべて受信します。

現在受信可能なデータがなく、受信タイプ (Set_Receive_Type コールにより設定) が CM_RECEIVE_AND_WAIT に設定されている場合は、ローカル・プログラムはデータが到着するまで待機します。受信タイプが CM_RECEIVE_IMMEDIATE に設定されている場合は、ローカル・プログラムは待機しません。

WINDOWS

受信コールが非ブロッキング・モードで発行された場合 (直前の Set_Processing_Mode コールにより指定)、アプリケーションは、受信が未解決である間、以下のコールを発行することができます。

- Request_To_Send
- Send_Error
- Test_Request_to_Send_Received
- Cancel_Conversation
- Deallocate

アプリケーションが、受信コールが未解決である間に非ブロッキング・モードでこれらのコールのうちの 1 つを使用する場合、 Specify_Windows_Handle を使用して、CPI-C が非ブロッキング・コールの結果を戻せるようにする必要があります。受信に加えて他のコールも未解決である場合、Wait_For_Conversation を発行することはできません。同一の会話で複数のコールが未解決である場合、このコールの結果は未定義になります。

## プログラムでのデータの受信方法

データの受信プロセスは次のとおりです。

- ローカル・プログラムは、1 単位のデータ全体の受信を完了するまで、Receive コールを発行します。1 単位のデータの受信を完了するために、ローカル・プログラムは Receive コールを複数回発行しなければならないこともあります。*data_received* パラメーターは、データの受信が完了したかどうかを示します。

  受信されるデータは次のいずれかです。

  – マップ式会話で伝送される 1 つのデータ・レコード
  – 会話の充てん特性が CM_FILL_LL に設定されている、基本会話の中で伝送される 1 つの論理レコード
  – 充てん特性が CM_FILL_BUFFER に設定されている、基本会話の中で論理レコードの形式とは無関係に受信される 1 バッファー分のデータ

  1 単位分のデータを完全に受信し終わった時点で、ローカル・プログラムはそのデータを操作できます。
- ローカル・プログラムは、*status_received* パラメーターを介して受信する制御情報に基づいて、次にとるアクションを判断します。制御情報を受信するために、ローカル・プログラムは再び Receive コールを発行しなければならないこともあります。

会話タイプは Set_Conversation_Type コールで設定され、充てん特性は Set_Fill コールで設定されます。

## 関数コール

```
void cmrcv (
            unsigned char CM_PTR           conversation_ID,
            unsigned char CM_PTR           buffer,
            CM_INT32 CM_PTR                requested_length,
            CM_DATA_RECEIVED_TYPE CM_PTR   data_received,
            CM_INT32 CM_PTR                received_length,
            CM_STATUS_RECEIVED CM_PTR      status_received,
            CM_INT32 CM_PTR                request_to_send_received,
            CM_RETURN_CODE CM_PTR          return_code
           );
```

## Java CPI-C の関数コール

AIX、LINUX

```
public native void cmrcv (
                            byte[]                          conversation_ID,
                            byte[]                          buffer,
                            CPICLength                      requested_length,
                            CPICDataReceivedType            data_received,
                            CPICLength                      received_length,
                            CPICStatusReceived              status_received,
                            CPICControlInformationReceived  request_to_send_received,
                            CPICReturnCode                  return_code
                          );
```

## 指定パラメーター

指定パラメーターは次のとおりです。

*conversation_ID*
このパラメーターは会話の ID です。このパラメーターの値は、Initialize_Conversation コール、Initialize_For_Incoming コール、または Accept_Conversation コールから戻されます。

*requested_length*
このパラメーターは、ローカル・プログラムが受信するデータの最大バイト数を示します。

この値の範囲は 0 ～ 32,767 です。

*buffer* このパラメーターは、ローカル・プログラムが受信するデータを入れるバッファーのアドレスです。

## 戻りパラメーター

verb の実行後に、CS/AIX は次のパラメーターを戻します。

*buffer* 次の条件に該当する場合、アプリケーションのデータ・バッファーにデータが入っています。

- *data_received* パラメーターが CM_NO_DATA_RECEIVED 以外の値に設定されている。
- *return_code* パラメーターが CM_OK または CM_DEALLOCATED_NORMAL に設定されている。

*data_received*
このパラメーターは、プログラムがデータを受信したかどうかを示します。値は次のとおりです。*return_code* が CM_OK または CM_DEALLOCATED_NORMAL に設定されていない場合は、これらのコードは無関係です。

会話の充てん特性が、プログラムがデータの論理形式とは無関係にデータを受信していることを示す CM_FILL_BUFFER に設定されている場合は、CM_DATA_RECEIVED が戻されることがあります。ローカル・プログラムは、*requested_length* の指定値またはデータの終わりに達するまで、データを受信しました。

データの終わりは、次のどちらかで示されます。

- *return_code* パラメーター、*status_received* パラメーター、および *data_received* パラメーターに基づき、別の会話状態に変化する。
- エラー状態

会話の受信タイプが CM_RECEIVE_IMMEDIATE に設定されている場合は、パートナー・プログラムから到着したデータの量が少なければ、受信データは *requested_length* の値より小さいことがあります。

**CM_COMPLETE_DATA_RECEIVED**
マップ式会話では、このパラメーターは、ローカル・プログラムが 1 つのデータ・レコード全体またはデータ・レコードの最後の部分を受信したことを示します。

充てん特性が CM_FILL_LL に設定された基本会話では、この値は、ローカル・プログラムが 1 つの論理レコード全体または論理レコードの終わりを受信したことを示します。

**CM_INCOMPLETE_DATA_RECEIVED**
マップ式会話では、この値は、ローカル・プログラムが不完全なデータ・レコードを受信したことを示します。すなわち、*requested_length* パラメーターが、データ・レコードの長さより小さい値を指定した場合 (もしくは、このまだ読んでいないレコードを読み取るための 2 回目以降の Receive コールの際に、データ・レコードの残りの部分より小さい値を指定した場合) を意味します。受信されるデータの量は、*requested_length* パラメーターの値と同じです。

充てん特性が CM_FILL_LL に設定された基本会話では、この値は、ローカル・プログラムが不完全な論理レコードを受信したことを示します。受信されるデータの量は、 *requested_length* パラメーターの値と同じです。 (受信データが切り捨てられると、データの長さは *requested_length* より短くなります)。

この値を受信すると、ローカル・プログラムは通常、Receive コールを再発行してレコードの次の部分を受信します。

**CM_NO_DATA_RECEIVED**
プログラムは、データを受信しませんでした。

**注:** *return_code* パラメーターが CM_OK に設定されていれば、*status_received* パラメーターを介して状況情報を利用できることがあります。

*received_length*
このパラメーターは、ローカル・プログラムがこの Receive コールで受信したデータのバイト数を示します。*return_code* または *data_received* パラメーターに、プログラムがデータを受信しなかったことが示されている場合は、この値は無関係です。

*status_received*
このパラメーターは、会話の状態の変化を示します。*return_code* が CM_OK に設定されていない場合は、これらのコードは無関係です。値は次のとおりです。

**CM_NO_STATUS_RECEIVED**
このコールでは、会話状態の変化は受信されませんでした。

**CM_SEND_RECEIVED**
パートナー・プログラム側で、会話は受信状態になりましたが、ローカル・プログラム側では、このコールでデータが受信されていなければ送信状態、このコールでデータが受信されていれば送信 - 保留状態です。

この値を受信すると、ローカル・プログラムは通常、Send_Data コールを使用してデータの送信を開始します。

**CM_CONFIRM_DEALLOC_RECEIVED**
パートナー・プログラムが、確認要求と共に Deallocate コールを発行しました。ローカル・プログラム側では、会話は確認 - 割り振り解除状態になっています。

この値を受信すると、ローカル・プログラムは通常、Confirmed コールを発行します。

**CM_CONFIRM_RECEIVED**
パートナー・プログラムが Confirm コールを発行しました。ローカル・プログラム側では、会話は確認状態になっています。

この値を受信すると、ローカル・プログラムは通常、Confirmed コールを発行します。

**CM_CONFIRM_SEND_RECEIVED**
パートナー・プログラム側では、会話は受信状態になり、確認要求がローカル・プログラムにより受信されました。ローカル・プログラム側では、会話は確認 - 送信状態です。

プログラムは、通常、応答として Confirmed コールを発行します。Confirmed コールが正常に実行されると、ローカル・プログラム側での会話が送信状態に変化します。

*request_to_send_received*
このパラメーターは、送信要求受信インディケーターです。値は次のとおりです。

**CM_REQ_TO_SEND_RECEIVED**
パートナー・プログラムが Request_To_Send コールを発行しました。このコールは、ローカル・プログラムに、会話を受信状態に変更するよう要求します。

**CM_REQ_TO_SEND_NOT_RECEIVED**
パートナー・プログラムは Request_To_Send コールを発行していません。

*return_code* パラメーターが次のどちらかに設定されている場合は、この値は無関係です。

- CM_PROGRAM_PARAMETER_CHECK
- CM_PROGRAM_STATE_CHECK

*return_code*
値は次のとおりです。

**CM_OK** コールは正常に実行されました。

**CM_UNSUCCESSFUL**
受信タイプが CM_RECEIVE_IMMEDIATE に設定されていますが、パートナー・プログラムから現在受信可能なデータまたは状況情報がありません。

**CM_DEALLOCATED_NORMAL**
会話の割り振りが正常に解除されました。パートナー・プログラムが、会話の割り振り解除タイプを次のどちらかに設定して、Deallocate コールを発行しました。

- CM_DEALLOCATE_FLUSH
- 会話の同期レベルが CM_NONE に指定された CM_DEALLOCATE_SYNC_LEVEL

**CM_PROGRAM_PARAMETER_CHECK**
次のいずれかの状態が発生しました。

- *conversation_ID* に指定された値が有効でない。
- *requested_length* に指定された値が範囲外である。

プログラムがこの戻りコードを受け取った場合は、他の戻りパラメーターは無効になります。

**CM_PROGRAM_STATE_CHECK**
次のいずれかの状態が発生しました。

- 受信タイプが CM_RECEIVE_AND_WAIT に設定されているが、会話状態が受信、送信、または送信 - 保留ではない。
- 受信タイプが CM_RECEIVE_IMMEDIATE に設定されているが、会話状態が受信ではない。
- 基本会話が送信状態で、受信タイプが CM_RECEIVE_AND_WAIT に設定されており、プログラムが論理レコードを送信し終わっていない。

プログラムがこの戻りコードを受け取った場合は、他の戻りパラメーターは無効になります。

次の戻りコードの説明については、191 ページの『付録 B. 共通な戻りコード』を参照してください。

CM_CONVERSATION_TYPE_MISMATCH
CM_DEALLOCATED_ABEND
CM_DEALLOCATED_ABEND_SVC (基本会話のみ)
CM_DEALLOCATED_ABEND_TIMER (基本会話のみ)
CM_OPERATION_INCOMPLETE (*receive_type* = CM_RECEIVE_AND_WAIT の場合のみ)
CM_OPERATION_NOT_ACCEPTED
CM_PIP_NOT_SPECIFIED_CORRECTLY
CM_PRODUCT_SPECIFIC_ERROR
CM_PROGRAM_ERROR_NO_TRUNC
CM_PROGRAM_ERROR_PURGING
CM_PROGRAM_ERROR_TRUNC (基本会話のみ)
CM_RESOURCE_FAILURE_NO_RETRY
CM_RESOURCE_FAILURE_RETRY

CM_SECURITY_NOT_VALID
CM_SYNC_LVL_NOT_SUPPORTED_PGM
CM_SYNC_LVL_NOT_SUPPORTED_LU
CM_TP_NOT_AVAILABLE_NO_RETRY
CM_TP_NOT_AVAILABLE_RETRY
CM_TPN_NOT_RECOGNIZED
CM_SVC_ERROR_NO_TRUNC (基本会話のみ)
CM_SVC_ERROR_PURGING (基本会話のみ)
CM_SVC_ERROR_TRUNC (基本会話のみ)

## 発行時の状態

会話は、受信状態、送信状態、送信 - 保留状態のいずれでも構いません。

*receive_type* が CM_RECEIVE_IMMEDIATE に設定されている場合は、会話は受信状態でなければなりません。

WINDOWS

アプリケーションが、非ブロッキング・モードで受信コールを正常に発行した場合、会話は 2 度、状態を変更します。コールが最初に戻されると、会話は、Pending-Post 状態に変更されます。CPI-C がコール処理の結果を戻した後、会話の状態変更は、以下に述べるとおりです。

### 送信状態または送信 - 保留状態でのコールの発行

会話が送信または送信 - 保留状態の間に Receive コールを発行すると、ローカル LU は、その送信バッファー内の情報と送信インディケーターをパートナー・プログラムに送信します。*data_received* パラメーターおよび *status_received* パラメーターに基づき、ローカル・プログラムから見た会話が受信状態に変化する場合があります。詳しくは、『状態の変化』を参照してください。

## 状態の変化

新しい会話状態は、次の要因によって決まります。

- プログラムがコールを発行したときの会話の状態
- *return_code* パラメーター
- *data_received* パラメーターおよび *status_received* パラメーター

### 受信状態でコールを発行した場合

111 ページの表 18 に、会話が受信状態であり *return_code* が CM_OK に設定されているときに Receive コールを発行した場合の状態の変化を示します。

*表 18. 受信状態で Receive コールを発行したときの状態の変化*

| *data_received* | *status_received* | **新しい状態** |
|---|---|---|
| CM_DATA_RECEIVED<br>CM_COMPLETE_DATA_RECEIVED<br>CM_INCOMPLETE_DATA_RECEIVED | CM_NO_STATUS_RECEIVED | 変化なし |
| CM_DATA_RECEIVED<br>CM_COMPLETE_DATA_RECEIVED | CM_SEND_RECEIVED | 送信 - 保留 |
| CM_NO_DATA_RECEIVED | CM_SEND_RECEIVED | 送信 |

*return_code* が CM_UNSUCCESSFUL に設定されている場合、つまり *receive_type* が CM_RECEIVE_IMMEDIATE に設定され受信可能なデータがないことが示されている場合は、状態の変化は起こりません。

## 送信状態でコールを発行した場合

表 19 に、会話が送信状態であり *return_code* が CM_OK に設定されているときに Receive コールを発行した場合の状態の変化を示します。

*表 19. 送信状態で Receive コールを発行したときの状態の変化*

| *data_received* | *status_received* | **新しい状態** |
|---|---|---|
| CM_DATA_RECEIVED<br>CM_COMPLETE_DATA_RECEIVED<br>CM_INCOMPLETE_DATA_RECEIVED | CM_NO_STATUS_RECEIVED | 受信 |
| CM_DATA_RECEIVED<br>CM_COMPLETE_DATA_RECEIVED | CM_SEND_RECEIVED | 送信 - 保留 |
| CM_NO_DATA_RECEIVED | CM_SEND_RECEIVED | 変化なし |

## 送信 - 保留状態でコールを発行した場合

表 20 に、会話が送信 - 保留状態であり *return_code* が CM_OK に設定されているときに Receive コールを発行した場合の状態の変化を示します。

*表 20. 送信 - 保留状態で Receive コールを発行したときの状態の変化*

| *data_received* | *status_received* | **新しい状態** |
|---|---|---|
| CM_DATA_RECEIVED<br>CM_COMPLETE_DATA_RECEIVED<br>CM_INCOMPLETE_DATA_RECEIVED | CM_NO_STATUS_RECEIVED | 受信 |
| CM_DATA_RECEIVED<br>CM_COMPLETE_DATA_RECEIVED | CM_SEND_RECEIVED | 変化なし |
| CM_NO_DATA_RECEIVED | CM_SEND_RECEIVED | 送信 |

## 任意の許可状態でコールを発行した場合

次の各項では、任意の許可状態で Receive コールを発行した場合に起こり得る状態の変化について簡単に示します。

### 確認処理

次の条件のもとでは次の状態変化が起こります。

- *return_code* が CM_OK に設定されている。

- *data_received* パラメーターが CM_DATA_RECEIVED、CM_COMPLETE_DATA_RECEIVED、または CM_NO_DATA_RECEIVED に設定されている。
- *status_received* パラメーターが、表 21 に示すように、確認状態への変更を示している。

*表 21. 任意の許可状態で Receive コールを発行したときの状態の変化*

| *status_received* | 新しい状態 |
|---|---|
| CM_CONFIRM_DEALLOC_RECEIVED | 確認 - 割り振り解除 |
| CM_CONFIRM_SEND_RECEIVED | 確認 - 送信 |
| CM_CONFIRM_RECEIVED | 確認 |

## 正常な割り振り解除

*return_code* パラメーターが CM_DEALLOCATED_NORMAL に設定されていれば、会話はリセット状態に変わります。

## アベンド

次のアベンド条件は、*return_code* パラメーターにより示されるもので、これによって会話はリセット状態に変わります。

CM_CONVERSATION_TYPE_MISMATCH
CM_PIP_NOT_SPECIFIED_CORRECTLY
CM_SECURITY_NOT_VALID
CM_SYNC_LVL_NOT_SUPPORTED_PGM
CM_SYNC_LVL_NOT_SUPPORTED_LU
CM_TPN_NOT_RECOGNIZED
CM_TP_NOT_AVAILABLE_NO_RETRY
CM_TP_NOT_AVAILABLE_RETRY
CM_DEALLOCATED_ABEND
CM_DEALLOCATED_ABEND_SVC
CM_DEALLOCATED_ABEND_TIMER
CM_SVC_ERROR_TRUNC
CM_RESOURCE_FAILURE_NO_RETRY
CM_RESOURCE_FAILURE_RETRY

## エラー

データ伝送エラーが検出されると、表 22 に示す状態の変化が起こることがあります。(この状態の変化は、次のいずれかの戻りコードにより示されます。CM_PROGRAM_ERROR_PURGING、CM_PROGRAM_ERROR_NO_TRUNC、CM_SVC_ERROR_PURGING、または CM_SVC_ERROR_NO_TRUNC。)

*表 22. データ伝送エラーによる状態の変化*

| *return_code* | 元の状態 | 新しい状態 |
|---|---|---|
| CM_PROGRAM_ERROR_PURGING | 受信 | 変化なし |
| CM_PROGRAM_ERROR_NO_TRUNC | 受信 | 変化なし |
| CM_SVC_ERROR_PURGING | 送信 | 受信 |
| CM_SVC_ERROR_NO_TRUNC | 送信 - 保留 | 受信 |

## 使用上の注意

Receive コールを使用する場合の追加注意事項について、次の各項で説明します。

### レコードの切り捨て

パートナー・プログラムが論理レコードを切り捨てた場合は、ローカル・プログラムは、次の Receive コールの *return_code* パラメーターを介して切り捨ての通知を受け取ります。

### Requested_Length パラメーターをゼロに設定した場合

パラメーターが、*requested_length* を 0 (ゼロ) に設定して Receive コールを出した場合、そのコールは通常どおり実行されます。

ただし、*data_received* パラメーターと *status_received* パラメーターは同じ Receive コールでは設定されません (この状況に対する唯一の例外は、次に説明するように、マップ式会話でヌル・レコードが送信された場合です)。

マップ式会話では、パートナー・プログラムからのデータが使用可能であり、*data_received* パラメーターが CM_INCOMPLETE_DATA_RECEIVED に設定されます。ヌル・レコードが使用可能な場合 (パートナー・プログラムが発行した Send_Data コールの *send_length* が 0 に設定されている場合) は、*data_received* パラメーターが CM_COMPLETE_DATA_RECEIVED に設定され、*received_length* パラメーターが 0 (ゼロ) に設定されます。

データが使用可能な基本会話で、充てん特性が CM_FILL_LL に設定されている場合は、*data_received* パラメーターは CM_INCOMPLETE_DATA_RECEIVED に設定されます。充てん特性が CM_FILL_BUFFER に設定されている場合は、*data_received* は CM_DATA_RECEIVED に設定されます。

### ストリング変換

LU は、受信データ・ストリングをバッファーに入れる前に、その受信データ・ストリングに対して、自動的に EBCDIC と ASCII の間の変換を実行することはありません。

リモート・プログラムがデータを EBCDIC で送信した場合、ローカル・プログラムは、Convert_Incoming コールを使用して、受信データを ASCII に変換できます。

WINDOWS

ローカル・プログラムは、CSV CONVERT verb を使用して、受信データを ASCII に変換することもできます。詳しくは、「*Communications Server for AIX Common Service Verb プログラマーズ・ガイド*」を参照してください。

# Release_Local_TP_Name (cmrltp)

AIX、LINUX

Release_Local_TP_Name コールは、着呼 Allocate の TP 名要求をこれ以上受け入れないことを示すために、プログラムが発行します。この TP 名は、37 ページの『ローカル TP 名の指定』に示した方法の 1 つを使用して指定したものです。

## 関数コール

```
void cmrltp (
             unsigned char CM_PTR      TP_name,
             CM_INT32 CM_PTR           TP_name_length,
             CM_RETURN_CODE CM_PTR     return_code
            );
```

## Java CPI-C の関数コール

```
public native void cmrltp (
                             byte[]          TP_name,
                             CPICLength      TP_name_length,
                             CPICReturnCode  return_code
                           );
```

## 指定パラメーター

指定パラメーターは次のとおりです。

*TP_name*
このパラメーターは、TP 名の開始アドレスを示します。これは、プログラムが前に Specify_Local_TP_Name コールに指定した TP 名である必要があります。

*TP_name_length*
このパラメーターは、名前の長さ (1〜64 文字) を示します。

## 戻りパラメーター

verb の実行後に、CS/AIX は、処理が正常に実行されたかどうかを示すパラメーターと、失敗した場合はその理由を示すパラメーターを戻します。

*return_code*
値は次のとおりです。

**CM_OK** コールは正常に実行されました。

**CM_PROGRAM_PARAMETER_CHECK**
次のいずれかの状態が発生しました。

- *TP_name* に指定された値が、このプログラムに関連付けられた TP 名ではない。
- *TP_name_length* に指定された値が範囲外である。

**CM_PRODUCT_SPECIFIC_ERROR**
この戻りコードの説明については、191 ページの『付録 B. 共通な戻りコード』を参照してください。

### 発行時の状態

このコールは、会話には関連付けられていません。

### 状態の変化

状態には変化はありません。

### 使用上の注意

*return_code* が CM_OK でない場合は、プログラムに関連付けられた名前は変更されていません。

このコールの発行時に、未解決の Accept_Incoming コールがあった場合は、このコールは、指定された名前についての着呼 Allocate を受け付ける可能性があります。ただし、後続の Accept_Conversation コール、または Accept_Incoming コールは、この名前についての着呼 Allocate を受け付けません。

APPCTPN 環境変数により指定された名前がある場合はそれも含めて、プログラムが、その TP 名をすべて解除した場合は、プログラムは、新しいローカル TP 名をまず最初に指定しない限り、Accept_Conversation コール、または Accept_Incoming コールをこれ以上発行することはできません。詳しくは、37 ページの『ローカル TP 名の指定』を参照してください。

## Request_To_Send (cmrts)

Request_To_Send コールは、ローカル・プログラムがデータの送信を要求していることを、パートナー・プログラムに通知します。

### パートナー・プログラムのアクション

この要求に応答して、パートナー・プログラムは、次のコールのいずれかを発行して会話を受信状態に変更できます。

- Receive。*receive_type* を CM_RECEIVE_AND_WAIT に設定して発行します。
- Prepare_To_Receive
- Send_Data。*send_type* を CM_SEND_AND_PREP_TO_RECEIVE に設定して発行します。

パートナー・プログラムは、送信要求を無視することもできます。

### ローカル・プログラムがデータを送信できる時点

後続の Receive コールの *status_received* パラメーターを介してローカル・プログラムが次のどちらかの値を受信すると、ローカル・プログラムから見た会話は送信状態に変化します。

- CM_SEND_RECEIVED
- CM_CONFIRM_SEND_RECEIVED および Confirmed コールによる応答

## 関数コール

```
void cmrts (
            unsigned char CM_PTR       conversation_ID,
            CM_RETURN_CODE CM_PTR      return_code
           );
```

## Java CPI-C の関数コール

AIX、LINUX

```
public native void cmrts (
                           byte[]          conversation_ID,
                           CPICReturnCode  return_code
                         );
```

## 指定パラメーター

指定パラメーターは次のとおりです。

*conversation_ID*
このパラメーターは会話の ID です。このパラメーターの値は、Initialize_Conversation コール、Initialize_For_Incoming コール、または Accept_Conversation コールから戻されます。

## 戻りパラメーター

verb の実行後に、CS/AIX は、処理が正常に実行されたかどうかを示すパラメーターと、失敗した場合はその理由を示すパラメーターを戻します。

*return_code*
値は次のとおりです。

**CM_OK** コールは正常に実行されました。

**CM_PROGRAM_PARAMETER_CHECK**
*conversation_ID* に指定された値は無効です。

**CM_PROGRAM_STATE_CHECK**
会話が受信、送信、送信 - 保留、確認、確認 - 送信、または確認 - 割り振り解除の状態になっていません。

次の戻りコードの説明については、191 ページの『付録 B. 共通な戻りコード』を参照してください。
CM_OPERATION_INCOMPLETE
CM_OPERATION_NOT_ACCEPTED
CM_PRODUCT_SPECIFIC_ERROR

## 発行時の状態

会話は、以下の状態のいずれであっても構いません: 受信、送信、送信 - 保留、確認、確認 - 送信、確認 - 割り振り解除、保留 - 通知。

### 状態の変化

状態には変化はありません。

### 使用上の注意

パートナー・プログラムは、次のコールの *request_to_send_received* パラメーターを介して、送信要求の通知を受信します。

- Confirmed
- Receive
- Send_Data
- Send_Error
- Test_Request_to_Send_Received

Request-to-send 通知は、即時にパートナー・プログラムに送信されます。CPI-C は、送信バッファーが満ぱいになったりフラッシュされるまで待機することはありません。したがって、送信要求の通知は順序どおりには到着しないことがあります。たとえば、ローカル・プログラムが送信状態で、Prepare_To_Receive コールに続けて Request_To_Send コールを発行した場合、受信状態のパートナー・プログラムは、送信通知の前に送信要求の通知を受け取る可能性があります。このため、送信要求の通知を Receive コールを介してプログラムに報告するように設定できます。

送信要求の通知を受信すると、パートナー LU は、パートナー・プログラムが *request_to_send_received* パラメーターを戻すコールを発行するまで、その送信要求の通知を保存しておきます。LU は、1 つの会話について 1 つしか送信要求の通知を保存しません。したがって、ローカル・プログラムが発行した Request_To_Send がすべてパートナー・プログラムに通知されるとは限りません。

## Send_Data (cmsend)

Send_Data コールは、パートナー・プログラムに伝送するためのデータをローカル LU の送信バッファーに入れます。

ローカル LU の送信バッファーに入れられたデータは、次のどちらかの状態が発生したときに、パートナー LU (およびパートナー・プログラム) に伝送されます。

- 送信バッファーが満ぱいになる
- ローカル・プログラムが、Flush コール、Confirm コール、または Deallocate コール、または LU の送信バッファーをフラッシュするその他のコールを発行した (Set_Send_Type コールにより設定される送信タイプには、フラッシュ機能が備わっているものもあります。)

送信されるデータは、次のどちらかです。

- マップ式会話の完全なデータ・レコード。完全なデータ・レコードは、*send_length* パラメーターにより指定された長さのストリングです。
- 基本会話の完全な論理レコードまたは論理レコードの一部。完全な論理レコードの長さは LL 値により決定されます (送信されるデータのストリングの途中で、1 つの論理レコードが終わって新しい論理レコードが始まる場合もあります)。

## 関数コール

```
void cmsend (
             unsigned char CM_PTR                conversation_ID,
             unsigned char CM_PTR                buffer,
             CM_INT32 CM_PTR                     send_length,
             CM_Request_to_Send_Received CM_PTR  request_to_send_received,
             CM_RETURN_CODE CM_PTR               return_code
            );
```

## Java CPI-C の関数コール

AIX、LINUX

```
public native void cmsend (
                           byte[]                          conversation_ID,
                           byte[]                          buffer,
                           CPICLength                      buffer_length,
                           CPICControlInformationReceived  request_to_send_received,
                           CPICReturnCode                  return_code
                          );
```

## 指定パラメーター

指定パラメーターは次のとおりです。

*conversation_ID*
このパラメーターは会話の ID です。このパラメーターの値は、Initialize_Conversation コール、Initialize_For_Incoming コール、または Accept_Conversation コールから戻されます。

*buffer* このパラメーターは、ローカル LU の送信バッファーに入れるデータ収容用のバッファーのアドレスを指定します。

*send_length*
このパラメーターは、ローカル LU の送信バッファーに入れるデータのバイト数です。

この値の範囲は 0 ～ 32,767 です。

マップ式会話の場合は、*send_length* が 0 に設定されていれば、パートナー・プログラムにヌル・データ・レコードが送信されます。

基本会話の場合は、*send_length* が 0 (ゼロ) に設定されていれば、データは送信されません。バッファー・パラメーターは無視されます。ただし、その他のパラメーターは有効です。

## 戻りパラメーター

verb の実行後に、CS/AIX は次のパラメーターを戻します。

*request_to_send_received*
このパラメーターは、送信要求受信インディケーターです。値は次のとおりです。

**CM_REQ_TO_SEND_RECEIVED**
パートナー・プログラムが Request_To_Send コールを発行しました。このコールは、ローカル・プログラムに、会話を受信状態に変更するよう要求します。

**CM_REQ_TO_SEND_NOT_RECEIVED**
パートナー・プログラムは Request_To_Send コールを発行していません。

*return_code* パラメーターが CM_PROGRAM_PARAMETER_CHECK または CM_PROGRAM_STATE_CHECK に設定されている場合は、この値は無関係です。

*return_code*
値は次のとおりです。

**CM_OK** コールは正常に実行されました。

**CM_PROGRAM_PARAMETER_CHECK**
次のいずれかの状態が発生しました。

- *conversation_ID* に指定された値が有効でない。
- *send_length* に指定された値が範囲外である。
- 基本会話で、*buffer* パラメーターの最初の 2 バイトに有効でない論理レコード長が入っている (0x0000、0x0001、0x8000、または 0x8001)。

**CM_PROGRAM_STATE_CHECK**
次のいずれかの状態が発生しました。

- 会話状態が送信または送信 - 保留ではない。
- 基本会話が送信状態になっており、送信タイプが CM_SEND_AND_CONFIRM、CM_SEND_AND_DEALLOCATE、または CM_SEND_AND_PREP_TO_RECEIVE に設定されているが、データが論理レコード境界で終了していない。Send_Data を論理レコードの途中で発行できるのは、送信タイプが CM_SEND_AND_DEALLOCATE に、割り振り解除タイプが CM_DEALLOCATE_ABEND に設定されているときだけです。

次の戻りコードの説明については、191 ページの『付録 B. 共通な戻りコード』を参照してください。

CM_CONVERSATION_TYPE_MISMATCH
CM_DEALLOCATED_ABEND
CM_DEALLOCATED_ABEND_SVC
CM_DEALLOCATED_ABEND_TIMER
CM_OPERATION_INCOMPLETE
CM_OPERATION_NOT_ACCEPTED
CM_PIP_NOT_SPECIFIED_CORRECTLY
CM_PRODUCT_SPECIFIC_ERROR
CM_PROGRAM_ERROR_PURGING
CM_RESOURCE_FAILURE_NO_RETRY
CM_RESOURCE_FAILURE_RETRY
CM_SECURITY_NOT_VALID
CM_SVC_ERROR_PURGING

CM_SYNC_LVL_NOT_SUPPORTED_PGM
CM_SYNC_LVL_NOT_SUPPORTED_LU
CM_TP_NOT_AVAILABLE_NO_RETRY
CM_TP_NOT_AVAILABLE_RETRY
CM_TPN_NOT_RECOGNIZED

## 発行時の状態

プログラムがこのコールを発行するとき、会話は送信状態または送信 - 保留状態になっていなければなりません。

## 状態の変化

*return_code* パラメーターが CM_OK に設定されている場合、新しい会話状態は、表 23 に示すように、*send_type* パラメーターに応じて異なります。

*表 23. Send_Data コールによる状態の変化*

| *send_type* | **新しい状態** |
|---|---|
| CM_BUFFER_DATA | 送信 |
| CM_SEND_AND_FLUSH | 送信 |
| CM_SEND_AND_CONFIRM | 送信 |
| CM_SEND_AND_PREP_TO_RECEIVE | 受信 |
| CM_SEND_AND_DEALLOCATE | リセット |

*return_code* 値が CM_PROGRAM_ERROR_PURGING または CM_SVC_ERROR_PURGING の場合は、会話は受信状態に変わります。OK 以外のその他の値のときは、会話はリセット状態に変わります。

## 使用上の注意

LU は、送信するデータ・ストリングに対して、自動的に ASCII と EBCDIC の間の変換を実行することはありません。

リモート・プログラムに EBCDIC でデータを送信する必要がある場合、ローカル・プログラムは Convert_Outgoing コールを使用して、送信前にデータを EBCDIC に変換できます。

WINDOWS

ローカル・プログラムは、CSV CONVERT verb を使用して、送信前にデータを EBCDIC に変換することもできます。詳しくは、「*Communications Server for AIX Common Service Verb プログラマーズ・ガイド*」を参照してください。

# Send_Error (cmserr)

Send_Error コールは、ローカル・プログラムがアプリケーション・レベルのエラーを検出したことを、パートナー・プログラムに通知します。ローカル・プログラムは、Send_Error コールを使用して、受信データにエラーが見つかったことをパートナー・プログラムに通知したり、確認要求をリジェクトしたり、または送信中の不完全な論理レコードを切り捨てたりすることができます。

Send_Error コールは、ローカル LU の送信バッファーをフラッシュし、送信バッファーの内容とエラー通知をパートナー・プログラムに送信します。

エラー通知は、次の *return_code* 値の 1 つとして、パートナーに送信されます。

- CM_PROGRAM_ERROR_TRUNC
- CM_PROGRAM_ERROR_NO_TRUNC
- CM_PROGRAM_ERROR_PURGING

このコールが正常に実行されると、会話は、ローカル・プログラムにとっては送信状態になり、パートナー・プログラム側では受信状態になります。

## 関数コール

```
void cmserr (
             unsigned char CM_PTR                conversation_ID,
             CM_Request_to_Send_Received CM_PTR  request_to_send_received,
             CM_RETURN_CODE CM_PTR               return_code
            );
```

## Java CPI-C の関数コール

AIX、LINUX

```
public native void cmserr (
                           byte[]                          conversation_ID,
                           CPICControlInformationReceived  request_to_send_received,
                           CPICReturnCode                  return_code
                          );
```

## 指定パラメーター

指定パラメーターは次のとおりです。

*conversation_ID*
このパラメーターは会話の ID です。このパラメーターの値は、Initialize_Conversation コール、Initialize_For_Incoming コール、または Accept_Conversation コールから戻されます。

## 戻りパラメーター

verb の実行後に、CS/AIX は、処理が正常に実行されたかどうかを示すパラメーターと、失敗した場合はその理由を示すパラメーターを戻します。

*request_to_send_received*

このパラメーターは、送信要求受信インディケーターです。値は次のとおりです。

**CM_REQ_TO_SEND_RECEIVED**
パートナー・プログラムが Request_To_Send コールを発行しました。このコールは、ローカル・プログラムに、会話を受信状態に変更するよう要求します。

**CM_REQ_TO_SEND_NOT_RECEIVED**
パートナー・プログラムは Request_To_Send コールを発行していません。

*return_code* が CM_PROGRAM_PARAMETER_CHECK または CM_STATE_CHECK に設定されている場合は、この値は無関係です。

*return_code*

戻りコードは、コールが発行されたときの会話の状態に応じて異なります。

**送信状態**

会話が送信状態のときにプログラムがコールを発行した場合は、次の戻りコードが戻されます。

**CM_OK** コールは正常に実行されました。

次の戻りコードの説明については、191 ページの『付録 B. 共通な戻りコード』を参照してください。

CM_CONVERSATION_TYPE_MISMATCH
CM_DEALLOCATED_ABEND
CM_DEALLOCATED_ABEND_SVC
CM_DEALLOCATED_ABEND_TIMER
CM_OPERATION_INCOMPLETE
CM_OPERATION_NOT_ACCEPTED
CM_PIP_NOT_SPECIFIED_CORRECTLY
CM_PRODUCT_SPECIFIC_ERROR
CM_PROGRAM_ERROR_PURGING
CM_RESOURCE_FAILURE_NO_RETRY
CM_RESOURCE_FAILURE_RETRY
CM_SECURITY_NOT_VALID
CM_SVC_ERROR_PURGING
CM_SYNC_LVL_NOT_SUPPORTED_PGM
CM_SYNC_LVL_NOT_SUPPORTED_LU
CM_TP_NOT_AVAILABLE_NO_RETRY
CM_TP_NOT_AVAILABLE_RETRY
CM_TPN_NOT_RECOGNIZED

**受信状態または保留 - 通知状態**

受信状態または保留 - 通知状態でコールが発行される場合、以下の戻りコードが考えられます。

**CM_OK** 受信状態または保留 - 通知状態で Send_Error コールが発行された場合には着呼情報は消去されるため、以下の戻りコードの代わりに CM_OK が生成されます。

CM_PROGRAM_ERROR_NO_TRUNC
CM_PROGRAM_ERROR_PURGING
CM_SVC_ERROR_NO_TRUNC
CM_SVC_ERROR_PURGING
CM_PROGRAM_ERROR_TRUNC
CM_SVC_ERROR_TRUNC

**共通な戻りコード**

次の戻りコードの説明については、 191 ページの『付録 B. 共通な戻りコード』を参照してください。

CM_OPERATION_INCOMPLETE
CM_OPERATION_NOT_ACCEPTED
CM_PRODUCT_SPECIFIC_ERROR
CM_RESOURCE_FAILURE_NO_RETRY
CM_RESOURCE_FAILURE_RETRY

**CM_DEALLOCATED_NORMAL**

受信状態または保留 - 通知状態で Send_Error コールが発行された場合には着呼情報は消去されるため、以下の戻りコードの代わりに CM_DEALLOCATED_NORMAL が生成されます。

CM_CONVERSATION_TYPE_MISMATCH
CM_DEALLOCATED_ABEND
CM_DEALLOCATED_ABEND_SVC
CM_DEALLOCATED_ABEND_TIMER
CM_PIP_NOT_SPECIFIED_CORRECTLY
CM_SECURITY_NOT_VALID
CM_SYNC_LVL_NOT_SUPPORTED_PGM
CM_SYNC_LVL_NOT_SUPPORTED_LU
CM_TPN_NOT_RECOGNIZED
CM_TP_NOT_AVAILABLE_NO_RETRY
CM_TP_NOT_AVAILABLE_RETRY

**送信 - 保留状態**

送信 - 保留状態でコールが発行された場合は、次の戻りコードが戻されます。

**CM_OK** コールは正常に実行されました。

次の戻りコードの説明については、 191 ページの『付録 B. 共通な戻りコード』を参照してください。

CM_OPERATION_INCOMPLETE
CM_OPERATION_NOT_ACCEPTED
CM_DEALLOCATED_ABEND
CM_DEALLOCATED_ABEND_SVC
CM_DEALLOCATED_ABEND_TIMER
CM_PRODUCT_SPECIFIC_ERROR
CM_PROGRAM_ERROR_PURGING
CM_RESOURCE_FAILURE_NO_RETRY
CM_RESOURCE_FAILURE_RETRY
CM_SVC_ERROR_PURGING

**確認、確認 - 送信、あるいは確認 - 割り振り解除状態**

確認、確認 - 送信、または確認 - 割り振り解除の状態でコールが発行された場合は、次の戻りコードが戻されます。

**CM_OK** コールは正常に実行されました。

次の戻りコードの説明については、 191 ページの『付録 B. 共通な戻りコード』を参照してください。

CM_OPERATION_INCOMPLETE
CM_OPERATION_NOT_ACCEPTED
CM_PRODUCT_SPECIFIC_ERROR
CM_RESOURCE_FAILURE_NO_RETRY
CM_RESOURCE_FAILURE_RETRY

**その他の状態**

会話がリセット、初期化、または初期化 - 着呼の状態のときに Send_Error コールを発行するのは、規則違反です。次の戻りコードが戻されます。

**CM_OPERATION_NOT_ACCEPTED**
191 ページの『付録 B. 共通な戻りコード』を参照してください。

**CM_PROGRAM_PARAMETER_CHECK**
*conversation_ID* に指定された値は無効です。

**CM_PROGRAM_STATE_CHECK**
会話状態が、送信、受信、確認、確認 - 送信、確認 - 割り振り解除、または送信 - 保留になっていません。

## 発行時の状態

会話は、初期化、初期化 - 着呼、またはリセット以外ならどの状態でも構いません。

## 状態の変化

新しい状態は、*return_code* パラメーターによって決まります。表 24 に、状態の変化が起こる場合を要約して示します。

*表 24. Send_Error コールによる状態の変化*

| *return_code* | **新しい状態** |
|---|---|
| CM_OK | 送信 |
| CM_CONVERSATION_TYPE_MISMATCH | リセット |
| CM_PIP_NOT_SPECIFIED_CORRECTLY | |
| CM_SECURITY_NOT_VALID | |
| CM_SYNC_LEVEL_NOT_SUPPORTED_PGM | |
| CM_SYNC_LEVEL_NOT_SUPPORTED_LU | |
| CM_TPN_NOT_RECOGNIZED | |
| CM_TP_NOT_AVAILABLE_NO_RETRY | |
| CM_TP_NOT_AVAILABLE_RETRY | |
| CM_RESOURCE_FAILURE_RETRY | リセット |
| CM_RESOURCE_FAILURE_NO_RETRY | |

表 24. *Send_Error* コールによる状態の変化 (続き)

| *return_code* | 新しい状態 |
|---|---|
| CM_DEALLOCATED_ABEND | リセット |
| CM_DEALLOCATED_ABEND_SVC | |
| CM_DEALLOCATED_ABEND_TIMER | |
| CM_DEALLOCATED_NORMAL | リセット |
| CM_PROGRAM_ERROR_PURGING | 受信 |
| CM_SVC_ERROR_PURGING | |
| 上記以外 | 変化なし |

## 使用上の注意

Send_Error コールを使用する場合の追加注意事項について、次の各項で説明します。

### ログ・データの送信

基本会話では、ローカル・プログラムは Set_Log_Data コールを使用して、パートナー LU に送信するエラー・ログ・データを指定することができます。基本会話のログ・データ長の特性が 0 (ゼロ) より大きい値の場合、LU はデータを形式設定して送信バッファーに入れます。

Send_Error コールが完了すると、ログ・データ長は 0 (ゼロ) に設定され、ログ・データはヌルに設定されます。

### 除去データ

プログラムが Send_Error コールを発行するときに、会話が受信状態または保留 - 通知状態である場合、着呼データは CPI-C により除去されます。除去されるデータは次のとおりです。

- Send_Data コールにより送信されたデータ
- 確認要求
- 割り振り解除要求。これは、会話の割り振り解除タイプが CM_DEALLOCATE_CONFIRM または CM_DEALLOCATE_SYNC_LEVEL に設定され、同期レベルが CM_CONFIRM に設定されている場合です。

着呼送信要求のインディケーターを CPI-C が除去することはありません。

### 送信 - 保留状態

会話が送信 - 保留状態のときは、ローカル・プログラムは Set_Error_Direction コールを発行して、報告されたエラーの原因が受信データにあるのか、またはデータの受信に成功したあとのローカル・プログラムの処理にあるのかを判別できます。

# Set_Conversation_Context (cmsctx)

AIX、LINUX

Set_Conversation_Context コールは、プログラムの現行コンテキストを、Extract_Conversation_Context コールで直前に戻された値に設定します。これによって、プログラムは、前の会話と同じコンテキストを使用して新しい会話を開始することができます。

会話コンテキストの詳細については、 13 ページの『複数会話』を参照してください。

## 関数コール

```
void cmsctx (
             unsigned char CM_PTR      context_ID,
             CM_INT32 CM_PTR           context_ID_length,
             CM_RETURN_CODE CM_PTR     return_code
            );
```

## Java CPI-C の関数コール

```
public native void cmsctx (
                             byte[]          context_ID,
                             CPICLength      context_ID_length,
                             CPICReturnCode  return_code
                           );
```

**注:** このコールは Java CPI-C の標準仕様ではありません。他の Java CPI-C インプリメンテーションではサポートされていません。

## 指定パラメーター

指定パラメーターは次のとおりです。

*context_ID*
このパラメーターは必須コンテキストを示します。

*context_ID_length*
このパラメーターは *context_ID* の長さ (1〜32 バイト) を示します。

## 戻りパラメーター

verb の実行後に、CS/AIX は、処理が正常に実行されたかどうかを示すパラメーターと、失敗した場合はその理由を示すパラメーターを戻します。

*return_code*
値は次のとおりです。

**CM_OK** コールは正常に実行されました。

**CM_PROGRAM_PARAMETER_CHECK**
この戻りコードは、次のいずれかの状態が発生したことを示します。

- *context_ID* に指定された値が、プログラムのどの現行会話のコンテキストでもない、またはプログラムの最新の会話のコンテキストではない。
- *context_ID_length* に指定された値が有効でない。

**CM_PRODUCT_SPECIFIC_ERROR**
この戻りコードの説明については、191 ページの『付録 B. 共通な戻りコード』を参照してください。

## 発行時の状態

会話は、リセット以外であればどの状態にあっても構いません。

## 状態の変化

状態には変化はありません。

## 使用上の注意

アプリケーションで Set_Conversation_Context を使用するのは、次のような状況のときです。

- アプリケーションで複数の会話を取り扱っていて、既存の会話と同じコンテキストを使用する新規会話を割り振る必要がある場合。
- 新規コンテキストを割り当てる CPI-C コールが非ブロッキング・モードで完了する場合。たとえば、Accept_Incoming が即時に完了し *return_code* CM_OK を戻した場合は、プログラムの現行コンテキストは新規会話のコンテキストに設定されます。しかし、Accept_Incoming が CM_OPERATION_INCOMPLETE を戻した場合は、後続の Wait_For_Conversation から Accept_Incoming の結果が戻されても、プログラムの現行コンテキストは変更されません。プログラムは、Extract_Conversation_Context および Set_Conversation_Context を使用して、現行コンテキストを正しい値に設定する必要があります。

# Set_Conversation_Security_Password (cmscsp)

Set_Conversation_Security_Password コールは、呼び出し対象プログラムへのアクセスに必要なパスワードを指定するために、呼び出し側プログラムが発行します。このコールは、会話セキュリティー・タイプが CM_SECURITY_PROGRAM または CM_SECURITY_PROGRAM_STRONG (AIX または Linux システム)、あるいは XC_SECURITY_PROGRAM (Windows システム) である場合に限り、会話上で有効です。指定したパスワードは、Initialize_Conversation コールにより指定されたサイド情報から得られる初期パスワードを変更します。Allocate コールの発行後にこのコールを発行することはできません。

## 関数コール

```
void cmscsp (
             unsigned char CM_PTR      conversation_ID,
             unsigned char CM_PTR      security_password,
             CM_INT32 CM_PTR           security_password_length,
             CM_RETURN_CODE CM_PTR     return_code
            );
```

## Java CPI-C の関数コール

AIX、LINUX

```
public native void cmscsp (
                            byte[]          conversation_ID,
                            byte[]          security_password,
                            CPICLength      security_password_length,
                            CPICReturnCode  return_code
                           );
```

## 指定パラメーター

指定パラメーターは次のとおりです。

*conversation_ID*

このパラメーターは会話の ID です。このパラメーターの値は、Initialize_Conversation コールから戻されます。

*security_password*

このパラメーターは、パートナー・プログラムにアクセスするのに必要なパスワードを指定します。この値の範囲は、1 から 10 文字 (AIX または Linux システム)、または 1 から 8 文字 (Windows システム) であり、大/小文字の区別が必要です。この値は、パートナー・プログラムのために構成されているユーザー ID のパスワードに一致していなければなりません。

使用できる文字は次のとおりです。

- 大文字および小文字の英字
- 数字 0〜9
- 特殊文字 $、#、@、および . (ピリオド)

*security_password_length*

このパラメーターは、*security_password* の長さを示します。

この値の範囲は、1 から 10 文字 (AIX または Linux システム)、または 1 から 8 文字 (Windows システム) です。 *security_password_length* が 0 (ゼロ) に設定されている場合は、*security_password* パラメーターは無視されます。これは、*security_password* をヌル・ストリングに設定するのと同じです。

## 戻りパラメーター

verb の実行後に、CS/AIX は次のパラメーターを戻します。

*return_code*
値は次のとおりです。

**CM_OK** コールは正常に実行されました。

**CM_PROGRAM_PARAMETER_CHECK**
次のいずれかの状態が発生しました。

- *conversation_ID* に指定された値が有効でない。
- *security_password_length* に指定された値が範囲外である。

**CM_PROGRAM_STATE_CHECK**
次のいずれかの状態が発生しました。

- 会話が初期化状態ではない。
- 会話のセキュリティー・タイプが CM_SECURITY_PROGRAM または CM_SECURITY_PROGRAM_STRONG に設定されていない。

**CM_PRODUCT_SPECIFIC_ERROR**
この戻りコードの説明については、191 ページの『付録 B. 共通な戻りコード』を参照してください。

## 発行時の状態

会話は初期化状態になっていなければなりません。

## 状態の変化

状態には変化はありません。

## 使用上の注意

パスワードのほかに、ユーザー ID が必要です。ユーザー ID は、直前の Initialize_Conversation コールで指定されたサイド情報エントリーから取得するか、Set_Conversation_Security_User_ID を使用してプログラムで指定できます。

有効でないパスワードは、Allocate コールにより生成された割り振り要求がパートナー LU に送信されるまでは検出されません。エラーは、後続のコールが発行された時点で呼び出し側プログラムに戻ります。

戻りコードが CM_OK でない場合は、会話特性 *security_password* および *security_password_length* は変更されません。

# Set_Conversation_Security_Password (xcscsp)

この関数は Java CPI-C では使用できません。

このコールは、呼び出し対象プログラムへのアクセスに必要なパスワードを指定するために、呼び出し側プログラムが発行します。

xcscsp コールは、X/Open CPI-C 定義を使用するアプリケーションに互換性を提供します。これは、コール Set_Conversation_Security_Password (cmscsp) としてすでに IBM CPI-C 2.0 に組み込まれています。他のプラットフォームへのプログラムの移植性を高めるために、できるだけ cmscsp を使用してください。

このコールのパラメーターは、cmscsp コールのパラメーターと同一です。cmscsp の詳細については、 127 ページの『Set_Conversation_Security_Password (cmscsp)』を参照してください。

## Set_Conversation_Security_Type (cmscst)

Set_Conversation_Security_Type コールは、呼び出し対象プログラムへのアクセスの妥当性検査にパートナー LU が必要とする情報を指定するために、呼び出し側プログラムが発行します。このコールは、Initialize_Conversation コールにより指定されたサイド情報から得られる初期セキュリティー・タイプを変更します。Allocate の発行後にこのコールを発行することはできません。

### 関数コール

```
void cmscst (
             unsigned char CM_PTR                    conversation_ID,
             XC_CONVERSATION_SECURITY_TYPE CM_PTR    conversation_security_type,
             CM_RETURN_CODE CM_PTR                   return_code
            );
```

### Java CPI-C の関数コール

AIX、LINUX

```
public native void cmscst (
                           byte[]                       conversation_ID,
                           CPICConversationSecurityType conversation_security_type,
                           CPICReturnCode               return_code
                          );
```

### 指定パラメーター

指定パラメーターは次のとおりです。

*conversation_ID*

このパラメーターは会話の ID です。このパラメーターの値は、Initialize_Conversation コールから戻されます。

*conversation_security_type*

このパラメーターは、パートナー LU が呼び出し対象プログラムへのアクセスの妥当性検査をするのに必要な情報を指定します。構成時に呼び出し対象プログラム用に設定された会話セキュリティーに基づき、次のいずれかの値を使用します。

AIX、LINUX

**CM_SECURITY_NONE**

呼び出し対象プログラムは会話セキュリティーを使用していません。

**CM_SECURITY_SAME**
呼び出し対象プログラムは会話セキュリティーを使用していて、検査済みインディケーター (14 ページの『会話セキュリティーの概要』を参照) を受け入れるように構成されています。ローカル・プログラムの現行コンテキスト (Allocate コールの発行時の) から得られるユーザー ID は、検査済みインディケーターと共に呼び出し対象プログラムに送信されます。このインディケーターは、パスワードが不要なことを呼び出し対象プログラムに通知します。

**CM_SECURITY_PROGRAM**
呼び出し対象プログラムは会話セキュリティーを使用しているため、ユーザー ID とパスワードが必要です。セキュリティー情報は、現行会話特性 (Allocate コールの発行時の) から取得されます。

**CM_SECURITY_PROGRAM_STRONG**
ローカル・ノードはネットワークを介して平文形式でパスワードを送信してはいけない、という点以外は CM_SECURITY_PROGRAM と同じです。この値を使用できるのは、リモート・システムがパスワードの置換をサポートしている場合だけです。

WINDOWS

**XC_SECURITY_NONE**
CM_SECURITY_NONE と同等

**XC_SECURITY_SAME**
CM_SECURITY_SAME と同等

**XC_SECURITY_PROGRAM**
CM_SECURITY_PROGRAM と同等

## 戻りパラメーター

verb の実行後に、CS/AIX は次のパラメーターを戻します。

*return_code*
値は次のとおりです。

**CM_OK** コールは正常に実行されました。

**CM_PROGRAM_STATE_CHECK**
会話は初期化状態ではありません。

**CM_PROGRAM_PARAMETER_CHECK**
*conversation_ID* または *conversation_security_type* に指定された値が有効ではありません。

**CM_PRODUCT_SPECIFIC_ERROR**
この戻りコードの説明については、191 ページの『付録 B. 共通な戻りコード』を参照してください。

### 発行時の状態

会話は初期化状態になっていなければなりません。

### 状態の変化

状態には変化はありません。

### 使用上の注意

*return_code* が CM_OK でない場合は、*conversation_security_type* は変更されません。

## Set_Conversation_Security_Type (xcscst)

この関数は Java CPI-C では使用できません。

このコールは、呼び出し対象プログラムへのアクセスの妥当性検査にパートナー LU が必要とする情報を指定するために、呼び出し側プログラムが発行します。このコールは、Initialize_Conversation コールにより指定されたサイド情報から得られる初期セキュリティー・タイプを変更します。

このコールは、X/Open CPI-C 定義を使用するアプリケーションに互換性を提供します。これは、コール Set_Conversation_Security_Type (cmscst) としてすでに IBM CPI-C 2.0 に組み込まれています。他のプラットフォームへのプログラムの移植性を高めるために、できるだけ cmscst を使用してください。

このコールのパラメーターは、cmscst コールのパラメーターと同一です。cmscst の詳細については、130 ページの『Set_Conversation_Security_Type (cmscst)』を参照してください。

## Set_Conversation_Security_User_ID (cmscsu)

Set_Conversation_Security_User_ID コールは、呼び出し対象プログラムへのアクセスに必要なユーザー ID を指定するために、呼び出し側プログラムが発行します。このコールは、Initialize_Conversation コールにより指定されたサイド情報から得られる初期ユーザー ID を変更します。

Allocate コールの発行後にこのコールを発行することはできません。会話セキュリティー・タイプが CM_SECURITY_NONE (AIX または Linux システム) または XC_SECURITY_NONE (Windows システム) である場合、このコールは無効です。

### 関数コール

```
void cmscsu (
             unsigned char CM_PTR      conversation_ID,
             unsigned char CM_PTR      security_user_ID,
             CM_INT32 CM_PTR           security_user_ID_length,
             CM_RETURN_CODE CM_PTR     return_code
            );
```

### Java CPI-C の関数コール

AIX、LINUX

```
public native void cmscsu (
                            byte[]          conversation_ID,
                            byte[]          security_user_ID
                            CPICLength      security_user_ID_length,
                            CPICReturnCode  return_code
                          );
```

## 指定パラメーター

指定パラメーターは次のとおりです。

*conversation_ID*
このパラメーターは会話の ID です。このパラメーターの値は、Initialize_Conversation コールから戻されます。

*security_user_ID*
このパラメーターは、パートナー・プログラムへのアクセスに必要なユーザー ID を示します。この値の範囲は、1 から 10 文字 (AIX または Linux システム)、または 1 から 8 文字 (Windows システム) であり、大/小文字の区別が必要です。

使用できる文字は次のとおりです。

- 大文字および小文字の英字
- 数字 0〜9
- 特殊文字 $、#、@、および . (ピリオド)

*security_user_ID_length*
このパラメーターは、*security_user_ID* の長さを示します。この値の範囲は、1 から 10 文字 (AIX または Linux システム)、または 1 から 8 文字 (Windows システム) です。長さが 0 (ゼロ) の場合は、*security_user_ID* パラメーターは無視されます。これは、*security_user_ID* をヌル・ストリングに設定するのと同じです。

## 戻りパラメーター

verb の実行後に、CS/AIX は次のパラメーターを戻します。

*return_code*
値は次のとおりです。

**CM_OK** コールは正常に実行されました。

**CM_PROGRAM_PARAMETER_CHECK**
次のいずれかの状態が発生しました。

- *conversation_ID* に指定された値が有効でない。
- *security_user_ID_length* に指定された値が範囲外である。

**CM_PROGRAM_STATE_CHECK**
次のいずれかの状態が発生しました。

- 会話が初期化状態ではない。

- 会話のセキュリティー・タイプが、CM_SECURITY_NONE に設定されている。

**CM_PRODUCT_SPECIFIC_ERROR**
この戻りコードの説明については、191 ページの『付録 B. 共通な戻りコード』を参照してください。

### 発行時の状態

会話は初期化状態になっていなければなりません。

### 状態の変化

状態には変化はありません。

### 使用上の注意

戻りコードが CM_OK でない場合は、会話特性 *security_user_ID* および *security_user_ID_length* は変更されません。

有効でないユーザー ID は、Allocate コールにより生成される割り振り要求がパートナー LU に送信されるまでは検出されません。エラーは、後続のコールが発行された時点で呼び出し側プログラムに戻ります。

## Set_Conversation_Security_User_ID (xcscsu)

この関数は Java CPI-C では使用できません。

このコールは、呼び出し対象プログラムへのアクセスに必要なユーザー ID を指定するために、呼び出し側プログラムが発行します。

xcscsu コールは、X/Open CPI-C 定義を使用するアプリケーションに互換性を提供します。これは、コール Set_Conversation_Security_User_ID (cmscsu) としてすでに IBM CPI-C 2.0 に組み込まれています。他のプラットフォームへのプログラムの移植性を高めるために、できるだけ cmscsu を使用してください。

このコールのパラメーターは、cmscsu コールのパラメーターと同一です。 cmscsu の詳細については、132 ページの『Set_Conversation_Security_User_ID (cmscsu)』を参照してください。

## Set_Conversation_Type (cmsct)

Set_Conversation_Type コールは、会話をマップ式会話または基本会話として定義するために、呼び出し側プログラムが発行します。このコールは、Initialize_Conversation コールにより設定されたデフォルトの会話タイプを変更します。デフォルトの会話タイプは CM_MAPPED_CONVERSATION です。Allocate の発行後にこのコールを発行することはできません。

## 関数コール

```
void cmsct (
            unsigned char CM_PTR          conversation_ID,
            CM_CONVERSATION_TYPE CM_PTR   conversation_type,
            CM_RETURN_CODE CM_PTR         return_code
           );
```

## Java CPI-C の関数コール

AIX、LINUX

```
public native void cmsct (
                           byte[]                conversation_ID,
                           CPICConversationType  conversation_type,
                           CPICReturnCode        return_code
                         );
```

## 指定パラメーター

指定パラメーターは次のとおりです。

*conversation_ID*
このパラメーターは会話の ID です。このパラメーターの値は、Initialize_Conversation コールから戻されます。

*conversation_type*
このパラメーターは、Allocate コールにより割り振られる会話のタイプを指定します。値は次のとおりです。

CM_BASIC_CONVERSATION
CM_MAPPED_CONVERSATION

## 戻りパラメーター

verb の実行後に、CS/AIX は、処理が正常に実行されたかどうかを示すパラメーターと、失敗した場合はその理由を示すパラメーターを戻します。

*return_code*
値は次のとおりです。

**CM_OK** コールは正常に実行されました。

**CM_PROGRAM_STATE_CHECK**
会話は初期化状態ではありません。

**CM_PROGRAM_PARAMETER_CHECK**
次のいずれかの状態が発生しました。

- *conversation_ID* または *conversation_type* に指定された値が有効でない。
- *conversation_type* パラメーターにはマップ式会話が指定されているが、充てん特性は CM_FILL_BUFFER に設定されていて、マップ

式会話と両立できない。Set_Fill コールを発行して充てんタイプを CM_FILL_LL に変更してから、会話タイプをマップ式に変更する必要があります。

- *conversation_type* パラメーターにマップ式会話が指定されているが、基本会話についてのみ使用できる前の Set_Log_Data コールがまだ効力を持っている。

**CM_PRODUCT_SPECIFIC_ERROR**
この戻りコードの説明については、191 ページの『付録 B. 共通な戻りコード』を参照してください。

## 発行時の状態

会話は初期化状態になっていなければなりません。

## 状態の変化

状態には変化はありません。

## 使用上の注意

戻りコードが CM_OK でない場合は、*conversation_type* 会話特性は変更されません。

---

# Set_CPIC_Side_Information (xcmssi)

この関数は Java CPI-C では使用できません。

Set_CPIC_Side_Information コールは、このアプリケーションが使用するためのサイド情報エントリーを指定します。 CPI-C サイド情報エントリーは、一組の会話特性をシンボリック宛先名に関連付けます。

サイド情報エントリーは、CS/AIX 構成ファイルの中で定義されます。このコールは、このアプリケーションが使用する追加エントリーを指定するか、指定のシンボリック宛先名がすでに存在している場合は、構成ファイル内の定義 (またはアプリケーションのローカル定義) を変更します。

このコールは、X/Open CPI-C および Windows CPI-C 仕様との互換性を確保するために提供されているもので、IBM CPI-C 2.0 には組み込まれていません。

## 関数コール

```
     void xcmssi (
                  unsigned char CM_PTR      key,
                  SIDE_INFO CM_PTR          side_info_entry,
                  CM_INT32 CM_PTR           side_info_entry_length,
                  CM_RETURN_CODE CM_PTR     return_code
                 );

typedef struct side_info_entry
{
  unsigned char     sym_dest_name[8];      /* symbolic destination name      */
  unsigned char     partner_LU_name[17];   /* Fully qualified partner LU name*/
  unsigned char     reserved[3];           /* Reserved                       */
  XC_TP_NAME_TYPE   TP_name_type;          /* TP name type                   */
  unsigned char     TP_name[64];           /* TP name                        */
  unsigned char     mode_name[8];          /* Mode name                      */
  XC_CONVERSATION_SECURITY_TYPE
```

```
                    conversation_security_type; /* Conversation security type*/
  unsigned char     security_user_ID[8];   /* User ID                        */
  unsigned char     security_password[8];  /* Password                       */
} SIDE_INFO;
```

## 指定パラメーター

指定パラメーターは次のとおりです。

*key*　このパラメーターは無視されます。

*side_info_entry*

このパラメーターは、次のように、サイド情報エントリーの内容を示します。構造内の各フィールドは左寄せにする必要があります。必要に応じて、フィールドの右の部分にスペースを埋め込んでください。

*side_info_entry.sym_dest_name*

サイド情報エントリーを識別するシンボリック宛先名。パラメーター *sym_dest_name* は、8 バイトの ASCII 文字ストリングで、表示可能な文字はすべて使用することができます。

*side_info_entry.partner_LU_name*

パートナー LU の完全修飾名。この名前は、ドットで連結された 2 つの文字ストリングで構成されます。それぞれの名前は、組み込みスペースを含まず、最大 8 バイトです。有効な文字は、大文字の英字 A〜Z と数字 0〜9 です。

*side_info_entry.TP_name_type*

ターゲット TP のタイプ (TP 名に対して有効な文字は、TP タイプによって決まります)。指定できる値は次のとおりです。

**XC_APPLICATION_TP**

アプリケーション TP。TP 名のすべての文字は、有効な ASCII 文字でなければなりません。

**XC_SNA_SERVICE_TP**

サービス TP。TP 名は、16 進数字 (4 文字) 2 つを表す 8 文字の ASCII ストリングとして指定しなければなりません。たとえば、名前の 16 進表示が 0x21F0F0F8 の場合は、*tp_name* パラメーターを 8 文字ストリング「21F0F0F8」に設定します。

最初の文字 (2 バイトで表されます) は、0x0E および 0x0F を除く、0x0〜0x3F の範囲の 16 進値でなければなりません。残りの文字 (それぞれ 2 バイトで表されます) は、有効な EBCDIC 文字でなければなりません。

*side_info_entry.TP_name*

ターゲット TP の TP 名。

Set_CPIC_Side_Information は、SNA サービス TP をパートナー・プログラムとして指定できる唯一の CPI-C コールです。TP 名の指定方法の詳細については、上の *TP_name_type* パラメーターの説明を参照してください。

*side_info_entry.mode_name*

ターゲット TP にアクセスするために使用するモードの名前。

マップ式会話の場合、モード名 SNASVCMG は SNA の内部使用のために予約済みです。この名前を使用すると、Allocate コールは失敗します。基本会話では SNASVCMG を使用しないようお勧めします。また、どちらの会話タイプにおいても、CPSVCMG (これも SNA 予約名です) は使用しないでください。

*side_info_entry.conversation_security_type*
ターゲット TP が会話セキュリティーを使用するかどうかを示します。指定できる値は次のとおりです。

AIX、LINUX

**CM_SECURITY_NONE**
ターゲット TP は会話セキュリティーを使用していません。

**CM_SECURITY_PROGRAM**
ターゲット TP は会話セキュリティーを使用しています。ターゲット TP へのアクセスには、以下に示す *security_user_ID* と *security_password* という 2 つのパラメーターが使用されます。

**CM_SECURITY_SAME**
ターゲット TP は会話セキュリティーを使用していて、ローカル TP からの「検査済み」インディケーターを受け入れることができます。(これは、ローカル TP 自体が別の TP から呼び出され、その TP から提供されたセキュリティー・ユーザー ID とパスワードを検査したことを示しています)。ターゲット TP へのアクセスには、以下に示す *security_user_ID* パラメーターが使用されます。パスワードは不要です。

**CM_SECURITY_PROGRAM_STRONG**
ローカル・ノードはネットワークを介して平文形式でパスワードを送信してはいけない、という点以外は CM_SECURITY_PROGRAM と同じです。この値を使用できるのは、リモート・システムがパスワードの置換をサポートしている場合だけです。

WINDOWS

**XC_SECURITY_NONE**
CM_SECURITY_NONE と同等

**XC_SECURITY_SAME**
CM_SECURITY_SAME と同等

**XC_SECURITY_PROGRAM**
CM_SECURITY_PROGRAM と同等

*side_info_entry.security_user_ID*
パートナー TP にアクセスするために使用するユーザー ID。*conversation_security_type* パラメーターが CM_SECURITY_NONE に設定されている場合は、このパラメーターは不要です。

*side_info_entry.security_password*
パートナー TP にアクセスするために使用するパスワード。このパラメーターが必要なのは、*conversation_security_type* パラメーターが CM_SECURITY_PROGRAM または CM_SECURITY_PROGRAM_STRONG に設定されているときだけです。

AIX、LINUX

セキュリティー・ユーザー ID には 10 文字まで指定できますが、X/Open CPI-C との互換性を確保するために、この verb では、ユーザー ID とパスワードには 8 文字までしか指定できません。パートナー TP で 9 文字または 10 文字のユーザー ID またはパスワードが必要な場合は、Set_Conversation_Security_User_ID コール、または Set_Conversation_Security_Password コールを使用して、明示的に指定する必要があります。

*side_info_entry_length*
この値は常に sizeof(SIDE_INFO) に設定する必要があります。

WINDOWS

*side_info_entry_length*
この値は常に 124 に設定する必要があります。

## 戻りパラメーター

verb の実行後に、CS/AIX は、処理が正常に実行されたかどうかを示すパラメーターと、失敗した場合はその理由を示すパラメーターを戻します。

*return_code*
値は次のとおりです。

**CM_OK** コールは正常に実行されました。

**CM_PROGRAM_PARAMETER_CHECK**
次のいずれかの状態が発生しました。

- *side_info_entry* 構造に指定された値が有効でない。
- *side_info_entry* の先頭文字にスペースが含まれている。

**CM_PRODUCT_SPECIFIC_ERROR**
この戻りコードの説明については、191 ページの『付録 B. 共通な戻りコード』を参照してください。

## 発行時の状態

会話の状態は問いません。

## 状態の変化

状態には変化はありません。

## 使用上の注意

このコールにより、構成ファイル内に保持されているサイド情報は変更されません。変更は、このアプリケーションだけに適用されます。CS/AIX は変更後の情報をこのオペレーティング・システム・プロセスに関連付けられたメモリーに保管します。そのプロセスが終了すると (またはアプリケーションが Delete_CPIC_Side_Information コールを発行してそのエントリーを除去すると)、変更内容は破棄されます。詳しくは、34 ページの『サイド情報』を参照してください。

*return_code* が CM_OK でない場合は、サイド情報は変更されません。

サイド情報内の有効でないストリング・パラメーター (たとえば、存在しないパートナー LU を指定しているもの) は、Allocate コールが発行されるまでは検出されません。エラーは後続の Allocate コールで発行された時点で戻ります。

# Set_Deallocate_Type (cmsdt)

Set_Deallocate_Type コールは、会話の割り振りを解除する方法を指定します。このコールは、Initialize_Conversation コール、または Accept_Conversation コールにより設定されたデフォルトの割り振り解除タイプを変更します。デフォルトの割り振り解除タイプは CM_DEALLOCATE_SYNC_LEVEL です。

このコールにより指定された割り振り解除命令が有効になるのは、Deallocate コールが発行されたとき、または送信タイプが CM_SEND_AND_DEALLOCATE に設定されていて Send_Data コールが発行されたときです。

## 関数コール

```
void cmsdt (
            unsigned char CM_PTR          conversation_ID,
            CM_DEALLOCATE_TYPE CM_PTR     deallocate_type,
            CM_RETURN_CODE CM_PTR         return_code
           );
```

## Java CPI-C の関数コール

AIX、LINUX

```
public native void cmsdt (
                           byte[]              conversation_ID,
                           CPICDeallocateType  deallocate_type,
                           CPICReturnCode      return_code
                          );
```

## 指定パラメーター

指定パラメーターは次のとおりです。

*conversation_ID*
このパラメーターは会話の ID です。このパラメーターの値は、

Initialize_Conversation コール、Initialize_For_Incoming コール、または Accept_Conversation コールから戻されます。

*deallocate_type*

このパラメーターは、割り振り解除の実行方法を指定します。値は次のとおりです。

**CM_DEALLOCATE_ABEND**

会話の割り振りは、異常状態として無条件に解除されます。プログラムでは、トランザクションの正常終了を妨げるエラーを検出したときは、CM_DEALLOCATE_ABEND を指定する必要があります。

会話が送信状態の場合は、CPI-C は、会話の割り振りを解除する前に、ローカル LU の送信バッファーの内容をパートナー・プログラムに送信します。会話が受信状態の場合は、着呼データが除去されることがあります。送信状態の基本会話の場合は、論理レコードが切り捨てられることがあります。

**CM_DEALLOCATE_CONFIRM**

この値は、ローカル LU の送信バッファーの内容と割り振り解除確認の要求をパートナー・プログラムに送信します。会話の同期レベルが CM_NONE の場合は、アプリケーションはこの値を使用できません。

この割り振り解除確認の要求は、Deallocate コールにより送信されるか、または送信タイプを CM_SEND_AND_DEALLOCATE に設定した Send_Data コールにより送信されます。確認要求に応答してパートナー・プログラムが Confirmed コールを発行すると、会話は正常に割り振り解除されます。

**CM_DEALLOCATE_FLUSH**

この値は、会話を正常に割り振り解除する前に、ローカル LU の送信バッファーの内容をパートナー・プログラムに送信します。

**CM_DEALLOCATE_SYNC_LEVEL**

この値は、会話の同期レベルを使用して、会話の割り振り解除方法を判断します。デフォルトの同期レベルは、Initialize_Conversation コールにより設定され、Set_Sync_Level コールにより変更できます。

会話の同期レベルがデフォルトの CM_NONE に設定されている場合は、ローカル LU の送信バッファーの内容がパートナー・プログラムに送信され、会話が正常に割り振り解除されます。

会話の同期レベルが CM_CONFIRM の場合は、ローカル LU の送信バッファーの内容および割り振り解除確認の要求がパートナー・プログラムに送信されます。この割り振り解除確認の要求は、Deallocate コールにより送信されるか、または送信タイプを CM_SEND_AND_DEALLOCATE に設定した Send_Data コールにより送信されます。確認要求に応答してパートナー・プログラムが Confirmed コールを発行すると、会話は正常に割り振り解除されます。

## 戻りパラメーター

verb の実行後に、CS/AIX は、処理が正常に実行されたかどうかを示すパラメーターと、失敗した場合はその理由を示すパラメーターを戻します。

*return_code*
値は次のとおりです。

**CM_OK** コールは正常に実行されました。

**CM_PROGRAM_PARAMETER_CHECK**
次のいずれかの状態が発生しました。

- *conversation_ID* または *deallocate_type* に指定された値が有効でない。
- *deallocate_type* パラメーターに CM_DEALLOCATE_CONFIRM が指定されているが、会話の同期レベルが CM_NONE に設定されている。

**CM_PRODUCT_SPECIFIC_ERROR**
この戻りコードの説明については、191 ページの『付録 B. 共通な戻りコード』を参照してください。

## 発行時の状態

会話は、リセット以外であればどの状態にあっても構いません。

## 状態の変化

状態には変化はありません。

## 使用上の注意

*return_code* が CM_OK でない場合は、*deallocate_type* 会話特性は変更されません。

会話の同期レベルが CM_NONE または CM_CONFIRM に設定されている場合は、*deallocate_type* を CM_FLUSH に設定できます。

CM_DEALLOCATE_FLUSH は、会話の同期レベルが CM_NONE に設定されている状態で CM_DEALLOCATE_SYNC_LEVEL を指定した場合と同じ意味を持ちます。

CM_DEALLOCATE_CONFIRM は、会話の同期レベルが CM_CONFIRM に設定されている状態で CM_DEALLOCATE_SYNC_LEVEL を指定した場合と同じ意味を持ちます。

# Set_Error_Direction (cmsed)

Set_Error_Direction コールは、プログラムがエラーを検出したのがデータの受信中かデータ送信の準備中かを示します。このコールは、Initialize_Conversation コール、または Accept_Conversation コールにより設定されたエラー時の通信方向のデフォルトを変更します。エラー時の通信方向のデフォルトは CM_RECEIVE_ERROR です。

エラー時の通信方向が有効なのは、プログラムが Receive コールを発行し、データ (*data_received* が CM_NO_DATA_RECEIVED 以外の値) と送信インディケーター (*status_received* = CM_SEND_RECEIVED) を受信した直後に、送信 - 保留状態で Send_Error コールを発行したときだけです。

## 関数コール

```
void cmsed (
            unsigned char CM_PTR          conversation_ID,
            CM_ERROR_DIRECTION CM_PTR     error_direction,
            CM_RETURN_CODE CM_PTR         return_code
           );
```

## Java CPI-C の関数コール

AIX、LINUX

```
public native void cmsed (
                           byte[]              conversation_ID,
                           CPICErrorDirection  error_direction,
                           CPICReturnCode      return_code
                         );
```

## 指定パラメーター

指定パラメーターは次のとおりです。

*conversation_ID*
このパラメーターは会話の ID です。このパラメーターの値は、Initialize_Conversation コール、Initialize_For_Incoming コール、または Accept_Conversation コールから戻されます。

*error_direction*
このパラメーターは、プログラムがエラーを検出したときのデータの通信方向を示します。値は次のとおりです。

**CM_RECEIVE_ERROR**
パートナー・プログラムから受信したデータにエラーが起きました。

**CM_SEND_ERROR**
ローカル・プログラムがパートナー・プログラムにデータを送信する準備をしている間に、エラーが起きました。

## 戻りパラメーター

verb の実行後に、CS/AIX は、処理が正常に実行されたかどうかを示すパラメーターと、失敗した場合はその理由を示すパラメーターを戻します。

*return_code*
値は次のとおりです。

**CM_OK** コールは正常に実行されました。

**CM_PROGRAM_PARAMETER_CHECK**
*conversation_ID* または *error_direction* に指定された値が有効ではありません。

**CM_PRODUCT_SPECIFIC_ERROR**
この戻りコードの説明については、 191 ページの『付録 B. 共通な戻りコード』を参照してください。

## 発行時の状態

会話は、リセット以外であればどの状態にあっても構いません。

## 状態の変化

状態には変化はありません。

## 使用上の注意

*return_code* が CM_OK でない場合は、*error_direction* 会話特性は変更されません。

会話が送信 - 保留状態であって、プログラムが受信データにエラーを検出した場合、またはローカル・プログラムがデータの送信準備をしている間にエラーが起きた場合は、プログラムは Send_Error コールを発行します。 LU はどちらの種類のエラー (受信か送信か) が起きたのか判断できないため、プログラムは、Send_Error コールを発行する前に、Set_Error_Direction コールを使用してエラー時の通信方向の情報を提供する必要があります。新しいエラー時の通信方向は、後続の Set_Error_Direction により変更されるまで有効です。

Send_Error コールが発行されると、パートナー・プログラムは次のどちらかの戻りコードを受け取ります。

- CM_PROGRAM_ERROR_PURGING (*error_direction* が CM_RECEIVE_ERROR に設定されている場合)
- CM_PROGRAM_ERROR_NO_TRUNC (*error_direction* が CM_SEND_ERROR に設定されている場合)

---

# Set_Fill (cmsf)

Set_Fill コールは、プログラムがデータを論理レコードの形式で受信するか、指定長のデータとして受信するかを指定します。このコールを使用できるのは、基本会話のときだけです。これは、Initialize_Conversation コール、または Accept_Conversation コールにより設定されたデフォルトの充てんを変更します。デフォルト時の充てんは CM_FILL_LL です。

充てん値は、後続のすべての Receive コールに影響を与えます。値を変更するには、Set_Fill コールを再発行します。

## 関数コール

```
void cmsf  (
            unsigned char CM_PTR       conversation_ID,
            CM_FILL CM_PTR             fill,
            CM_RETURN_CODE CM_PTR      return_code
           );
```

## Java CPI-C の関数コール

AIX、LINUX

```
public native void cmsf (
                           byte[]          conversation_ID,
                           CPICFill        fill,
                           CPICReturnCode  return_code
                         );
```

## 指定パラメーター

指定パラメーターは次のとおりです。

*conversation_ID*
このパラメーターは会話の ID です。このパラメーターの値は、Initialize_Conversation コール、Initialize_For_Incoming コール、または Accept_Conversation コールから戻されます。

*fill* このパラメーターは、プログラムがデータを受信する形式を指定します。値は次のとおりです。

**CM_FILL_BUFFER**
ローカル・プログラムは、Receive コールの *requested_length* パラメーターにより指定されたバイト数に達するまで、またはデータの終わりに達するまで、データを受信します。データは、論理レコード形式に関係なく受信されます。

**CM_FILL_LL**
データは論理レコード形式で受信されます。受信されるデータは次のいずれかです。

- 完全な論理レコード
- Receive コールの *requested_length* パラメーターに等しい、論理レコードの一部分
- 論理レコードの終わり

## 戻りパラメーター

verb の実行後に、CS/AIX は、処理が正常に実行されたかどうかを示すパラメーターと、失敗した場合はその理由を示すパラメーターを戻します。

*return_code*
値は次のとおりです。

**CM_OK** コールは正常に実行されました。

**CM_PROGRAM_PARAMETER_CHECK**
次のいずれかの状態が発生しました。

- *conversation_ID* パラメーターまたは *fill* パラメーターに指定された値が有効でない。

- 現行会話がマップ式である。*fill* パラメーターはマップ式会話には適用されません。

**CM_PRODUCT_SPECIFIC_ERROR**
この戻りコードの説明については、191 ページの『付録 B. 共通な戻りコード』を参照してください。

## 発行時の状態

会話は、リセット以外であればどの状態にあっても構いません。

## 状態の変化

状態には変化はありません。

## 使用上の注意

*return_code* が CM_OK でない場合は、*fill* 会話特性は変更されません。

---

# Set_Local_LU_Name (cmslln)

Set_Local_LU_Name コールは、会話のためのローカル LU を指定するために、呼び出し側プログラムが発行します。このコールは、Initialize_Conversation の発行時にサイド情報から得られるシステム定義のローカル LU、および APPCLLU 環境変数により指定されたすべてのローカル LU を変更します。Allocate の発行後にこのコールを発行することはできません。このコールを発行しても、サイド情報自体には影響はありません。

このコールは標準 CPI-C 仕様の一部ではありません。他のインプリメンテーションでは使用できない場合があります。特に、他の Java CPI-C インプリメンテーションではサポートされていません。

## 関数コール

```
void cmslln   (
               unsigned char CM_PTR         Conversation_ID,
               unsigned char CM_PTR         lu_alias,
               CM_INT32 CM_PTR              lu_alias_length,
               CM_RETURN_CODE CM_PTR        return_code
              );
```

## Java CPI-C の関数コール

AIX、LINUX

```
public native void cmslln (
                             byte[]          conversation_ID,
                             byte[]          lu_alias,
                             CPICLength      lu_alias_length,
                             CPICReturnCode  return_code
                           );
```

## 指定パラメーター

指定パラメーターは次のとおりです。

*conversation_ID*
このパラメーターは会話の ID です。このパラメーターの値は、Initialize_Conversation コール、Initialize_For_Incoming コール、または Accept_Conversation コールから戻されます。

*lu_alias*
このパラメーターは、LU の別名の開始アドレスを示します。LU の別名には、最大 8 文字の ASCII 文字を使用できます。

*lu_alias_length*
このパラメーターは、LU の別名の長さを示します。この値の範囲は、0〜8 バイトです。*lu_alias_length* が 0 (ゼロ) の場合は、LU の別名は全桁ゼロに設定されます。

## 戻りパラメーター

verb の実行後に、CS/AIX は次のパラメーターを戻します。

*return_code*
値は次のとおりです。

**CM_OK** コールは正常に実行されました。

**CM_PROGRAM_STATE_CHECK**
会話は初期化状態ではありません。

**CM_PROGRAM_PARAMETER_CHECK**
次のいずれかの状態が発生しました。

- *conversation_ID* に指定された値が有効でない。
- *lu_alias_length* に指定された値が範囲外である (8 より大きいかまたは 0 より小さい)。

**CM_PRODUCT_SPECIFIC_ERROR**
この戻りコードの説明については、191 ページの『付録 B. 共通な戻りコード』を参照してください。

## 発行時の状態

会話は初期化状態になっていなければなりません。

## 状態の変化

状態には変化はありません。

## 使用上の注意

*return_code* が CM_OK でない場合は、*lu_alias* 会話特性は変更されません。

*lu_alias* に有効でない値 (構成ファイルで使えない名前) を指定しても、Allocate コールが発行されるまでは検出されません。

# Set_Log_Data (cmsld)

Set_Log_Data コールは、パートナー LU に送信するログ・メッセージ (ログ・データ) とその長さを指定します。このコールを使用できるのは、基本会話のときだけです。このコールは、デフォルトのログ・データ (ヌル) とデフォルトのログ・データ長 (0 (ゼロ) ) を変更します。

## 関数コール

```
void cmsld (
            unsigned char CM_PTR       conversation_ID,
            unsigned char CM_PTR       log_data,
            CM_INT32 CM_PTR            log_data_length,
            CM_RETURN_CODE CM_PTR      return_code
           );
```

## Java CPI-C の関数コール

AIX、LINUX

```
public native void cmsld (
                            byte[]          conversation_ID,
                            byte[]          log_data,
                            CPICLength      log_data_length,
                            CPICReturnCode  return_code
                          );
```

## 指定パラメーター

指定パラメーターは次のとおりです。

*conversation_ID*

このパラメーターは会話の ID です。このパラメーターの値は、Initialize_Conversation コール、Initialize_For_Incoming コール、または Accept_Conversation コールから戻されます。

*log_data*

エラー情報が入っているデータ・バッファーのアドレス。このデータは、ローカル・エラー・ログとパートナー LU に送信されます。

このパラメーターは、*log_data_length* が 0 (ゼロ) より大きいときに、Send_Error コールで使用されます。

プログラムは、エラー・データを汎用データ・ストリーム (GDS) エラー・ログ変数として形式設定する必要があります。詳細については、IBM 発行の資料「*IBM Systems Network Architecture: LU 6.2 Reference: Peer Protocols*」を参照してください。

*log_data_length*

このパラメーターは、ログ・データの長さを示します。

この値の範囲は、0〜512 バイトです。

長さ 0 (ゼロ) は、ログ・データがないことを示します。*log_data* パラメーターは無視され、*log_data* 会話特性はヌル・ストリングに設定されます。

## 戻りパラメーター

verb の実行後に、CS/AIX は次のパラメーターを戻します。

*return_code*

値は次のとおりです。

**CM_OK** コールは正常に実行されました。

**CM_PROGRAM_PARAMETER_CHECK**

次のいずれかの状態が発生しました。

- *conversation_ID* に指定された値が有効でない。
- 会話タイプがマップ式に設定されている。
- *log_data_length* に指定された値が範囲外である (512 より大きいかまたは 0 より小さい)。

**CM_PRODUCT_SPECIFIC_ERROR**

この戻りコードの説明については、191 ページの『付録 B. 共通な戻りコード』を参照してください。

## 発行時の状態

会話は、リセット以外であればどの状態にあっても構いません。

## 状態の変化

状態には変化はありません。

## 使用上の注意

*return_code* が CM_OK でない場合は、会話特性 *log_data* および *log_data_length* は変更されません。

ローカル・プログラムが次のいずれかのコールを発行すると、Set_Log_Data コールで指定されたログ・データがパートナー LU に送信されます。

- Send_Error
- Deallocate (会話の *deallocate_type* が CM_DEALLOCATE_ABEND に設定されている場合)
- Send_Data (会話の *send_type* が CM_SEND_AND_DEALLOCATE に、*deallocate_type* が CM_DEALLOCATE_ABEND に設定されている場合)

パートナー LU にログ・データを送信したあとで、ローカル LU はログ・データをヌルにリセットし、ログ・データ長を 0 (ゼロ) にリセットします。

CPI-C は、必要に応じて、ログ・データを ASCII から EBCDIC に自動的に変換します。

# Set_Mode_Name (cmsmn)

Set_Mode_Name コールは、会話のためのモード名を指定するために、呼び出し側プログラムが発行します。このコールは、Initialize_Conversation コールの発行時にサイド情報から得られたシステム定義のモード名を変更します。Allocate の発行後にこのコールを発行することはできません。このコールを発行しても、サイド情報自体には影響はありません。

## 関数コール

```
void cmsmn (
            unsigned char CM_PTR       conversation_ID,
            unsigned char CM_PTR       mode_name,
            CM_INT32 CM_PTR            mode_name_length,
            CM_RETURN_CODE CM_PTR      return_code
           );
```

## Java CPI-C の関数コール

AIX、LINUX

```
public native void cmsmn (
                            byte[]          conversation_ID,
                            byte[]          mode_name,
                            CPICLength      mode_name_length,
                            CPICReturnCode  return_code
                          );
```

## 指定パラメーター

指定パラメーターは次のとおりです。

*conversation_ID*

このパラメーターは会話の ID です。このパラメーターの値は、Initialize_Conversation コールから戻されます。

*mode_name*

このパラメーターは、モード名 (構成時に定義された一連のネットワーク特性の名前) の開始アドレスを示します。モード名には、最大 8 文字の ASCII 文字を使用できます。使用できる文字は次のとおりです。

- 大文字の英字
- 数字 0〜9

名前の先頭文字は、文字でなければなりませんが、SNA 定義のモードのいずれかに対しては、例えば #INTER のように、# でも構いません。 SNA 定義モードについて詳しくは、「*Communications Server for AIX 管理ガイド*」を参照してください。

*mode_name* の値は、構成時にパートナー LU に関連付けられたモードの名前に一致していなければなりません。

マップ式会話の場合、モード名 SNASVCMG は SNA の内部使用のために予約済みです。この名前を使用すると、Allocate コールは失敗します。基本会話では SNASVCMG を使用しないようお勧めします。また、どちらの会話タイプにおいても、CPSVCMG (これも SNA 予約名です) は使用しないでください。

*mode_name_length*
このパラメーターは、モード名の長さを示します。

この値の範囲は、0〜8 バイトです。

*mode_name_length* が 0 (ゼロ) に設定されている場合は、Set_Mode_Name コールは無視されます。

## 戻りパラメーター

verb の実行後に、CS/AIX は次のパラメーターを戻します。

*return_code*
値は次のとおりです。

**CM_OK** コールは正常に実行されました。

**CM_PROGRAM_STATE_CHECK**
会話は初期化状態ではありません。

**CM_PROGRAM_PARAMETER_CHECK**
次のいずれかの状態が発生しました。

- *conversation_ID* に指定された値が有効でない。
- *mode_name_length* に指定された値が範囲外である (8 より大きいかまたは 0 より小さい)。

**CM_PRODUCT_SPECIFIC_ERROR**
この戻りコードの説明については、191 ページの『付録 B. 共通な戻りコード』を参照してください。

## 発行時の状態

会話は初期化状態になっていなければなりません。

## 状態の変化

状態には変化はありません。

## 使用上の注意

*return_code* が CM_OK でない場合は、*mode_name* 会話特性は変更されません。

*mode_name* に有効でない値 (構成ファイルで使えない名前) を指定しても、Allocate コールが発行されるまでは検出されません。

# Set_Partner_LU_Name (cmspln)

Set_Partner_LU_Name コールは、パートナー LU 名を指定するために、呼び出し側プログラムが発行します。このコールは、Initialize_Conversation コールの発行時にサイド情報から得られたシステム定義のパートナー LU 名を変更します。Allocate の発行後にこのコールを発行することはできません。このコールを発行しても、サイド情報自体には影響はありません。

## 関数コール

```
void cmspln (
             unsigned char CM_PTR      conversation_ID,
             unsigned char CM_PTR      partner_LU_name,
             CM_INT32 CM_PTR           partner_LU_name_length,
             CM_RETURN_CODE CM_PTR     return_code
            );
```

## Java CPI-C の関数コール

AIX、LINUX

```
public native void cmspln (
                             byte[]          conversation_ID,
                             byte[]          partner_LU_name,
                             CPICLength      partner_LU_name_length,
                             CPICReturnCode  return_code
                           );
```

## 指定パラメーター

指定パラメーターは次のとおりです。

*conversation_ID*
このパラメーターは会話の ID です。このパラメーターの値は、Initialize_Conversation コールから戻されます。

*partner_LU_name*
このパラメーターは、パートナー LU 名の開始アドレスを示します。使用できる文字は次のとおりです。

- 大文字の英字
- 数字 0〜9

パートナー LU 名は、次のどちらかです。

- 1〜8 文字の ASCII 文字から成る別名
- 2〜17 文字の ASCII 文字から成る完全修飾ネットワーク名。ピリオド (.) で、ネットワーク ID (0〜8 文字) とネットワーク LU 名 (1〜8 文字) を区切ります。ネットワーク ID の長さがゼロ文字の場合でも、ピリオドは必要です。

パートナー LU を別名で指定する場合は、その別名は CS/AIX 構成の中でパートナー LU 用に定義されている別名と一致していなければなりません。

*partner_LU_name_length*
このパラメーターは、パートナー LU 名の長さを示します。

この値の範囲は、1〜17 です。

## 戻りパラメーター

verb の実行後に、CS/AIX は次のパラメーターを戻します。

*return_code*
値は次のとおりです。

**CM_OK** コールは正常に実行されました。

**CM_PROGRAM_STATE_CHECK**
会話は初期化状態ではありません。

**CM_PROGRAM_PARAMETER_CHECK**
次のいずれかの状態が発生しました。

- *conversation_ID* に指定された値が有効でない。
- *partner_LU_name_length* に指定された値が範囲外である。

**CM_PRODUCT_SPECIFIC_ERROR**
この戻りコードの説明については、191 ページの『付録 B. 共通な戻りコード』を参照してください。

## 発行時の状態

会話は初期化状態になっていなければなりません。

## 状態の変化

状態には変化はありません。

## 使用上の注意

*return_code* が CM_OK でない場合は、*partner_LU_name* 会話特性は変更されません。

*partner_LU_name* に有効でない値 (構成で使えない名前) を指定しても、Allocate コールが発行されるまでは検出されません。

# Set_Prepare_To_Receive_Type (cmsptr)

Set_Prepare_To_Receive_Type コールは、後続の Prepare_To_Receive コールの実行方法を指定します。これは、Initialize_Conversation コール、または Accept_Conversation コールにより設定されたデフォルトの受信準備処理を変更します。デフォルトでは、受信準備処理は会話の同期レベルに基づいて決められています。

受信準備タイプは、後続のすべての Prepare_To_Receive コールに影響を与えます。このタイプを変更するには、Set_Prepare_To_Receive_Type コールを再発行します。

## 関数コール

```
void cmsptr (
             unsigned char CM_PTR                    conversation_ID,
             CM_PREPARE_TO_RECEIVE_TYPE CM_PTR       prepare_to_receive_type,
             CM_RETURN_CODE CM_PTR                   return_code
            );
```

## Java CPI-C の関数コール

AIX、LINUX

```
public native void cmsptr (
                            byte[]                    conversation_ID,
                            CPICPrepareToReceiveType  prepare_to_receive_type,
                            CPICReturnCode            return_code
                           );
```

## 指定パラメーター

指定パラメーターは次のとおりです。

*conversation_ID*
このパラメーターは会話の ID です。このパラメーターの値は、Initialize_Conversation コール、Initialize_For_Incoming コール、または Accept_Conversation コールから戻されます。

*prepare_to_receive_type*
このパラメーターは、後続の Prepare_To_Receive コールの実行方法を指定します。値は次のとおりです。

**CM_PREP_TO_RECEIVE_CONFIRM**
この値は、LU の送信バッファーの内容と確認要求をパートナー・プログラムに送信します。確認を受け取ると、会話は受信状態に変わります。

**CM_PREP_TO_RECEIVE_FLUSH**
この値は、ローカル LU の送信バッファーの内容をパートナー・プログラムに送信し、会話を受信状態に変更します。

**CM_PREP_TO_RECEIVE_SYNC_LEVEL**
この値は、会話の同期レベルを使用して、会話の受信準備処理を判別します。デフォルトの同期レベルは、Initialize_Conversation コールにより設定され、Set_Sync_Level コールにより変更できます。

会話の同期レベルがデフォルト値 CM_NONE に設定されている場合は、ローカル LU の送信バッファーの内容がパートナー・プログラムに送信され、会話が受信状態に変更されます。

会話の同期レベルが CM_CONFIRM の場合は、LU の送信バッファーの内容と確認要求がパートナー・プログラムに送信されます。確認要求に応答してパートナー・プログラムが Confirmed コールを発行すると、会話は受信状態に変わります。

## 戻りパラメーター

verb の実行後に、CS/AIX は、処理が正常に実行されたかどうかを示すパラメーターと、失敗した場合はその理由を示すパラメーターを戻します。

*return_code*
値は次のとおりです。

**CM_OK** コールは正常に実行されました。

**CM_PROGRAM_PARAMETER_CHECK**
次のいずれかの状態が発生しました。

- *prepare_to_receive_type* パラメーターまたは *conversation_ID* パラメーターに指定された値が有効でない。
- *prepare_to_receive_type* パラメーターが CM_PREP_TO_RECEIVE_CONFIRM に設定されているのに、会話の同期レベルが CM_NONE に設定されている。

**CM_PRODUCT_SPECIFIC_ERROR**
この戻りコードの説明については、191 ページの『付録 B. 共通な戻りコード』を参照してください。

## 発行時の状態

会話は、リセット以外であればどの状態にあっても構いません。

## 状態の変化

状態には変化はありません。

## 使用上の注意

*return_code* が CM_OK でない場合は、*prepare_to_receive_type* 会話特性は変更されません。

---

# Set_Processing_Mode (cmspm)

この関数は Java CPI-C では使用できません。Java CPI-C 関数は常にブロッキング・モードで動作します。すなわちこのモードでは、Java CPI-C 関数は、要求された処理が完了するまでアプリケーションに制御を戻しません。

Set_Processing_Mode コールは、要求された操作が完了したときに後続の CPI-C コールが戻るのか (ブロッキング・モード)、操作が完了しなくても即時に戻るのか (非ブロッキング・モード) を指定します。 Initialize_Conversation コール、または Accept_Conversation コールにより設定されるデフォルトの処理モードは CM_BLOCKING (ブロッキング・モード) です。

AIX、LINUX

会話の処理モードがブロッキングの場合は、この会話で発行された CPI-C コールは即時に戻り、要求された操作がまだ完了していないことを示す戻りコード CM_OPERATION_INCOMPLETE が戻されます。アプリケーションは、この会話に関係ない他の処理を実行するか、または次のコールのいずれかを発行することができます。

- Check_For_Completion。未解決のコール (または他の会話) が完了したかどうかを判別するために発行します。
- Wait_For_Conversation。このコールが完了するまで待つために発行します。
- Cancel_Conversation。未解決のコールを取り消し、会話を割り振り解除するために発行します。

WINDOWS

前述したように Windows アプリケーションは Wait_For_Conversation コールを使用することができます。ただし、非ブロッキング・コールの取り扱いについて推奨される方法は、 Specify_Windows_Handle を使用することです。この関数は、非ブロッキング・コールの前に発行される必要があり、コールの処理が完了したときに CPI-C がメッセージを送信する Windows ハンドルを指定します。アプリケーションは、このメッセージを受信するときにコールの結果を確認し、Wait_For_Conversation を使用しません。 Check_For_Completion は、AIX または Linux システム向けに述べたように、 Windows システムではサポートされません。

未解決のコールが受信コールである場合、Windows アプリケーションは、前述したコールの他に、Request_To_Send、Send_Error、Test_Request_to_Send_Received、または Deallocate コールを発行することができます。詳しくは、 104 ページの『Receive (cmrcv)』を参照してください。

処理モードは、後続のすべての CPI-C コールに影響を与えます。処理モードを変更するには、Set_Processing_Mode コールを再発行します。

## 関数コール

```
void cmspm (
            unsigned char CM_PTR       conversation_ID,
            CM_INT32 CM_PTR            processing_mode,
            CM_RETURN_CODE CM_PTR      return_code
           );
```

## 指定パラメーター

指定パラメーターは次のとおりです。

*conversation_ID*
このパラメーターは会話の ID です。このパラメーターの値は、Initialize_Conversation コール、または Accept_Conversation コールから戻されます。

*processing_mode*
このパラメーターは、後続の CPI-C コールをブロッキング・モードで実行するか非ブロッキング・モードで実行するかを指定します。値は次のとおりです。

**CM_BLOCKING**
後続の CPI-C コールは、操作が完了するまで戻りません。

**CM_NON_BLOCKING**
後続の CPI-C コールは、操作が完了したかどうかに関係なく、操作の開始直後に戻ります。

## 戻りパラメーター

verb の実行後に、CS/AIX は、処理が正常に実行されたかどうかを示すパラメーターと、失敗した場合はその理由を示すパラメーターを戻します。

*return_code*
値は次のとおりです。

**CM_OK** コールは正常に実行されました。

**CM_PROGRAM_PARAMETER_CHECK**
*processing_mode* パラメーターまたは *conversation_ID* パラメーターに指定された値が有効ではありません。

次の戻りコードの説明については、191 ページの『付録 B. 共通な戻りコード』を参照してください。
CM_OPERATION_NOT_ACCEPTED
CM_PRODUCT_SPECIFIC_ERROR

## 発行時の状態

会話は、リセット以外であればどの状態にあっても構いません。

## 状態の変化

状態には変化はありません。

## 使用上の注意

*return_code* が CM_OK でない場合は、*processing_mode* 会話特性は変更されません。

# Set_Receive_Type (cmsrt)

Set_Receive_Type コールは、後続の Receive コールでプログラムがデータを受信する方法を指定します。これは、Initialize_Conversation コール、または Accept_Conversation コールにより設定されたデフォルトの受信タイプを変更します。デフォルトでは、Receive コールの発行時に受信可能なデータがなければ、プログラムはデータが到着するまで待機します。

指定した受信タイプ値は、後続のすべての Receive コールに影響を与えます。値を変更するには、Set_Receive_Type コールを再発行します。

## 関数コール

```
void cmsrt (
            unsigned char CM_PTR       conversation_ID,
            CM_RECEIVE_TYPE CM_PTR     receive_type,
            CM_RETURN_CODE CM_PTR      return_code
           );
```

## Java CPI-C の関数コール

AIX、LINUX

```
public native void cmsrt (
                           byte[]           conversation_ID,
                           CPICReceiveType  receive_type,
                           CPICReturnCode   return_code
                          );
```

## 指定パラメーター

指定パラメーターは次のとおりです。

*conversation_ID*
このパラメーターは会話の ID です。このパラメーターの値は、Initialize_Conversation コール、Initialize_For_Incoming コール、または Accept_Conversation コールから戻されます。

*receive_type*
このパラメーターは、後続の Receive コールでプログラムがデータを受信する方法を指定します。値は次のとおりです。

**CM_RECEIVE_AND_WAIT**
ローカル・プログラムは、パートナー・プログラムから現在受信可能なデータをすべて受信します。現在受信可能なデータがない場合は、ローカル・プログラムはデータが到着するまで待機します。

**CM_RECEIVE_IMMEDIATE**
ローカル・プログラムは、パートナー・プログラムから現在受信可能なデータをすべて受信します。受信可能なデータがない場合は、ローカル・プログラムは待機しません。

## 戻りパラメーター

verb の実行後に、CS/AIX は、処理が正常に実行されたかどうかを示すパラメーターと、失敗した場合はその理由を示すパラメーターを戻します。

*return_code*
値は次のとおりです。

**CM_OK** コールは正常に実行されました。

**CM_PROGRAM_PARAMETER_CHECK**
*conversation_ID* または *receive_type* に指定された値が有効ではありません。

**CM_PRODUCT_SPECIFIC_ERROR**
この戻りコードの説明については、191 ページの『付録 B. 共通な戻りコード』を参照してください。

## 発行時の状態

会話は、リセット以外であればどの状態にあっても構いません。

## 状態の変化

状態には変化はありません。

## 使用上の注意

*return_code* が CM_OK でない場合は、*receive_type* 会話特性は変更されません。

---

# Set_Return_Control (cmsrc)

Set_Return_Control コールは、セッションが使用可能でなかった場合に Allocate コールが即時に戻るのか、セッションが割り振られるまで待機するのかを指定するために、呼び出し側プログラムが発行します。

このコールは、Initialize_Conversation コールにより設定されたデフォルトの戻り制御を変更します。デフォルトでは、CPI-C はセッションが割り振られるまで待機します。Allocate コールの発行後にこのコールを発行することはできません。

セッションの詳細については、21 ページの『第 2 章 CPI-C アプリケーションの作成』を参照してください。

## 関数コール

```
void cmsrc (
            unsigned char CM_PTR         conversation_ID,
            CM_RETURN_CONTROL CM_PTR     return_control,
            CM_RETURN_CODE CM_PTR        return_code
           );
```

## Java CPI-C の関数コール

AIX、LINUX

```
public native void cmsrc (
                           byte[]             conversation_ID,
                           CPICReturnControl  return_control,
                           CPICReturnCode     return_code
                         );
```

## 指定パラメーター

指定パラメーターは次のとおりです。

*conversation_ID*
このパラメーターは会話の ID です。このパラメーターの値は、Initialize_Conversation コールから戻されます。

*return_control*
このパラメーターは、Allocate コールに対して働くローカル LU がローカル・プログラムにいつ制御を戻すのかを指定します。指定できる値は次のとおりです。

**CM_IMMEDIATE**
LU は、使用可能なコンテンション勝者セッションがあればすぐに割り振り、プログラムに制御を戻します。

**CM_WHEN_SESSION_ALLOCATED**
LU は、セッションを割り振るかまたは特定のエラーを検出するまで、プログラムに制御権を戻しません。使用可能なセッションがない場合、プログラムはセッションが使用可能になるまで待機します (セッション限度が 0 の場合は、LU はすぐに制御を戻します)。

## 戻りパラメーター

verb の実行後に、CS/AIX は、処理が正常に実行されたかどうかを示すパラメーターと、失敗した場合はその理由を示すパラメーターを戻します。

*return_code*
値は次のとおりです。

**CM_OK** コールは正常に実行されました。

**CM_PROGRAM_STATE_CHECK**
会話は初期化状態ではありません。

**CM_PROGRAM_PARAMETER_CHECK**
*conversation_ID* または *return_control* に指定された値が有効ではありません。

**CM_PRODUCT_SPECIFIC_ERROR**
この戻りコードの説明については、191 ページの『付録 B. 共通な戻りコード』を参照してください。

## 発行時の状態

会話は初期化状態になっていなければなりません。

## 状態の変化

状態には変化はありません。

## 使用上の注意

*return_code* が CM_OK でない場合は、*return_control* 会話特性は変更されません。

LU がセッションを割り振ることができなかった場合、その通知は Allocate コールで戻されます。

## Set_Send_Type (cmsst)

Set_Send_Type コールは、次の Send_Data コールでデータをどのように送信するかを指定します。これは、Initialize_Conversation コール、または Accept_Conversation コールにより設定された、デフォルトの送信タイプを変更します。デフォルトの送信タイプは CM_BUFFER_DATA であり、これはデータのみを (制御情報なしで) 送信することを示します。

指定した送信タイプ値は、後続のすべての Send_Data コールに影響を与えます。値を変更するには、Set_Send_Type コールを再発行します。

### 関数コール

```
void cmsst (
            unsigned char CM_PTR       conversation_ID,
            CM_SEND_TYPE CM_PTR        send_type,
            CM_RETURN_CODE CM_PTR      return_code
           );
```

### Java CPI-C の関数コール

AIX、LINUX

```
public native void cmsst (
                           byte[]          conversation_ID,
                           CPICSendType    send_type,
                           CPICReturnCode  return_code
                         );
```

### 指定パラメーター

指定パラメーターは次のとおりです。

*conversation_ID*
このパラメーターは会話の ID です。このパラメーターの値は、Initialize_Conversation コール、Initialize_For_Incoming コール、または Accept_Conversation コールから戻されます。

*send_type*
このパラメーターは、後続の Send_Data コールでデータを送信する方法を指定します。値は次のとおりです。

**CM_BUFFER_DATA**
Send_Data コールが指すデータは、バッファーが満ぱいになるかフラッシュされるまで、バッファーに入れられます。

**CM_SEND_AND_FLUSH**
Send_Data コールが指すデータは、すぐに送信されます。これは、

*send_type* が CM_BUFFER_DATA に設定された状態で Send_Data を発行し、続いて Flush を発行するのと同じです。

**CM_SEND_AND_CONFIRM**
データは、確認要求と共にすぐに送信されます。これは、*send_type* が CM_BUFFER_DATA に設定された状態で Send_Data を発行し、続いて Confirm を発行するのと同じです。

**CM_SEND_AND_PREP_TO_RECEIVE**
データは、送信側プログラムの会話状態が受信に変更されているという通知と共に、すぐにパートナー・プログラムに送信されます。これは、*send_type* が CM_BUFFER_DATA に設定された状態で Send_Data を発行し、続いて Prepare_To_Receive を発行するのと同じです。

**CM_SEND_AND_DEALLOCATE**
データは、割り振り解除通知と共にすぐに送信されます。これは、*send_type* が CM_BUFFER_DATA に設定された状態で Send_Data を発行し、続いて Deallocate を発行するのと同じです。

## 戻りパラメーター

verb の実行後に、CS/AIX は、処理が正常に実行されたかどうかを示すパラメーターと、失敗した場合はその理由を示すパラメーターを戻します。

*return_code*
値は次のとおりです。

**CM_OK** コールは正常に実行されました。

**CM_PROGRAM_PARAMETER_CHECK**
次のいずれかの状態が発生しました。

- *conversation_ID* または *send_type* に指定された値が有効でない。
- *send_type* パラメーターは CM_SEND_AND_CONFIRM に設定されているが、会話の同期レベルが CM_NONE に設定されている。

**CM_PRODUCT_SPECIFIC_ERROR**
この戻りコードの説明については、191 ページの『付録 B. 共通な戻りコード』を参照してください。

## 発行時の状態

会話は、リセット以外であればどの状態にあっても構いません。

## 状態の変化

状態には変化はありません。

## 使用上の注意

*return_code* が CM_OK でない場合は、*send_type* 会話特性は変更されません。

CM_BUFFER_DATA 以外の *send_type* 値を使用すると、Send_Data コールに別の CPI-C コールの機能を組み込むことができるので、発行するコールの数を少なくすることができます。

# Set_Sync_Level (cmssl)

Set_Sync_Level コールは、会話の同期レベルを指定するために、呼び出し側プログラムが発行します。同期レベルは、プログラムが Confirm コールと Confirmed コールを介して処理を同期させるかどうかを決定します。

このコールは、Initialize_Conversation コールにより設定された同期レベルを変更します。デフォルトの同期レベルは、同期しないことを示す CM_NONE です。Allocate コールの発行後にこのコールを発行することはできません。

## 関数コール

```
void cmssl (
            unsigned char CM_PTR       conversation_ID,
            CM_SYNC_LEVEL CM_PTR       sync_level,
            CM_RETURN_CODE CM_PTR      return_code
           );
```

## Java CPI-C の関数コール

AIX、LINUX

```
public native void cmssl (
                           byte[]          conversation_ID,
                           CPICSyncLevel   sync_level,
                           CPICReturnCode  return_code
                         );
```

## 指定パラメーター

指定パラメーターは次のとおりです。

*conversation_ID*
このパラメーターは会話の ID です。このパラメーターの値は、Initialize_Conversation コールから戻されます。

*sync_level*
このパラメーターは、会話の同期レベルを指定します。値は次のとおりです。

**CM_NONE**
プログラムは確認処理を実行しません。

**CM_CONFIRM**
プログラムは確認処理を実行できます。

インプリメントされた CPI-C の種類によっては、第三のレベルである同期点が提供される場合もありますが、これは CS/AIX CPI-C ではサポートされていません。

## 戻りパラメーター

verb の実行後に、CS/AIX は、処理が正常に実行されたかどうかを示すパラメーターと、失敗した場合はその理由を示すパラメーターを戻します。

*return_code*
値は次のとおりです。

**CM_OK** コールは正常に実行されました。

**CM_PROGRAM_STATE_CHECK**
会話は初期化状態ではありません。

**CM_PROGRAM_PARAMETER_CHECK**
次のいずれかの状態が発生しました。

- *conversation_ID* または *sync_level* に指定された値が有効でない。
- *sync_level* パラメーターに CM_NONE が指定されている (ただし、次のいずれかに該当する場合)。
  - *send_type* パラメーターが CM_SEND_AND_CONFIRM に設定されている。
  - *prepare_to_receive_type* パラメーターが CM_PREP_TO_RECEIVE_CONFIRM に設定されている。
  - *deallocate_type* パラメーターが CM_DEALLOCATE_CONFIRM に設定されている。

**CM_PRODUCT_SPECIFIC_ERROR**
この戻りコードの説明については、191 ページの『付録 B. 共通な戻りコード』を参照してください。

## 発行時の状態

会話は初期化状態になっていなければなりません。

## 状態の変化

状態には変化はありません。

## 使用上の注意

*return_code* が CM_OK でない場合は、*sync_level* 会話特性は変更されません。

---

# Set_TP_Name (cmstpn)

Set_TP_Name コールは、パートナー・プログラム名を指定するために呼び出し側プログラムが発行します。このコールは、Initialize_Conversation コールの発行時にサイド情報から得られたパートナー・プログラム名を変更します。Allocate コールの発行後にこのコールを発行することはできません。このコールを発行しても、サイド情報自体には影響はありません。

このコールは、Specify_Local_TP_Name とは働きが異なります。Set_TP_Name は、呼び出し側プログラムが、会話を割り振りたいプログラムの名前を指定するために発行します。Specify_Local_TP_Name は、呼び出し対象プログラムが、着呼 Allocate 要求を受け付けようとするプログラムの名前を指定するために発行します。

## 関数コール

```
void cmstpn (
             unsigned char CM_PTR      conversation_ID,
             unsigned char CM_PTR      TP_name,
             CM_INT32 CM_PTR           TP_name_length,
             CM_RETURN_CODE CM_PTR     return_code
            );
```

## Java CPI-C の関数コール

AIX、LINUX

```
public native void cmstpn (
                            byte[]           conversation_ID,
                            byte[]           TP_name,
                            CPICLength       TP_name_length,
                            CPICReturnCode   return_code
                           );
```

## 指定パラメーター

指定パラメーターは次のとおりです。

*conversation_ID*

このパラメーターは会話の ID です。このパラメーターの値は、Initialize_Conversation コールから戻されます。

*TP_name*

このパラメーターは、パートナー・プログラム名の開始アドレスを示します。プログラム名には、最大 64 文字を使用できます。使用できる文字は次のとおりです。

- 大文字および小文字の英字
- 数字 0〜9、および . (ピリオド)
- 特殊文字 < > ( ) + - & *; / , % _ ? : ' = " ( パートナー・プログラムが CPI-C プログラムのときのみ有効) $ # @ (パートナー・プログラムが APPC プログラムのときのみ有効)

Set_TP_Name コールに SNA サービス TP の名前を指定することはできません。SNA サービス TP 名には、このコールには使用できない文字が含まれています。ただしこれは、Set_CPIC_Side_Information コールを使用して指定できます。

漢字などの 2 バイト文字セットはサポートされていません。

*TP_name_length*

このパラメーターは、パートナー・プログラム名の長さを示します。

この値の範囲は 1〜64 です。

## 戻りパラメーター

verb の実行後に、CS/AIX は次のパラメーターを戻します。

*return_code*

値は次のとおりです。

**CM_OK** コールは正常に実行されました。

**CM_PROGRAM_STATE_CHECK**
会話は初期化状態ではありません。

**CM_PROGRAM_PARAMETER_CHECK**
次のいずれかの状態が発生しました。

- *conversation_ID* に指定された値が有効でない。
- *TP_name_length* に指定された値が範囲外である。

**CM_PRODUCT_SPECIFIC_ERROR**
この戻りコードの説明については、191 ページの『付録 B. 共通な戻りコード』を参照してください。

## 発行時の状態

会話は初期化状態になっていなければなりません。

## 状態の変化

状態には変化はありません。

## 使用上の注意

*return_code* が CM_OK でない場合は、*TP_name* 会話特性は変更されません。

---

# Specify_Local_TP_Name (cmsltp)

Specify_Local_TP_Name コールは、着呼 Allocate 要求を受け付けるローカル TP 名を指定するために、CPI-C アプリケーションが発行します。

このコールを使用する代わりに、APPCTPN 環境変数を使用するなどの他の方法でローカル TP 名を設定できます。ローカル TP 名の設定方法の詳細については、37 ページの『ローカル TP 名の指定』を参照してください。Specify_Local_TP_Name コールが必要なのは、1 つのアプリケーションが複数のローカル TP 名を対象とした着呼 Allocate を受け付けたい場合です。名前が 1 つだけの場合は APPCTPN を使用できますが、追加の名前を指定するには、このコールを使用する必要があります (着呼 Allocate 要求を受け付けるための Accept_Conversation コール、または Accept_Incoming コールを発行したあとで、Extract_TP_Name を使用して、どの名前がパートナー・アプリケーションで指定されているかを判別できます)。

このコールは Set_TP_Name とは働きが異なります。Set_TP_Name は、呼び出し側プログラムが、会話を割り振りたいプログラムの名前を指定するために発行します。Specify_Local_TP_Name は、呼び出し対象プログラムが、着呼 Allocate 要求を受け付けようとするプログラムの名前を指定するために発行します。

## 関数コール

```
void cmsltp (
             unsigned char CM_PTR      TP_name,
             CM_INT32 CM_PTR           TP_name_length,
             CM_RETURN_CODE CM_PTR     return_code
            );
```

## Java CPI-C の関数コール

AIX、LINUX

```
public native void cmsltp (
                           byte[]          TP_name,
                           CPICLength      TP_name_length,
                           CPICReturnCode  return_code
                          );
```

## 指定パラメーター

指定パラメーターは次のとおりです。

*TP_name*

このパラメーターは、TP 名の開始アドレスを示します。名前には、最大 64 文字を指定できます。使用できる文字は次のとおりです。

- 大文字および小文字の英字
- 数字 0〜9
- 特殊文字 . < > ( ) + - & *; / , % _ ? : ' = "

Specify_Local_TP_Name コールに SNA サービス TP の名前を指定することはできません。SNA サービス TP 名には、このコールには使用できない文字が含まれています。

漢字などの 2 バイト文字セットはサポートされていません。

*TP_name_length*

このパラメーターは名前の長さを示します。

この値の範囲は 1〜64 です。

## 戻りパラメーター

verb の実行後に、CS/AIX は次のパラメーターを戻します。

*return_code*

値は次のとおりです。

**CM_OK** コールは正常に実行されました。

**CM_PROGRAM_PARAMETER_CHECK**

次のいずれかの状態が発生しました。

- *TP_name* に指定された値が予約名である、または 1 つ以上の有効でない文字を含んでいる。

- *TP_name_length* に指定された値が範囲外である。

**CM_PRODUCT_SPECIFIC_ERROR**
この戻りコードの説明については、191 ページの『付録 B. 共通な戻りコード』を参照してください。

## 発行時の状態

このコールは、会話には関連付けられていません。

## 状態の変化

状態には変化はありません。

## 使用上の注意

*return_code* が CM_OK でない場合は、このプログラムが着呼 Allocate 要求を受け付ける TP 名は変更されません。

このコールの発行時に未解決の Accept_Incoming コールが存在すると、このコールで指定された名前を対象とした Allocate が着呼しても受け付けられません。ただし、後続の Accept_Conversation コール、または Accept_Incoming コールは、この名前を対象とした着呼 Allocate を受け付けます。

---

# Specify_Windows_Handle (xchwnd)

WINDOWS

Specify_Windows_Handle コールは、CPI-C アプリケーションにより発行され、非ブロッキング CPI-C 関数が完了するたびに CPI-C がメッセージを送信する先の Windows ハンドルを指定します。これにより、関数の完了を待つ Wait_For_Conversation (AIX または Linux システムの場合) の使用に代わる手段がもたらされます。Windows システム用の新規の CPI-C アプリケーションを作成している場合、この方法を使用し、Wait_For_Conversation を使用しないでください。

非ブロッキング・コールを使用して、メッセージを受信してコールの完了を示すには、アプリケーションは、非ブロッキング・コールを発行する前に、以下のコールを発行する必要があります。

- RegisterWindowMessage。CPI-C が非ブロッキング CPI-C 関数の完了を知らせるメッセージに使用するメッセージ ID を取得する。このコールは、標準の Windows 関数コールで、CPI-C に固有ではありません。関数についての詳細は、Windows の文書を参照してください。アプリケーションは、値 WIN_CPIC_ASYNC_COMPLETE_MESSAGE を関数に渡します。以下に述べるように、戻り値はメッセージ ID です。(後続の CPI-C コールの前に再度このコールを発行する必要はありません。戻り値は、アプリケーションが発行したすべてのコールのものと同じです。)
- Set_Processing_Mode。会話の処理モードをCM_NON_BLOCKINGに設定する。
- Specify_Windows_Handle。完了メッセージの送信先のハンドルを指定する。

非ブロッキング CPI-C 関数が完了するたびに、CPI-C は Specify_Windows_Handle コールで指定されたウィンドウ・ハンドルにメッセージを通知します。メッセージのフォーマットは、次の通りです。

- メッセージ ID は、RegisterWindowMessage コールから戻された値。
- *lParam* 引数には、完了した CPI-C コールの会話 ID が含まれる。
- *wParam* 引数には、完了した CPI-C コールからの会話 *return_code* パラメーターが含まれる。このパラメーターに可能な値は、個々のコールによって異なります。

## 関数コール

```
void xchwnd (
             HWND                       hwnd,
             CM_RETURN_CODE CM_PTR      return_code
            );
```

## 指定パラメーター

指定パラメーターは次のとおりです。

*hwnd* CPI-C が、非ブロッキング関数が完了したことを知らせるメッセージの通知に使用するウィンドウ・ハンドル。

## 戻りパラメーター

verb の実行後に、CS/AIX は、処理が正常に実行されたかどうかを示すパラメーターと、失敗した場合はその理由を示すパラメーターを戻します。

*return_code*

値は次のとおりです。

**CM_OK** コールは正常に実行されました。

**CM_PROGRAM_PARAMETER_CHECK**
指定されたパラメーターは、無効な Windows ハンドルです。

**CM_PRODUCT_SPECIFIC_ERROR**
この戻りコードの説明については、191 ページの『付録 B. 共通な戻りコード』を参照してください。

## 発行時の状態

このコールは、会話には関連付けられていません。

## 状態の変化

このコールに付随する状態変更はありません。

CPI-C が非ブロッキング・コールが完了したことを知らせるメッセージを送信する際、状態変更は、完了した関数とその戻りコードによって異なります。

# Test_Request_to_Send_Received (cmtrts)

Test_Request_to_Send_Received コールは、パートナー・プログラムから送信要求通知を受信したかどうかを判別します。

## 関数コール

```
void cmtrts (
             unsigned char CM_PTR                 conversation_ID,
             CM_Request_to_Send_Received CM_PTR   request_to_send_received,
             CM_RETURN_CODE CM_PTR                return_code
            );
```

## Java CPI-C の関数コール

AIX、LINUX

```
public native void cmtrts (
                           byte[]                          conversation_ID,
                           CPICControlInformationReceived  request_to_send_received,
                           CPICReturnCode                  return_code
                          );
```

## 指定パラメーター

指定パラメーターは次のとおりです。

*conversation_ID*

このパラメーターは会話の ID です。

このパラメーターの値は、Initialize_Conversation コール、Initialize_For_Incoming コール、または Accept_Conversation コールから戻されます。

## 戻りパラメーター

verb の実行後に、CS/AIX は、処理が正常に実行されたかどうかを示すパラメーターと、失敗した場合はその理由を示すパラメーターを戻します。

*request_to_send_received*

このパラメーターは、送信要求受信インディケーターです。値は次のとおりです。

**CM_REQ_TO_SEND_RECEIVED**

パートナー・プログラムが Request_To_Send コールを発行しました。このコールは、ローカル・プログラムに、会話を受信状態に変更するよう要求します。

**CM_REQ_TO_SEND_NOT_RECEIVED**

パートナー・プログラムは Request_To_Send コールを発行していません。

*return_code* パラメーターに CM_OK 以外の値が含まれている場合は、この値は無関係です。

*return_code*
値は次のとおりです。

**CM_OK** コールは正常に実行されました。

**CM_PROGRAM_PARAMETER_CHECK**
*conversation_ID* に指定された値は無効です。

**CM_PROGRAM_STATE_CHECK**
会話が無効な状態になっています。

次の戻りコードの説明については、191 ページの『付録 B. 共通な戻りコード』を参照してください。
CM_OPERATION_NOT_ACCEPTED
CM_PRODUCT_SPECIFIC_ERROR

### 発行時の状態

会話は、受信、送信、送信 - 保留、または保留 - 通知状態である必要があります。

### 状態の変化

状態には変化はありません。

## Wait_For_Conversation (cmwait)

この関数は Java CPI-C では使用できません。Java CPI-C 関数は常にブロッキング・モードで動作します。すなわちこのモードでは、Java CPI-C 関数は、要求された処理が完了するまでアプリケーションに制御を戻しません。

Wait_For_Conversation コールは、CM_OPERATION_INCOMPLETE を戻した直前の CPI-C コールが完了するまで待機します。

Wait_For_Conversation が発行されたときに直前のコールの処理がすでに終了していた場合は、このコールは即時に戻ります。それ以外の場合は、不完全操作を CPI-C が終了するまで、このコールはブロックされます。アプリケーションが複数の会話に関係している場合は、このコールはすべての会話で待機し、その中のいずれか 1 つでコールが完了すると同時に戻ります。

WINDOWS

Windows システム用に作成された新規のアプリケーションは、Wait_For_Conversation を使用せずに、 Specify_Windows_Handle を使用して、非ブロッキング・コールの結果を取得する必要があります。 168 ページの『Specify_Windows_Handle (xchwnd)』を参照してください。Wait_For_Conversation コールは、他の CPI-C インプリメンテーションとの互換性の確保のために備えられたものですが、 Windows アプリケーションによる使用は勧められません。

特に、アプリケーションが非ブロッキング・モードで受信コールを発行し、受信が未解決の間に同一の会話上で非ブロッキング・モードで他のコールを発行する場

合、 Specify_Windows_Handle を使用しなければなりません。同一の会話上で複数のコールが未解決である間、Wait_For_Conversation を発行してはなりません。この状況では、Wait_For_Conversation の結果が未定義になります。

## 関数コール

```
void cmwait (
             unsigned char CM_PTR      conversation_ID,
             CM_INT32 CM_PTR           conversation_return_code,
             CM_RETURN_CODE CM_PTR     return_code
            );
```

## 指定パラメーター

このコールには、指定パラメーターはありません。

## 戻りパラメーター

verb の実行後に、CS/AIX は、処理が正常に実行されたかどうかを示すパラメーターと、失敗した場合はその理由を示すパラメーターを戻します。

*conversation_ID*
このパラメーターは、未解決のコールが完了した会話の ID です。

*conversation_return_code*
このパラメーターは、完了したコール (直前に CM_OPERATION_INCOMPLETE を戻したコール) からの戻りコードです。このパラメーターに戻される値は、未解決になっていたコールによって異なります。詳細については、個別のコールの説明を参照してください。

*return_code* パラメーターに CM_OK 以外の値が含まれている場合は、この値は無関係です。

*return_code*
値は次のとおりです。

**CM_OK** Wait_For_Conversation コールは正常に実行されました。
*conversation_return_code* パラメーターは、直前の不完全操作が正常に完了したかどうかを示します。

**CM_PROGRAM_STATE_CHECK**
未解決の不完全操作はありません。

**CM_PRODUCT_SPECIFIC_ERROR**
この戻りコードの説明については、 191 ページの『付録 B. 共通な戻りコード』を参照してください。

WINDOWS

**CM_SYSTEM_EVENT**
このコールは、直前の CPI-C コールの完了ではなく、オペレーティング・システムのイベントにより、完了しました。

## 発行時の状態

このコールは、特定の会話に関連付けられていません。したがって、会話状態は無関係です。ただしアプリケーションは、未解決の不完全操作を持つ会話を 1 つ以上持っていなければなりません。

## 状態の変化

*return_code* が CM_OK に設定されている場合は、状態変化は完了した未解決のコール、およびそのコールからの戻りコード (このコールの *conversation_return_code* パラメーター) に応じて異なります。詳細については、個別のコールの説明を参照してください。*return_code* が CM_OK に設定されていない場合は、状態は変わりません。

## 使用上の注意

このコールは、プログラムの現行コンテキストは変更しません (完了した未解決の操作が、Accept_Incoming などのように、通常ならばコンテキストを変更する操作の場合でも、変更されません)。必要な場合には、プログラムは、このコールで戻された *conversation_ID* について Extract_Conversation_Context を使用して会話コンテキストの値を入手し、Set_Conversation_Context を使用して現行コンテキストをこの値に設定できます。

直前の未解決のコールが 1 つも完了していない場合は、どれか 1 つが完了するまで、このコールはブロックされます (そしてアプリケーションの処理は中断されます)。

AIX、LINUX

ブロッキングを避けて完了したコールの有無をチェックするために、アプリケーションは、Check_For_Completion (これは常に即時に戻ります) を使用してコールが完了したかどうかを判別し、Check_For_Completion によってコールの完了が示されたときにだけ (したがって Wait_For_Conversation が即時に戻るときにだけ) Wait_For_Conversation を呼び出せます。

(異なる会話で) 複数の未解決のコールがある場合は、各 Wait_For_Conversation コールから 1 つずつ未解決コールが戻されます。 Wait_For_Conversation を発行したあとで、アプリケーションは、Check_For_Completion を発行して、他に完了したコールがあるかどうかを調べることができます。

WINDOWS

前述したように Windows アプリケーションは Wait_For_Conversation を使用することができます。ただし、非ブロッキング・コールの取り扱いについて推奨される方法は、 Specify_Windows_Handle を使用することです。この関数は、非ブロッキング・コールの前に発行される必要があり、コールの処理が完了したときに CPI-C が

メッセージを送信する Windows ハンドルを指定します。アプリケーションは、このメッセージを受信するときにコールの結果を確認し、Wait_For_Conversation を使用しません。 Check_For_Completion は、AIX または Linux システム向けに述べたように、 Windows システムではサポートされません。

---

## WinCPICCleanup

WINDOWS

アプリケーションは、CPI-C コールの発行を完了後、この関数を使用して Windows CPI-C ユーザーとして登録を抹消します。

### 関数コール

```
BOOL WINAPI WinCPICCleanup (void);
```

### 指定パラメーター

このコールには、指定パラメーターはありません。

### 戻り値

関数の戻り値は、以下のうちいずれかです。

**TRUE** アプリケーションは正常に登録抹消されました。

**FALSE** コールの処理中にエラーが発生しました。アプリケーションは登録抹消されませんでした。ログ・ファイルを確認して、エラーの原因を示すメッセージを参照してください。

---

## WinCPICIsBlocking

WINDOWS

アプリケーションはこの関数を使用して、未解決のブロッキング CPI-C コール (CM_BLOCKING に設定された会話の処理モードで発行されたコール) があるかどうかを確認します。ブロッキング・コールについての詳細は、 46 ページの『Windows に関する考慮事項』を参照してください。

### 関数コール

```
BOOL WINAPI WinCPICIsBlocking (void);
```

## 指定パラメーター

このコールには、指定パラメーターはありません。

## 戻り値

関数の戻り値は、以下のうちいずれかです。

**TRUE** 未解決のブロッキング CPI-C コールがあります。必要に応じて、アプリケーションは Cancel_Conversation または Deallocate を使用して、コールを取り消し、会話を終了させることができます。

**FALSE** 未解決のブロッキング CPI-C コールはありません。

# WinCPICSetBlockingHook

WINDOWS

アプリケーションは、このコールを使用して、CPI-C がデフォルトのブロッキング関数の代わりに使用する、固有のブロッキング関数を指定します。ブロッキング関数がどのように動作するかということと、実行する必要のある関数についての詳細は、 48 ページの『ブロッキング・コール』を参照してください。

## 関数コール

```
FARPROC WINAPI WinCPICSetBlockingHook (FARPROC lpBlockFunc);
```

## 指定パラメーター

指定パラメーターは次のとおりです。

*lpBlockFunc*

アプリケーションのブロッキング関数のプロシージャー・インスタンス・アドレス。アプリケーションは、MakeProcInstance コールを使用して、このアドレスを取得する必要があります。詳細は、Windows の文書を参照してください。

## 戻り値

戻り値は、直前のブロッキング関数のプロシージャー・インスタンス・アドレスです。アプリケーションが複数のブロッキング関数を使用し、後で直前のブロッキング関数を復元する必要のある場合、このアドレスを保管する必要があります。保管した値を使用し、再度 WinCPICSetBlockingHook を発行して、直前のブロッキング関数を復元することができます。使用しているブロッキング関数が 1 つだけで、直前の値を復元する必要のない場合、このコールからの戻り値を無視することができます。

### 使用法

新規のブロッキング関数は、アプリケーションが以下のコールのいずれかを発行するまで、有効です。

- WinCPICSetBlockingHook (異なるプロシージャー・インスタンス・アドレスを持つ)。新規のブロッキング関数を指定するか、直前のブロッキング関数を復元する。
- WinCPICUnhookBlockingHook (後述)。現在のブロッキング関数の使用を停止し、デフォルトのブロッキング関数に戻す。

## WinCPICStartup

WINDOWS

アプリケーションはこの関数を使用して、Windows CPI-C ユーザーとして登録し、CPI-C ソフトウェアがアプリケーションに必要な Windows CPI-C バージョンをサポートしているかどうかを判別します。

### 関数コール

```
int WINAPI WinCPICStartup (
                            WORD              wVersionRequired;
                            LPWCPICDATA       lpData;
                           )
typedef struct
{
  WORD        wVersion;
  char        szDescription[128];
} WCPICDATA;
```

### 指定パラメーター

指定パラメーターは次のとおりです。

*wVersionRequired*

アプリケーションに必要な Windows CPI-C のバージョン。CS/AIX は、バージョン 1.0 をサポートしています。

このパラメーターの下位バイトでメジャーなバージョン番号を、上位バイトでマイナーなバージョン番号を指定します。たとえば、次のように指定します。

| バージョン | wVersionRequired |
|---|---|
| 1.0 | 0x0001 |
| 1.1 | 0x0101 |
| 2.0 | 0x0002 |

アプリケーションが複数のバージョンを使用する場合、使用できる最上位のバージョンを指定する必要があります。

## 戻り値

関数の戻り値は、以下のうちいずれかです。

**0 (ゼロ)**
アプリケーションは正常に登録され、Windows CPI-C ソフトウェアは、アプリケーションにより指定されたバージョン番号、あるいはその下位バージョンのいずれかをサポートしています。アプリケーションは、WCPICDATA 構造体 (後述の説明を参照) にあるバージョン番号をチェックして、それが十分上位であるかを確認する必要があります。

**WCPICVERNOTSUPPORTED**
アプリケーションに指定されたバージョン番号が、Windows CPI-C ソフトウェアにサポートされる最下位バージョンより下位でした。 アプリケーションは登録されませんでした。

**WCPICSYSNOTREADY**
CS/AIX ソフトウェアが開始されなかったか、ローカル・ノードがアクティブではありません。アプリケーションは登録されませんでした。

WinCPICStartup の戻り値が 0 (ゼロ) である場合、 WCPICDATA 構造体には、Windows CPI-C ソフトウェアによるサポート情報が含まれます。戻り値がゼロ以外の値である場合、この構造体の内容は未定義で、アプリケーションは構造体の内容をチェックする必要はありません。この構造体にあるパラメーターは次のとおりです。

*wVersion*
*wVersionRequired* パラメーターと同一形式で、ソフトウェアがサポートする Windows CPI-C のバージョン番号 (後述の説明を参照)。 CS/AIX は、バージョン 1.0 をサポートしています。

ソフトウェアが要求されるバージョン番号をサポートする場合、このパラメーターは *wVersionRequired* パラメーターと同一の値に設定されます。そうでない場合、ソフトウェアがサポートする最上位のバージョンに設定されますが、これは、アプリケーションに指定されるバージョン番号より下位になります。アプリケーションは戻り値を検査し、以下のようなアクションを行います。

- 戻されたバージョン番号が要求されたバージョン番号と同一である場合、アプリケーションはこの Windows CPI-C インプリメンテーションを使用することができる。
- 戻されたバージョン番号が要求されたバージョン番号より下位である場合、アプリケーションはこの Windows CPI-C インプリメンテーションを使用しますが、要求されたバージョン番号にサポートされない機能を使用することはできません。下位のバージョンでは使用できない機能が要求されるために、これを順守できない場合、初期化を失敗し、CPI-C コールを発行することができなくなります。

*szDescription*
Windows CPI-C ソフトウェアを記述するテキスト・ストリング。

## WinCPICUnhookBlockingHook

WINDOWS

アプリケーションは、このコールを使用して、これより前に WinCPICSetBlockingHook を使用して指定した、固有のブロッキング関数を除去し、CPI-C のデフォルトのブロッキング関数を使用するように戻します。

### 関数コール

```
BOOL WINAPI WinCPICUnhookBlockingHook (void);
```

### 指定パラメーター

このコールには、指定パラメーターはありません。

### 戻り値

戻り値は、以下のうちいずれかです。

**TRUE** ブロッキング機能は、正常に除去されました。今後、ブロッキング・コールは、デフォルトのブロッキング関数を使用します。

**FALSE** コールが正常に完了しませんでした。

## WinCPICSetEvent

WINDOWS

アプリケーションはこの関数を使用して、イベント・ハンドルを指定された会話の verb 完了と関連付けます。

### 関数コール

```
VOID WINAPI WinCPICSetEvent (
                             unsigned char CM_PTR  conversation_ID,
                             HANDLE CM_PTR         event_handle,
                             CM_INT32 CM_PTR       return_code
                            );
```

### 指定パラメーター

指定パラメーターは次のとおりです。

*conversation_ID*
これは、このイベントが使用された会話の ID です。このパラメーターは、最初の Accept_Conversation コールから戻されます。

*event_handle*
これは、会話上の非同期の verb が完了するときにクリアされるイベントの

ハンドルです。このパラメーターにより、既に定義済みのイベントを置き換えることも、既に定義済みのイベントを除去することもできます (パラメーターとして NULL にすることによる)。

## 戻りパラメーター

*return_code*
値は次のとおりです。

**CM_OK** WinCPICSetEvent 関数は正常に実行されました。

**CM_PROGRAM_PARAMETER_CHECK**
この関数に渡された 1 つ以上のパラメーターが無効です。

**CM_OPERATION_NOT_ACCEPTED**
この値は、この会話上の直前の操作が不完全で、 WinCPICSetEvent コールが受け入れられなかったことを示します。

## 使用上の注意

verb が非ブロッキングの会話で発行されると、それが非同期に完了する場合には CM_OPERATION_INCOMPLETE を戻します。イベントがその会話に登録済みである場合、アプリケーションは、WaitForSingleObject または WaitForMultipleObjects を呼び出して、verb の完了が通知されるようにすることができます。verb が完了した時、アプリケーションは Wait_for_Conversation を呼び出して、非同期の verb の戻りコードを判別する必要があります。

アプリケーションは、必ずイベントをリセットしなければなりません。

---

# WinCPICExtractEvent

WINDOWS

アプリケーションはこの関数を使用して、CPI-C 会話に使用されるイベント・ハンドルを決定します。

## 関数コール

```
VOID WINAPI WinCPICExtractEvent (
                                 unsigned char CM_PTR  conversation_ID,
                                 HANDLE CM_PTR         event_handle,
                                 CM_INT32 CM_PTR       return_code
                                );
```

## 指定パラメーター

この関数の指定パラメーターは次の通りです。

*conversation_ID*
これは、このイベントが使用された会話の ID です。このパラメーターは、最初の Accept_Conversation コールから戻されます。

## 戻りパラメーター

*event_handle*
これは、この会話に使用されるイベントのハンドルです。ハンドルが登録されていなかった場合、このパラメーターはヌル値を戻します。

*return_code*
値は次のとおりです。

**CM_OK** WinCPICExtractEvent 関数は正常に実行されました。

**CM_PROGRAM_PARAMETER_CHECK**
この関数に渡された 1 つ以上のパラメーターが無効です。

## 使用上の注意

verb が非ブロッキングの会話で発行されると、それが非同期に完了する場合には CM_OPERATION_INCOMPLETE を戻します。イベントがその会話に登録済みである場合、アプリケーションは、WaitForSingleObject または WaitForMultipleObjects を呼び出して、verb の完了が通知されるようにすることができます。WinCPICExtractEvent により、CPI-C アプリケーションはこのイベント・ハンドルを判別することができます。 verb が完了した時、アプリケーションは Wait_for_Conversation を呼び出して、非同期の verb の戻りコードを判別する必要があります。

Cancel_Conversation 関数を呼び出して、操作と会話を取り消すことができます。

イベントが登録されなかった場合、非同期の verb は、アプリケーションが CPI-C ライブラリーに登録されたウィンドウ宛のメッセージを通知することにより完了します。

# 第 4 章 サンプル CPI-C トランザクション・プログラム

この章では、CS/AIX のサンプル CPI-C トランザクション・プログラムについて説明します。このサンプル・プログラムは、AIX または Linux アプリケーションにおける CPI-C コールの使用法を例示します。 Java アプリケーションにおける CPI-C コールの使い方については、185 ページの『第 5 章 サンプルの Java CPI-C トランザクション・プログラム』を参照してください。

ここでは、次の事項について説明します。

- 2 つのプログラムの処理の概要
- それぞれのプログラムの疑似コード
- 2 つのプログラムをコンパイル、リンク、および実行するための説明

## 処理の概要

この章に示すプログラムを使用すると、別のシステムにあるファイルをブラウズできます。一度に 1 つのデータ・ブロックが、16 進形式および文字形式で表示されます。各ブロックのあとで、次のブロックまたは直前のブロックを要求することも、プログラムを終了することもできます。

CSAMPLE1 (呼び出し側プログラム) は、ファイル名を CSAMPLE2 (呼び出し対象プログラム) に送信します。CSAMPLE2 は、そのファイルを見つけると、先頭のブロックを CSAMPLE1 に戻します。見つからなかった場合は、会話の割り振りを解除し終了します。

CSAMPLE1 は、ブロックを受信すると、そのブロックを画面に表示し、ユーザーが F (進む)、B (戻る)、または Q (終了) を入力するまで待ちます。ユーザーが「進む」または「戻る」を選択すると、CSAMPLE1 がその要求を CSAMPLE2 に送信し、CSAMPLE2 は該当のブロックを送り返します。ユーザーが「終了」オプションを選択するまでこの処理が続きます。「終了」を選択すると、CSAMPLE1 により会話の割り振りが解除され、両方のプログラムが終了します。

ユーザーが次のブロックを要求し、CSAMPLE2 が最後のブロックを送信した場合は、CSAMPLE2 はファイルの先頭に折り返します。同様に、ユーザーが直前のブロックを要求し、先頭のブロックが表示されている場合は、CSAMPLE2 は折り返して最後のブロックを送信します。

どちらのプログラムも、エラーの回復処理は行いません。CPI-C からエラー戻りコードを受け取ると、プログラムが終了し、状況を説明するメッセージが表示されます。

## 疑似コード

この節には、トランザクション・プログラム CSAMPLE1 と CSAMPLE2 の疑似コードを収録しています。

サンプル・プログラムは、**csample1.c** および **csample2.c** として、ディレクトリー **/usr/lib/sna/samples** (AIX) または **/opt/ibm/sna/samples** (Linux) にあります。

## CSAMPLE1 (呼び出し側プログラム)

CSAMPLE1 (呼び出し側プログラム) の疑似コードは次のとおりです。

```
initialize
allocate
send data (data = filename)
do while no error and prompt not Q
    receive
    if data block received
        display data block
    else if permission to send received
        get user prompt (F, B, or Q)
        if prompt = F or B       /* Not Q */
            send data (data = prompt)
        endif
    endif
end do
deallocate
```

## CSAMPLE2 (呼び出し対象 TP)

CSAMPLE2 (呼び出し対象 TP) の疑似コードは次のとおりです。

```
initialize
do while conversing
    receive
    if data received
        if first time (data = filename)
            open file
            if file not found
                deallocate
                set conversing false
            endif
        else (data = prompt)
            read and store prompt
        endif
        if (conversing)
            read file block
            send data (file block)
        endif
    else if deallocate received
        set conversing false
    endif
 end while conversing
close file
```

# TP のテスト

CSAMPLE1 と CSAMPLE2 のソース・コードを調べたあとで、プログラムをテストできます。

CPI-C は通常は別個のコンピューター上のプログラム間の通信に使用されますが、テストを目的として、2 つのプログラムを同じコンピューターで実行する方法も場合によっては便利です。

AIX または Linux システムでプログラムをコンパイルしてリンクするには、次のステップを実行します。

1. 2 つのファイル **csample1.c** と **csample2.c** を、ディレクトリー **/usr/lib/sna/samples** (AIX) または **/opt/ibm/sna/samples** (Linux) からプライベート・ディレクトリーにコピーする。
2. AIX でプログラムをコンパイルしてリンクするには、次のコマンドを使用します。

```
cc -o csample1 -I /usr/include/sna -bimport:/usr/lib/sna/cpic_r.exp csample1.c
cc -o csample2 -I /usr/include/sna -bimport:/usr/lib/sna/cpic_r.exp csample2.c
```

   Linux でプログラムをコンパイルしてリンクするには、次のコマンドを使用します。

```
gcc -o csample1 -I /opt/ibm/sna/include -L /opt/ibm/sna/lib -lcpic -lappc -lsna_r -lpLiS -lpthread csample1.c
gcc -o csample2 -I /opt/ibm/sna/include -L /opt/ibm/sna/lib -lcpic -lappc -lsna_r -lpLiS -lpthread csample2.c
```

これらのプログラムを実行するには、次のステップを実行します。これらのステップの一部では、CS/AIX の構成が更新されます。通常この更新は、システム管理者が行います。

これらのプログラムは同じコンピューターで実行することも、あるいは別々のコンピューターで実行することもできます。次のステップで、「コンパイル用コンピューター」は、呼び出し側プログラム CSAMPLE1 を実行するためのコンピューターで、「ターゲット・コンピューター」は、呼び出し対象プログラム CSAMPLE2 を実行するためのコンピューターです。

1. 別々のコンピューターでプログラムを実行する場合は、コンパイル用コンピューターとターゲット・コンピューター間の CP-CP セッションをサポートするための通信リンクを構成します。詳しくは、「*Communications Server for AIX 管理ガイド*」を参照してください。
2. モード名 LOCMODE を使用してモードを構成します。
3. CSAMPLE1 (呼び出し側プログラム) のための論理装置 (LU) をコンパイル用コンピューターに構成します。 LU 名および LU 別名として、両方に TPLU1 を指定してください。その他のパラメーターについては、デフォルト値のままにしておきます。
4. コンパイル用コンピューターにシンボリック宛先名を構成します。次のようにしてください。
   - 「*Name (名前)*」に、CPICTEST を指定します。
   - 「*Local LU (ローカル LU)*」に「*Local LU alias (ローカル LU 別名)*」を選択し、 LU 別名として TPLU1 を指定します。
   - 「*Partner LU (パートナー LU)*」に、完全修飾名 *netname*.**TPLU2** を指定します。ここで、*netname* は、ターゲット・コンピューターの SNA ネットワーク名です。
   - 「*Mode (モード)*」に、LOCMODE を指定します。
   - 「*Partner TP (パートナー TP)*」に、TPNAME2 を指定します。

   その他のパラメーターは、デフォルト値のままにしておきます。

5. CSAMPLE2 (呼び出し対象プログラム) のための LU をターゲット・コンピューターに構成します。LU 名および LU 別名として、両方に TPLU2 を指定してください。その他のパラメーターについては、デフォルト値のままにしておきます。
6. 呼び出し対象 TP を、ターゲット・コンピューターの CS/AIX の呼び出し可能な TP データ・ファイル内に構成します。 詳しくは、「*Communications Server for AIX 管理ガイド*」を参照してください。
   - 「*TP name (TP 名)*」パラメーターに、TPNAME2 (呼び出し側 TP により指定された名前) を指定します。
   - 「*Full path to TP executable (TP 実行可能ファイルへの絶対パス)*」に、実行可能ファイル **csample2** の絶対パス名を入力します。
   - 「*User ID (ユーザー ID)*」パラメーターに、ターゲット・コンピューターで使用する AIX ユーザー ID を指定します。
   - その他のパラメーターは、デフォルト値のままにしておきます。
7. 呼び出し対象の TP を root の *user_id* で実行する場合は、実行可能ファイルに対する許可をそのように変更します。次のコマンドを使用します。

   **chmod +s csample2**
8. この構成ファイルを使用して CS/AIX ソフトウェアを開始します。
9. 次の環境変数を設定します。
   - APPCLLU を TPLU1 (**csample1** のローカル LU 名) に設定
   - APPCTPN を TPNAME1 に設定
10. 呼び出し側プログラム **csample1** を開始します。このプログラムには、1 つのパラメーター、つまり表示するファイルの (ターゲット・コンピューター上での) 絶対パス名が必要です。たとえば、次のように指定します。

    ```
    csample1 /usr/jim/myfile
    ```
11. 要求対象ファイルのブロックを表示するには、F または B を入力します。呼び出し側プログラムを終了するには、Q を使用します。このとき、呼び出し対象プログラムも同時に終了します。

# 第 5 章 サンプルの Java CPI-C トランザクション・プログラム

AIX、LINUX

この章では、CS/AIX のサンプルの Java CPI-C トランザクション・プログラム **JPing** について説明します。このサンプル・プログラムは、Java アプリケーションにおける CPI-C コールの使用法を示しています。標準の C プログラムにおける CPI-C コールの使用法については、181 ページの『第 4 章 サンプル CPI-C トランザクション・プログラム』を参照してください。

ここでは、次の事項について説明します。

- プログラムの概要
- プログラムのコンパイル、リンク、および実行のための説明

## 概要

サンプルの Java CPI-C プログラム **JPing** (ファイル **/usr/lib/sna/samples/JPing.java** (AIX) または **/opt/ibm/sna/samples/JPing.java** (Linux) にあります) は、標準 APPC 関数 **aping** の単純な Java インプリメンテーションです。この関数は、リモート・ノードとの接続性を検査するのに使用されます。**aping** についての詳細は、「*Communications Server for AIX APPC Application Suite ユーザーズ・ガイド*」または「*Communications Server for AIX 管理コマンド・リファレンス*」を参照してください。

オプションとして、接続するパートナー LU を識別するシンボリック宛先名、ping の反復試行回数、および反復ごとに送信される情報サイズを指定することができます。

プログラムの操作に関する詳細については、プログラムのソース・ファイルのコメントを参照してください。

## サンプル・プログラムのコンパイル、リンク、および実行

**JPing** のソース・コードを調べた後で、プログラムを構築してテストする場合について説明します。

アプリケーションをコンパイルしてリンクする前に、Java クラスを保管しているディレクトリーを指定します。そのためには、環境変数 `CLASSPATH` を **/usr/lib/sna/java/cpic.jar:.** (AIX) または **/opt/ibm/sna/java/cpic.jar:.** (Linux) に設定し、エクスポートします。

プログラムをコンパイルしてリンクするには、次のステップを実行します。

1. ファイル **JPing.java** をディレクトリー **/usr/lib/sna/samples** (AIX) または **/opt/ibm/sna/samples** (Linux) からプライベート・ディレクトリーにコピーする。

2. プライベート・ディレクトリーから、Java コンパイラー **javac** を通常の方法で使用してアプリケーションをコンパイルし、リンクする。次のコマンドを使用します。

   **javac JPing.java**

**JPing.class** というファイルが生成されたはずです。

アプリケーションを実行する前に、ライブラリーが保管されているディレクトリーを指定します。これによりアプリケーションが実行時にライブラリーを見つけ出すことができます。

そのためには、該当する環境変数を次のように設定し、エクスポートします。

**export LD_LIBRARY_PATH=/usr/lib/sna**

また、 37 ページの『ローカル TP 名の指定』で説明しているように、APPCTPN 環境変数を設定しエクスポートして、アプリケーションのローカル TP 名を指定する必要があります。

プログラムを実行する場合は、CS/AIX 構成を更新して、パートナー LU を識別するシンボリック宛先名を組み込みます。このタスクは、通常、システム管理者が実行します。次のステップが必要です。

- 「Symbolic Destination Name (シンボリック宛先名)」に JPING を指定する。
- 「Partner TP Name Type (パートナー TP 名のタイプ)」に、Application Program を指定する。
- 「Partner TP Name (パートナー TP 名)」に APINGD を指定する。
- 「Partner LU (パートナー LU)」に、接続したいパートナー LU の完全修飾名を指定する。
- 「Mode Name (モード名)」に #INTER を指定する。

その他のパラメーターは、デフォルト値のままにしておきます。

Java インタープリター **java** を通常の方法で使用して、アプリケーションを実行します。次のコマンドを使用します。

**java JPing** [*sym_dest_name*] [
**-i** *num_iterations*] [**-s** *data_len*]

*sym_dest_name* は、プログラムに使用されるシンボリック宛先名を表します。 このオプションを指定しない場合は、デフォルトは JPING になります。

**-i** オプションは、ping 反復実行回数を示します。このオプションを指定しない場合、デフォルトは 2 になります。

**-s** オプションは、パートナー・プログラムに送信されるデータのバイト数を示します。このオプションを指定しない場合、デフォルトは 100 になります。

ping の反復の回数およびデータ長の使用方法についての詳細は、「*Communications Server for AIX APPC Application Suite ユーザーズ・ガイド*」または「*Communications Server for AIX 管理コマンド・リファレンス*」の **aping** の説明を参照してください。

# 付録 A. 戻りコードの値

この付録は、CPI-C インターフェースのすべての可能な戻りコードを番号順にリストしています。値は、ヘッダー・ファイル **cmc.h** (AIX または Linux 用) または **wincpic.h** (Windows 用) に定義されています。

この付録をリファレンスとして使用して、アプリケーションが受け取った戻りコードの意味を検査することができます。

```
CM_OK                                0
CM_ALLOCATE_FAILURE_NO_RETRY         1
CM_ALLOCATE_FAILURE_RETRY            2
CM_CONVERSATION_TYPE_MISMATCH        3
CM_PIP_NOT_SPECIFIED_CORRECTLY       5
CM_SECURITY_NOT_VALID                6
CM_SYNC_LVL_NOT_SUPPORTED_LU         7
CM_SYNC_LVL_NOT_SUPPORTED_PGM        8
CM_TPN_NOT_RECOGNIZED                9
CM_TP_NOT_AVAILABLE_NO_RETRY        10
CM_TP_NOT_AVAILABLE_RETRY           11
CM_DEALLOCATED_ABEND                17
CM_DEALLOCATED_NORMAL               18
CM_PARAMETER_ERROR                  19
CM_PRODUCT_SPECIFIC_ERROR           20
CM_PROGRAM_ERROR_NO_TRUNC           21
CM_PROGRAM_ERROR_PURGING            22
CM_PROGRAM_ERROR_TRUNC              23
CM_PROGRAM_PARAMETER_CHECK          24
CM_PROGRAM_STATE_CHECK              25
CM_RESOURCE_FAILURE_NO_RETRY        26
CM_RESOURCE_FAILURE_RETRY           27
CM_UNSUCCESSFUL                     28
CM_DEALLOCATED_ABEND_SVC            30
CM_DEALLOCATED_ABEND_TIMER          31
CM_SVC_ERROR_NO_TRUNC               32
CM_SVC_ERROR_PURGING                33
CM_SVC_ERROR_TRUNC                  34
CM_OPERATION_INCOMPLETE             35
CM_SYSTEM_EVENT                     36
CM_OPERATION_NOT_ACCEPTED           37
CM_CONVERSATION_ENDING              38
CM_SEND_RCV_MODE_NOT_SUPPORTED      39
CM_BUFFER_TOO_SMALL                 40
CM_EXP_DATA_NOT_SUPPORTED           41
CM_DEALLOC_CONFIRM_REJECT           42
CM_ALLOCATION_ERROR                 43
CM_RETRY_LIMIT_EXCEEDED             44
CM_NO_SECONDARY_INFORMATION         45
CM_SECURITY_NOT_SUPPORTED           46
CM_SECURITY_MUTUAL_FAILED           47
CM_CALL_NOT_SUPPORTED               48
CM_PARM_VALUE_NOT_SUPPORTED         49
CM_TAKE_BACKOUT                    100
CM_DEALLOCATED_ABEND_BO            130
CM_DEALLOCATED_ABEND_SVC_BO        131
CM_DEALLOCATED_ABEND_TIMER_BO      132
CM_RESOURCE_FAIL_NO_RETRY_BO       133
CM_RESOURCE_FAILURE_RETRY_BO       134
CM_DEALLOCATED_NORMAL_BO           135
CM_CONV_DEALLOC_AFTER_SYNCPT       136
CM_INCLUDE_PARTNER_REJECT_BO       137
```

# 付録 B. 共通な戻りコード

この付録では、複数の CPI-C コールに共通する戻りコードについて説明します。戻りコードはアルファベット順に示されています。パートナー・プログラムが CPI-C LU 6.2 プログラムではない場合に生成された戻りコードは、別に示してあります。

コール固有の戻りコードについては、53 ページの『第 3 章 CPI-C コール』の個々のコールについての項に説明があります。

## パートナー・プログラムに共通する戻りコード

次の戻りコードは、どのパートナー・プログラムにも共通するコードです (その他の戻りコード、つまりパートナー・プログラムが CPI-C プログラムではない場合の戻りコードについては、別に示してあります)。

**CM_ALLOCATION_FAILURE_NO_RETRY**
構成エラーまたはセッション・プロトコル・エラーなどの永続的な障害のために、会話の割り振りができません。エラーを判別するには、システム管理者がエラー・ログ・ファイルを検査する必要があります。エラーが訂正されるまで、割り振りを再試行しないでください。

**CM_ALLOCATION_FAILURE_RETRY**
リンク障害などの一時的な障害のために、会話の割り振りができませんでした。障害の理由がシステム・エラー・ログに記録されています。割り振りを再試行してください。

AIX、LINUX

**CM_CALL_NOT_SUPPORTED**
この戻りコードが使用されるのは、Java CPI-C アプリケーションにおいてのみです。

アプリケーションが、Java CPI-C クラスで定義されているが、サポートはされていない CPI-C 関数を使用しました。

**CM_CONVERSATION_TYPE_MISMATCH**
パートナー LU またはパートナー・プログラムが、割り振り要求に指定された会話タイプ (基本またはマップ式) をサポートしていません。

**CM_DEALLOCATED_ABEND**
次のいずれかの理由で、会話の割り振りが解除されました。

- パートナー・プログラムが、割り振り解除タイプを CM_DEALLOCATE_ABEND に設定した Deallocate コールを発行した。ローカル・プログラムがこのコールを発行したときにパートナー・プログラム側の会話状態が受信になっていると、ローカル・プログラムが送信しパートナー・プログラムがまだ受信していないデータは破棄されます。

- パートナー・プログラムは正常に終了したが、終了前に会話の割り振りを解除しなかった。
- ローカル・プログラムが Cancel_Conversation コールを発行し、それによって会話の未解決の非同期 CPI-C コールがすべて取り消された。

**CM_DEALLOCATED_NORMAL**
この戻りコードはエラーを示すものではありません。

パートナー・プログラムが、割り振り解除タイプを次のどちらかに設定した Deallocate コールを発行しました。

- CM_DEALLOCATE_FLUSH
- 会話の同期レベルが CM_NONE に指定された CM_DEALLOCATE_SYNC_LEVEL

**CM_OK** コールは正常に実行されました。

**CM_OPERATION_INCOMPLETE**
コールは正常に発行され、非ブロッキング・モードで動作中です。つまり、コールの処理はまだ完了していない場合でも、制御権はプログラムに戻されています。

プログラムは、この会話に関係のない処理 (他の会話での CPI-C コールの発行を含む) を続けることができます。この会話では、限定された範囲の CPI-C コール (Extract_* コールなど) を発行できます。この点が、IBM CPI-C 2.0 の仕様とは異なります。IBM CPI-C 2.0 では、プログラムがこの会話で発行できるコールは、Wait_For_Conversation または Cancel_Conversation だけでした。

AIX、LINUX

あとでアプリケーションは、Check_For_Completion を発行して未解決の非ブロッキング・コールが完了したかどうかを判別する、Wait_For_Conversation を発行してそのようなコールの完了を待つ、または Cancel_Conversation を発行して未解決のコールを取り消して会話を終了する、などの操作ができます。

WINDOWS

アプリケーションが、非同期のコールの完了通知を受け取るために Specify_Windows_Handle を使用していた場合、この通知を受け取るまでは、この会話上でコールを発行することはできません。そうでない場合、アプリケーションは、Wait_For_Conversation を発行して完了する非ブロッキング・コールを待つか、 Cancel_Conversation を発行して、未解決のコールを取り消して会話を終了させることができます。

**CM_OPERATION_NOT_ACCEPTED**
次のどちらかの状態が発生したため、コールを発行できません。

- この会話に未解決の非ブロッキング・コールがある。プログラムは、この会話に無関係な処理 (他の会話での CPI-C コールの発行を含む) を続けることができますが、この会話ではほとんどの CPI-C コールは発行できません。

AIX、LINUX

あとでアプリケーションは、Check_For_Completion を発行して未解決の非ブロッキング・コールが完了したかどうかを判別する、Wait_For_Conversation を発行してそのようなコールの完了を待つ、または Cancel_Conversation を発行して未解決のコールを取り消して会話を終了する、などの操作ができます。

- プログラムが DCE マルチスレッド環境で実行されていて、プログラムの別のスレッドからのこの会話についてのコールが未解決になっている。1 つの会話で処理できるコールは 1 つだけです。

WINDOWS

アプリケーションが、非同期のコールの完了通知を受け取るために Specify_Windows_Handle を使用していた場合、この通知を受け取るまでは、この会話上でコールを発行することはできません。そうでない場合、アプリケーションは、Wait_For_Conversation を発行して完了する非ブロッキング・コールを待つか、 Cancel_Conversation を発行して、未解決のコールを取り消して会話を終了させることができます。

**CM_PARAMETER_ERROR**

CPI-C で参照されているパラメーターが有効ではありません。有効でないパラメーターは、プログラムで指定されている場合もあり、プログラムの制御外の他のコンポーネント (構成ファイルなど) で指定されている場合もあります。たとえば、*mode_name* パラメーターは、Set_Mode_Name を使用してプログラムで指定される場合と、*sym_dest_name* パラメーターにより指定されたサイド情報エントリーから取得される場合があります。

**CM_PRODUCT_SPECIFIC_ERROR**

CPI-C が CM_PRODUCT_SPECIFIC_ERROR 戻りコードを生成した場合は、エラーの原因と必要な処置を示すエントリーがログ・ファイルの中に作成されています。これらのメッセージの解釈についての詳細は、「*Communications Server for AIX 管理ガイド*」を参照してください。

**CM_PROGRAM_ERROR_NO_TRUNC**

パートナー・プログラムが、送信状態または送信 - 保留状態のときに、エラー時の通信方向を CM_SEND_ERROR に設定した Send_Error コールを発行しました。データは切り捨てられていません。

**CM_PROGRAM_ERROR_PURGING**

次のどちらかの状態が発生しました。

- パートナー・プログラムが受信状態または確認状態で Send_Error コールを発行した。データは送信されましたが、まだ受信されないうちに除去されました。
- パートナー・プログラムが、送信 - 保留状態で、エラー時の通信方向を CM_RECEIVE_ERROR に設定した Send_Error コールを発行した。データは除去されていません。

**CM_PROGRAM_ERROR_TRUNC**
基本会話のパートナー・プログラムが、完全な論理レコードを送信し終わる前に、送信状態で Send_Error コールを発行しました。ローカル・プログラムは、Receive コールを介してその論理レコードの先頭の部分を受信している可能性があります。

**CM_PROGRAM_PARAMETER_CHECK**
プログラムが、このコールには有効でないパラメーターを提供しました。詳細については、53 ページの『第 3 章 CPI-C コール』の個々のコールを参照してください。

**CM_PROGRAM_STATE_CHECK**
発行されたコールは、現行の会話状態では許されないか、または会話特性の現行設定では不適切です。詳細については、53 ページの『第 3 章 CPI-C コール』の個々のコールを参照してください。

**CM_RESOURCE_FAILURE_NO_RETRY**
次のどちらかの状態が発生しました。

- 永続的な障害により、会話がまだ完了しないうちに終了した。エラーが訂正されるまで、再試行しないでください。
- パートナー・プログラムが、正常に終了する前に会話の割り振りを解除しなかった。

**CM_RESOURCE_FAILURE_RETRY**
モデムの障害などの一時的な障害により、会話がまだ完了しないうちに終了しました。会話を再試行してください。

**CM_SECURITY_NOT_VALID**
割り振り要求に指定されたユーザー ID またはパスワードを、パートナー LU が受け入れませんでした。

**CM_SYNC_LVL_NOT_SUPPORTED_PGM**
パートナー・プログラムが、割り振り要求に指定された同期レベルをサポートしていません。

**CM_SYNC_LVL_NOT_SUPPORTED_LU**
パートナー LU は、割り振り要求に指定された同期レベルをサポートしていません。

**CM_TP_NOT_AVAILABLE_NO_RETRY**
永続的な障害のため、パートナー LU は割り振り要求に指定されたプログラムを開始できません。エラーの理由がリモート・ノードのログに記録されていることがあります。エラーが訂正されるまで、割り振りを再試行しないでください。

**CM_TP_NOT_AVAILABLE_RETRY**
一時的な障害のため、パートナー LU は割り振り要求に指定されたプログ

ラムを開始できません。エラーの理由がリモート・ノードのログに記録されていることがあります。割り振りを再試行してください。

**CM_TPN_NOT_RECOGNIZED**
パートナー LU は、割り振り要求に指定されたプログラム名を認識できません。

**CM_UNSUCCESSFUL**
コールが正常に実行されませんでした。この戻りコードは次の場合に戻されます。

- プログラムが *return_control* パラメーターを CM_IMMEDIATE に設定した Allocate を発行したが、CS/AIX が会話用のセッションを即時に割り振ることができなかった。
- プログラムが *receive_type* パラメーターを CM_RECEIVE_IMMEDIATE に設定した Receive を発行したが、パートナー・プログラムから現在受信可能なデータもしくは制御情報がなかった。

AIX、LINUX

- プログラムが Check_For_Completion を発行したが、プログラムの会話のいずれにも、未解決の非ブロッキング関数が完了したものがなかった。

## CPI-C LU 6.2 以外のパートナー・プログラム

パートナー・プログラムが CPI-C LU 6.2 以外のプログラム (たとえば APPC TP) の場合は、次の戻りコードが戻されることがあります。次の説明の中に示されている verb は、LU 6.2 verb です。

**CM_DEALLOCATED_ABEND_SVC**
次のいずれかの理由で、会話の割り振りが解除されました。

- パートナー・プログラムが、*TYPE* を ABEND_SVC に設定した DEALLOCATE verb を発行した。
- パートナー・プログラムが、終了する前に会話の割り振りを解除しなかった。

ローカル・プログラムがこのコールを発行したときにパートナー・プログラム側の会話状態が受信になっていると、ローカル・プログラムが送信しパートナー・プログラムがまだ受信していないデータは破棄されます。

**CM_DEALLOCATED_ABEND_TIMER**
パートナー・プログラムが *TYPE* を ABEND_TIMER に設定した DEALLOCATE verb を発行したために、会話の割り振りが解除されました。ローカル・プログラムがこのコールを発行したときにパートナー・プログラム側の会話状態が受信になっていると、ローカル・プログラムが送信しパートナー・プログラムがまだ受信していないデータは破棄されます。

**CM_PIP_NOT_SPECIFIED_CORRECTLY**
割り振り要求が CPI-C LU 6.2 以外のプログラムによりリジェクトされまし

た。パートナー・プログラムには 1 つ以上の PIP データ変数が必要ですが、CPI-C は PIP データをサポートしていません。

**CM_SVC_ERROR_NO_TRUNC**
パートナー・プログラム (またはパートナー LU) が、基本会話の実行中に送信状態で、*TYPE* パラメーターを SVC に設定した SEND_ERROR verb を発行しました。データは切り捨てられていません。

**CM_SVC_ERROR_PURGING**
パートナー・プログラム (またはパートナー LU) が、送信状態で、*TYPE* パラメーターを SVC に設定した SEND_ERROR verb を発行しました。パートナー・プログラムに送信されたデータは除去された可能性があります。

**CM_SVC_ERROR_TRUNC**
パートナー・プログラム (またはパートナー LU) が、基本会話の実行中に受信状態または確認状態で、完全な論理レコードを送信し終わる前に、*TYPE* パラメーターを SVC に設定した SEND_ERROR verb を発行しました。ローカル・プログラムは論理レコードの先頭部分を受信している可能性があります。

# 付録 C. 会話状態の変化

198 ページの表 25 は、各 CPI-C 関数コールを発行できる会話状態、および各コールの完了時に起こる状態の変化を示しています。

場合によっては、状態の変化はコールからの戻りコードに応じて決まることがあります。OK 以外の戻りコードが戻されると、ほとんどの場合、状態は変化しません。戻りコードが示されていない場合、戻りコードが CM_OK のときは表に示すように状態が変化し、OK 以外の戻りコードのときは、(次の注に説明した場合を除いて) 状態は変化しません。戻りコードによってこれとは異なる状態の変化がある場合は、適用される値が「戻りコード」欄に示されています。

会話状態は各列のヘッダーに示されています。各コールについて、各ヘッダーの下に、その状態でコールを発行した場合の結果を示す次の情報があります。

**X** この状態ではコールは発行できません。

**T、I、II、S、SP、R、C、CS、CD、または PP**
コールの完了後の会話の状態が次のように示されます。リセット (R)、初期化 (I)、初期化 - 着呼 (II)、送信 (S)、送信 - 保留 (SP)、受信 (R)、確認 (C)、確認 - 送信 (CS)、確認 - 割り振り解除 (CD)、または保留 - 通知 (PP)

**(blank (空欄))**
この状態では、示されている戻りコードは戻されません。

**関数を参照**
53 ページの『第 3 章 CPI-C コール』にあるこの関数の説明を参照してください。会話状態の変化は、そのコールからの戻りパラメーターに応じて決まります。

**注:** 次に示す戻りコードを受信すると、会話は必ずリセット状態になります。

- CM_ALLOCATION_FAILURE_NO_RETRY、CM_ALLOCATION_FAILURE_RETRY
- CM_CONVERSATION_TYPE_MISMATCH
- CM_DEALLOCATED_NORMAL、CM_DEALLOCATED_ABEND
- CM_PIP_NOT_SPECIFIED_CORRECTLY
- CM_RESOURCE_FAILURE_RETRY、CM_RESOURCE_FAILURE_NO_RETRY
- CM_SECURITY_NOT_VALID、CM_SYNC_LVL_NOT_SUPPORTED_PGM、CM_SYNC_LVL_NOT_SUPPORTED_LU
- CM_TPN_NOT_RECOGNIZED、CM_TP_NOT_AVAILABLE_RETRY、CM_TP_NOT_AVAILABLE_NO_RETRY

AIX、LINUX

保留 - 通知状態は、AIX または Linux システムには適用されません。この状態に対する参照はすべて無視されます。

WINDOWS

初期化 - 着呼状態は、Windows システムには適用されません。この状態に対する参照はすべて無視されます。

Windows 固有の関数のコールは特定の会話と関連せず、会話状態に影響を及ぼすこともありません。 Windows 固有の関数は、この付録にリストされていません。

*表 25. 会話状態の変化*

| **CPI-C コールと** *primary_rc* **値** | **発行時の状態** | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | **リセット (T)** | **初期化 (I)** | **初期化着呼 (II)** | **送信 (S)** | **送信保留 (SP)** | **受信 (R)** | **確認 (C)** | **確認送信 (CS)** | **確認割り振り解除 (CD)** | **保留通知 (PP)** |
| Accept_Conversation | R | X | X | X | X | X | X | X | X | X |
| Accept_Incoming | X | X | R | X | X | X | X | X | X | X |
| Allocate | X | | X | X | X | X | X | X | X | X |
| CM_OK | | S | | | | | | | | |
| (割り振り障害) | | T | | | | | | | | |
| Cancel_Conversation | X | T | T | T | T | T | T | T | T | T |
| Check_For_Completion | T | I | II | S | SP | R | C | CS | CD | X |
| Confirm | X | X | X | | | X | X | X | X | X |
| CM_OK | | | | S | S | | | | | |
| (プログラム・エラー、SVC エラー) | | | | R | R | | | | | |
| Confirmed | X | X | X | X | X | X | R | S | T | X |
| Convert_Incoming, Convert_Outgoing | T | I | II | S | SP | R | C | CS | CD | X |
| Deallocate (異常終了) | X | | | | | | | | | |
| CM_OK | | T | T | T | T | T | T | T | T | T |
| (プログラム・エラー、SVC エラー) | | R | R | R | R | R | R | R | R | R |
| Deallocate (その他) | X | X | X | | | X | X | X | X | X |
| CM_OK | | | | T | T | | | | | |
| (プログラム・エラー、SVC エラー) | | | | R | R | | | | | |
| Delete_CPIC_Side_Information | T | I | II | S | SP | R | C | CS | CD | X |
| Extract_Conversation_Context | X | X | X | S | SP | R | C | CS | CD | X |

*表 25. 会話状態の変化 (続き)*

| **CPI-C コールと** *primary_rc* **値** | **発行時の状態** | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | **リセット (T)** | **初期化 (I)** | **初期化着呼 (II)** | **送信 (S)** | **送信保留 (SP)** | **受信 (R)** | **確認 (C)** | **確認送信 (CS)** | **確認割り振り解除 (CD)** | **保留通知 (PP)** |
| Extract_Conversation_Security_Type | X | I | II | S | SP | R | C | CS | CD | X |
| Extract_Conversation_状態 | X | I | II | S | SP | R | C | CS | CD | X |
| Extract_Conversation_Type | X | I | II | S | SP | R | C | CS | CD | X |
| Extract_CPIC_Side_Information | T | I | II | S | SP | R | C | CS | CD | X |
| Extract_Local_LU_Name | X | I | II | S | SP | R | C | CS | CD | X |
| Extract_Maximum_Buffer_Size | T | I | II | S | SP | R | C | CS | CD | X |
| Extract_Mode_Name | X | I | II | S | SP | R | C | CS | CD | X |
| Extract_Partner_LU_Name | X | I | X | S | SP | R | C | CS | CD | X |
| Extract_Security_User_ID | X | I | II | S | SP | R | C | CS | CD | X |
| Extract_Sync_Level | X | I | X | S | SP | R | C | CS | CD | X |
| Extract_TP_Name | X | I | X | S | SP | R | C | CS | CD | X |
| Flush | X | X | X | S | S | X | X | X | X | X |
| Initialize_Conversation | I | X | X | X | X | X | X | X | X | X |
| Initialize_For_Incoming | II | X | X | X | X | X | X | X | X | X |
| Prepare_To_Receive | X | X | X | | | X | X | X | X | X |
| CM_OK、プログラム・エラー、SVC エラー | | | | R | R | | | | | |
| 受信 (受信タイプ CM_RECEIVE_IMMEDIATE) | X | X | X | X | X | 関数を参照 | X | X | X | X |
| 受信 (受信タイプ CM_RECEIVE_AND_WAIT) | X | X | X | 関数を参照 | 関数を参照 | 関数を参照 | X | X | X | X |
| Release_Local_TP_Name | I | X | X | X | X | X | X | X | X | X |
| Request_To_Send | X | X | X | S | SP | R | C | CS | CD | PP |

*表 25. 会話状態の変化 (続き)*

| **CPI-C コールと** *primary_rc* **値** | **発行時の状態** | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | **リセット (T)** | **初期化 (I)** | **初期化着呼 (II)** | **送信 (S)** | **送信保留 (SP)** | **受信 (R)** | **確認 (C)** | **確認送信 (CS)** | **確認割り振り解除 (CD)** | **保留通知 (PP)** |
| Send_Data | X | X | X | 関数を参照 | 関数を参照 | X | X | X | X | X |
| Send_Error | X | X | X | | | | | | | |
| CM_OK | | | | S | S | S | S | S | S | S |
| プログラム・エラー、SVC エラー | | | | R | R | R | R | R | R | PP |
| Set_Conversation_Context | X | I | II | S | SP | R | C | CS | CD | X |
| Set_Conversation_Security_Password | X | I | X | X | X | X | X | X | X | X |
| Set_Conversation_Security_Type | X | I | X | X | X | X | X | X | X | X |
| Set_Conversation_Security_User_ID | X | I | X | X | X | X | X | X | X | X |
| Set_Conversation_Type | X | I | X | X | X | X | X | X | X | X |
| Set_CPIC_Side_Information | T | I | II | S | SP | R | C | CS | CD | X |
| Set_Deallocate_Type | X | I | II | S | SP | R | C | CS | CD | X |
| Set_Error_Direction | X | I | II | S | SP | R | C | CS | CD | X |
| Set_Fill | X | I | II | S | SP | R | C | CS | CD | X |
| Set_Local_LU_Name | X | I | X | X | X | X | X | X | X | X |
| Set_Log_Data | X | I | II | S | SP | R | C | CS | CD | X |
| Set_Mode_Name | X | I | X | X | X | X | X | X | X | X |
| Set_Partner_LU_Name | X | I | X | X | X | X | X | X | X | X |
| Set_Prepare_To_Receive_Type | X | I | II | S | SP | R | C | CS | CD | X |
| Set_Processing_Mode | X | I | II | S | SP | R | C | CS | CD | X |
| Set_Receive_Type | X | I | II | S | SP | R | C | CS | CD | X |
| Set_Return_Control | X | I | X | X | X | X | X | X | X | X |
| Set_Send_Type | X | I | II | S | SP | R | C | CS | CD | X |
| Set_Sync_Level | X | I | X | X | X | X | X | X | X | X |
| Set_TP_Name | X | I | X | X | X | X | X | X | X | X |
| Specify_Local_TP_Name | T | I | II | S | SP | R | C | CS | CD | X |

*表 25. 会話状態の変化 (続き)*

| **CPI-C コールと** *primary_rc* **値** | **発行時の状態** | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | **リセット (T)** | **初期化 (I)** | **初期化着呼 (II)** | **送信 (S)** | **送信保留 (SP)** | **受信 (R)** | **確認 (C)** | **確認送信 (CS)** | **確認割り振り解除 (CD)** | **保留通知 (PP)** |
| Test_Request_To_ Send_Received | X | X | X | S | SP | R | X | X | X | PP |
| Wait_For_Conversation | どの状態でも発行可能。新しい状態は、完了した未解決のコールと、そのコールからの戻りコードによって異なります。それぞれのコールについての説明を参照してください。 | | | | | | | | | |

# 付録 D. アクセシビリティ

アクセシビリティ機能は、運動障害または視覚障害など身体に障害を持つユーザーがソフトウェア・プロダクトを快適に使用できるようにサポートします。 z/OS™ のアクセシビリティの主要機能により、ユーザーは以下のことができるようになります。

- 画面読み上げ機能および画面拡大機能などの支援機能の使用
- キーボードのみを使用して、特定の機能または画面を使用したのと同等の機能を操作
- 色、コントラスト、フォント・サイズなど表示属性のカスタマイズ

## 支援機能の使用

画面読み上げ機能などの支援機能は、 z/OS のユーザー・インターフェースを使用して機能します。この支援機能を使用して z/OS インターフェースにアクセスする場合、その特定情報については支援機能の資料を参照してください。

## ユーザー・インターフェースのキーボード・ナビゲーション

ユーザーは、TSO/E または ISPF を使用して z/OS ユーザー・インターフェースにアクセスできます。TSO/E および ISPF インターフェースのアクセスについて詳しくは、「*z/OS TSO/E 入門*」(邦文番号 SA88-8632: 英文番号 SA22-7787)、「*z/OS TSO/E ユーザーズ・ガイド*」(邦文番号 SA88-8638: 英文番号 SA22-7794)、および「*対話式システム生産性向上機能 (ISPF) ユーザーズ・ガイド 第 1 巻 z/OS バージョン 1 リリース 2.0*」(邦文番号 SC88-8965: 英文番号 SC34-4822) を参照してください。上記の資料には、キーボード・ショートカットまたはファンクション・キー (PF キー) の使用方法を含む TSO/E および ISPF の使用方法が記載されています。それぞれの資料では、PF キーのデフォルトの設定値とそれらの機能の変更方法についても説明しています。

## z/OS の情報

z/OS の情報は、スクリーン・リーダー (読み上げソフトウェア) を使用して、BookServer/Library Server バージョンの z/OS ブックに次のインターネット・ライブラリーでアクセスできます。

`http://www.ibm.com/servers/eserver/zseries/zos/bkserv/`

# 付録 E. 特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものであり、本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒106-0032
東京都港区六本木 3-2-31
IBM World Trade Asia Corporation
Licensing

**以下の保証は、国または地域の法律に沿わない場合は、適用されません。** IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。 IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM Corporation, Site Counsel

P.O. Box 12195
3039 Cornwallis Road
Research Triangle Park, NC 27709-2195
U.S.A.

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、 IBM より提供されます。

この文書に含まれるいかなるパフォーマンス・データも、管理環境下で決定されたものです。そのため、他の操作環境で得られた結果は、異なる可能性があります。一部の測定が、開発レベルのシステムで行われた可能性がありますが、その測定値が、一般に利用可能なシステムのものと同じである保証はありません。さらに、一部の測定値が、推定値である可能性があります。実際の結果は、異なる可能性があります。お客様は、お客様の特定の環境に適したデータを確かめる必要があります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。 IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、名称や住所が類似する企業が実在しているとしても、それは偶然にすぎません。

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。お客様は、IBM のアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、 IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。

それぞれの複製物、サンプル・プログラムのいかなる部分、またはすべての派生した創作物にも、次のように、著作権表示を入れていただく必要があります。「© (お客様の会社名) (西暦年)」このコードの一部は、IBM Corp. のサンプル・プログラムから取られています。 © IBM Corp. 2000, 2005. All rights reserved.

## 商標

以下は、IBM Corporation の商標です。

ACF/VTAM
Advanced Peer-to-Peer Networking
AIX
AIXwindows
AnyNet
Application System/400
APPN
AS/400
CICS
DATABASE 2
DB2
Enterprise System/3090
Enterprise System/4381
Enterprise System/9000
ES/3090
ES/9000
eServer
IBM
IBMLink
IMS
MVS
MVS/ESA
Operating System/2
Operating System/400
OS/2
OS/400
PowerPC
PowerPC Architecture
pSeries
S/390
System/390
VSE/ESA
VTAM
WebSphere
zSeries

以下は、各々の会社の商標または登録商標です。

Java およびすべての Java ベースの商標は、米国およびその他の国の Sun Microsystems, Inc. の商標です。

UNIX は、The Open Group の米国およびその他の国における登録商標です。

Intel は、Intel Corporation の米国およびその他の国における商標です。

Linux は、Linus Torvalds の米国およびその他の国における商標です。

Microsoft、Windows、Windows NT、Windows 2003 および Windows ロゴは、Microsoft Corporation の米国およびその他の国における商標です。

他の会社名、製品名およびサービス名等は、それぞれ各社の商標または登録商標です。

# 参考文献

以下の IBM 資料では、本書で説明しているトピックについての情報を記載しています。資料は、次のトピック別に大きく分けてあります。

- CS/AIX バージョン 6.3
- IBM Communications Server for AIX, バージョン 4.2
- Redbooks™
- AnyNet/2 および SNA
- ブロック・マルチプレクサーおよび S/390 ESCON チャネル PCI アダプター
- AIX オペレーティング・システム
- システム・ネットワーク体系 (SNA)
- ホスト構成
- z/OS Communications Server
- マルチプロトコル・トランスポート・ネットワーキング
- 伝送制御プロトコル / インターネット・プロトコル (TCP/IP)
- X.25
- 拡張プログラム間通信機能 (APPC)
- プログラミング
- その他の IBM ネットワーキング・トピック

CS/AIX 関連の資料については、簡単な説明を付記してあります。その他の資料については、タイトル、資料番号を記し、一部の資料については本書で使用している略称タイトルを記しています。

## CS/AIX バージョン 6.3 の資料

CS/AIX 関連資料として次のものがあります。なお、これらの資料のソフトコピー版が CD-ROM で提供されています。 CD-ROM のソフトコピーへのアクセスの方法については、「*IBM Communications Server for AIX 入門*」を参照してください。これらのソフトコピー・ブックをシステムにインストールするには、9 〜 15 MB のハードウェア・ディスク・スペースが必要になります (このスペースは、どの各国語バージョンをインストールするかによって異なります)。

- *IBM Communications Server for AIX CS/AIX 移行ガイド* (邦文番号 SC88-6949: 英文番号 SC31-8585)

  この資料は、Communications Server for AIX バージョン 4.2 以前のバージョンから CS/AIX バージョン 6 への移行方法を説明しています。

- *IBM Communications Server for AIX 入門* (邦文番号 GC88-6947: 英文番号 GC31-8583)

  この資料は CS/AIX の概要を示すもので、サポートされているネットワークの特性、インストール、構成、および操作について説明しています。

- *IBM Communications Server for AIX 管理ガイド* (邦文番号 SC88-6950: 英文番号 SC31-8586)

  この資料では、SNA および CS/AIX の概要、および CS/AIX の構成と操作について説明しています。
- *IBM Communications Server for AIX 管理コマンド・リファレンス* (邦文番号 SD88-6675: 英文番号 SC31-8587)

  この資料では、SNA および CS/AIX のコマンドについて説明しています。
- *IBM Communications Server for AIX CPI-C プログラマーズ・ガイド* (邦文番号 SC88-6954: 英文番号 SC31-8591)

  この資料では、「C」または Java の熟練したプログラマーを対象として、CS/AIX CPI 通信 API を使用する SNA トランザクション・プログラムの作成に関する情報を提供しています。
- *IBM Communications Server for AIX APPC プログラマーズ・ガイド* (邦文番号 SC88-6953: 英文番号 SC31-8590)

  この資料では、拡張プログラム間通信機能 (APPC) を使用するアプリケーション・プログラムを作成するために必要な情報を記載しています。
- *IBM Communications Server for AIX LUA プログラマーズ・ガイド* (邦文番号 SC88-6955: 英文番号 SC31-8592)

  この資料では、従来型 LU アプリケーション・プログラミング・インターフェース (LUA) を使用してアプリケーション・プログラムを作成するために必要な情報を記載しています。
- *IBM Communications Server for AIX Common Service Verb プログラマーズ・ガイド* (邦文番号 SC88-6956: 英文番号 SC31-8593)

  この資料では、Common Service Verbs (CSV) アプリケーション・プログラミング・インターフェース (API) を使用してアプリケーション・プログラムを作成するために必要な情報を記載しています。
- *IBM Communications Server for AIX Management Services プログラマーズ・ガイド* (邦文番号 SC88-6957: 英文番号 SC31-8594)

  この資料では、Management Services (MS) API を使用してアプリケーション・プログラムを作成するために必要な情報を記載しています。
- *IBM Communications Server for AIX Node Operator Facility プログラマーズ・ガイド* (邦文番号 SC88-6958: 英文番号 SC31-8595)

  この資料では、Node Operator Facility (NOF) API を使用してアプリケーション・プログラムを作成するために必要な情報を記載しています。
- *IBM Communications Server for AIX 診断用ガイド* (邦文番号 SC88-6951: 英文番号 SC31-8588)

  この資料では、SNA ネットワークの問題解決について説明しています。
- *IBM Communications Server for AIX AnyNet® ガイド: APPC over TCP/IP* (邦文番号 GC88-6961: 英文番号 GC31-8598)

この資料では、AnyNet APPC over TCP/IP 機能のインストール、構成、および使用法について説明しています。

- *IBM Communications Server for AIX AnyNet ガイド: Sockets over SNA* (邦文番号 GC88-6960: 英文番号 GC31-8597)

  この資料では、AnyNet Sockets over SNA 機能のインストール、構成、および使用法について説明しています。

- *IBM Communications Server for AIX APPC Application Suite ユーザーズ・ガイド* (邦文番号 SC88-6959: 英文番号 SC31-8596)

  この資料では、CS/AIX で使用される APPC アプリケーションについて説明しています。

- *IBM Communications Server for AIX 用語集* (邦文番号 SC88-6952: 英文番号 GC31-8589)

  この資料は、 IBM Communications Server for AIX 関連資料で頻繁に使用される用語とその定義を包括的に収録しています。

## IBM Communications Server for AIX バージョン 4.2 関連資料

以下に挙げた資料は、Communications Server for AIX の以前のリリースのものです。バージョン 6 には適用されません。これらの資料は、バージョン 6 には含まれていませんが、継続してサポートされている情報を参照する場合に役に立ちます。

- *IBM Communications Server for AIX Transaction Program Reference* (英文番号 SC31-8212)

  この資料では、トランザクション・プログラミング API に関するバージョン 4.2 の情報を記載しています。バージョン 4.2 の API を使用するように作成されたアプリケーションは、バージョン 6 でも使用することができます。

## IBM Redbooks

IBM は、Redbooks として知られている資料を作成している International Technical Support Center を運営しています。製品の資料と同様、Redbooks は SNA テクノロジーの理論的側面と実用的側面の両方を扱っています。ただし、製品に同梱される資料に書かれている内容は、Redbooks には記載されていません。

次の資料では、CS/AIX に役に立つ情報を記載しています。

- *IBM Communications Server for AIX Version 6* (英文番号 SG24-5947)
- *IBM CS/AIX Understanding and Migrating to Version 5: Part 2 - Performance* (英文番号 SG24-2136)
- *Load Balancing for Communications Servers* (英文番号 SG24-5305)

ユーザーはワールド・ワイド・ウェブ (WWW) の http://www.redbooks.ibm.com から、 Redbooks 資料をダウンロードすることができます。

## ブロック・マルチプレクサーおよび S/390 ESCON チャネル PCI アダプター 関連資料

次の資料では、ブロック・マルチプレクサーと S/390 ESCON チャネル PCI アダプターについての情報を記載しています。

- AIX Version 4.1 Block Multiplexer Channel Adapter: User's Guide and Service Information (英文番号 SC31-8196)
- AIX Version 4.1 Enterprise Systems Connection Adapter: User's Guide and Service Information (英文番号 SC31-8197)
- AIX Version 4.3 S/390 ESCON Channel PCI: User's Guide and Service Information (英文番号 SC23-4232)
- IBM Communications Server for AIX チャネル接続ユーザーズ・ガイド (邦文番号 SC88-6905: 英文番号 SC31-8219)

## AnyNet/2 ソケットおよび SNA 関連資料

次の資料では、AnyNet/2 ソケットおよび SNA についての情報を記載しています。

- AnyNet/2 Version 2.0: Guide to Sockets over SNA (英文番号 GV40-0376)
- AnyNet/2 Version 2.0: Guide to SNA over TCP/IP (英文番号 GV40-0375)
- AnyNet/2: Guide to Sockets over SNA Gateway Version 1.1 (英文番号 GV40-0374)
- z/OS V1R2.0 Communications Server: AnyNet Sockets over SNA (英文番号 SC31-8831)
- z/OS V1R2.0 Communications Server: AnyNet SNA over TCP/IP (英文番号 SC31-8832 )

## AIX オペレーティング・システム関連資料

次の資料では、AIX オペレーティング・システムについての情報を記載しています。

- *AIX バージョン 5.3 システム・マネージメント・ガイド: オペレーティング・システムおよびデバイス* (邦文番号 SC88-6944: 英文番号 SC23-4910)
- *AIX バージョン 5.3 システム・マネージメント・コンセプト: オペレーティング・システムおよびデバイス* (邦文番号 SC88-6936: 英文番号 SC23-4908)
- *AIX バージョン 5.3 システム・マネージメント・ガイド: コミュニケーションおよびネットワーク* (邦文番号 SC88-6943: 英文番号 SC23-4909)
- *AIX バージョン 5.3 パフォーマンス・マネージメント・ガイド* (邦文番号 SC88-6934: 英文番号 SC23-4905)
- *AIX Version Performance Tools Guide and Reference* (英文番号 SC23-4906)
- *Performance Toolbox Version 2 and 3 Guide and Reference* (英文番号 SC23-2625)
- *AIXlink/X.25 Version 2.1 for AIX: Guide and Reference* (英文番号 SC23-2520)

## システム・ネットワーク体系 (SNA) 関連資料

次の資料では、SNA ネットワークについての情報を記載しています。

- *Systems Network Architecture: Format and Protocol Reference Manual - Architecture Logic for LU Type 6.2* (英文番号 SC30-3269)
- *Systems Network Architecture: Formats* (英文番号 GA27-3136)
- *Systems Network Architecture: Guide to SNA Publications* (英文番号 GC30-3438)
- *Systems Network Architecture: Network Product Formats* (英文番号 LY43-0081)
- *Systems Network Architecture: Technical Overview* (英文番号 GC30-3073)
- *Systems Network Architecture: APPN Architecture Reference* (英文番号 SC30-3422)
- *Systems Network Architecture: Sessions between Logical Units* (英文番号 GC20-1868)
- *Systems Network Architecture: LU 6.2 Reference - Peer Protocols* (英文番号 SC31-6808)
- *Systems Network Architecture: Transaction Programmer's Reference Manual for LU Type 6.2* (英文番号 GC30-3084)
- *IBM 3270 情報表示システム データストリーム プログラマー用解説書* (邦文番号 N:GA23-0059: 英文番号 GA23-0059)
- *Networking Blueprint Executive Overview* (英文番号 GC31-7057)
- *Systems Network Architecture: Management Services Reference* (英文番号 SC30-3346)

## ホスト構成関連資料

次の資料では、ホスト構成についての情報を記載しています。

- *ES/9000, ES/3090 IOCP User's Guide Volume A04* (英文番号 GC38-0097)
- *3174 Establishment Controller Installation Guide* (英文番号 GG24-3061)
- *3174 制御装置: 計画の手引き 構成サポート C リリース 5* (邦文番号 N:GA27-3918: 英文番号 GA27-3918)
- *OS/390 ハードウェア構成定義 User's Guide* (邦文番号 SC88-6630: 英文番号 SC28-1848)
- *IBM 9032 の導入計画 ESCON ディレクター* (邦文番号 N:GA23-0364: 英文番号 GA23-0364)

## z/OS Communications Server 関連資料

次の資料では、z/OS Communications Server についての情報を記載しています。

- *z/OS V1R7 Communications Server: SNA ネットワーク導入の手引き* (邦文番号 SC88-8928: 英文番号 SC31-8777)
- *z/OS V1R7 Communications Server: SNA Diagnostics* (英文番号 Vol 1: GC31-6850、Vol 2: GC31-6851)
- *z/OS V1R6 Communications Server SNA リソース定義解説書* (邦文番号 SC88-8929: 英文番号 SC31-8778)

## マルチプロトコル・トランスポート・ネットワーキング関連資料

次の資料では、マルチプロトコル・トランスポート・ネットワーキング・アーキテクチャーについての情報を記載しています。

- Multiprotocol Transport Networking: Formats (英文番号 GC31-7074)
- Multiprotocol Transport Networking Architecture: Technical Overview (英文番号 GC31-7073)

## TCP/IP 関連資料

次の資料では、伝送制御プロトコル / インターネット・プロトコル (TCP/IP) ネットワーク・プロトコルについての情報を記載しています。

- *z/OS V1R7 Communications Server: IP 構成ガイド* (邦文番号 SC88-8926: 英文番号 SC31-8775)
- *z/OS V1R7 Communications Server: IP 構成解説書* (邦文番号 SC88-8927: 英文番号 SC31-8776)
- *z/VM V5R1 TCP/IP 計画およびカスタマイズ* (邦文番号 SD88-6453: 英文番号 SC24-6125)

## X.25 関連資料

次の資料では、X.25 ネットワーク・プロトコルについての情報を記載しています。

- *AIXLink/X.25 for AIX: Guide and Reference* (英文番号 SC23-2520)
- *RS/6000® AIXLink/X.25 Cookbook* (英文番号 SG24-4475)
- *Communications Server for OS/2 Version 4 X.25 Programming* (英文番号 SC31-8150)

## APPC 関連資料

次の資料では、拡張プログラム間通信機能 (APPC) についての情報を記載しています。

- *APPC Application Suite V1 User's Guide* (英文番号 SC31-6532)
- *APPC Application Suite V1 Administration* (英文番号 SC31-6533)
- *APPC Application Suite V1 Programming* (英文番号 SC31-6534)
- *APPC Application Suite V1 Online Product Library* (英文番号 SK2T-2680)
- *APPC Application Suite Licensed Program Specifications* (英文番号 GC31-6535)
- *z/OS V1R2.0 Communications Server: APPC Application Suite User's Guide* (英文番号 SC31-8809)

## プログラミング関連資料

次の資料では、プログラミングについての情報を記載しています。

- *共通プログラミング・インターフェース コミュニケーション・インターフェース CPI-C 解説書* (邦文番号 SC88-7217: 英文番号 SC26-4399)

- *Communications Server for OS/2 Warp 日本語版 32 ビットアプリケーション・プログラミングの手引き* (邦文番号 SC88-5585: 英文番号 SC31-8152)

## その他の IBM ネットワーキング関連資料

次の資料では、CS/AIX に関連するその他のトピックについての情報を記載しています。

- *同期データ・リンク制御 (SDLC) 解説書* (邦文番号 N:GA27-3093: 英文番号 GA27-3093)
- *Local Area Network Concepts and Products: LAN Architecture* (英文番号 SG24-4753)
- *Local Area Network Concepts and Products: LAN Adapters, Hubs and ATM* (英文番号 SG24-4754)
- *Local Area Network Concepts and Products: Routers and Gateways* (英文番号 SG24-4755)
- *Local Area Network Concepts and Products: LAN Operating Systems and Management* (英文番号 SG24-4756)
- *ネットワーク制御プログラムおよびシステム・サポート・プログラム 資源定義の手引き* (邦文番号 N:SC30-3349: 英文番号 SC30-3349)

# 索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

## [ア行]


アクセシビリティ 203
アプリケーション TP 4
アプリケーション・プログラム・インターフェース 1
エラー時の通信方向 142
エラーの報告 121
エラー戻りコード 191
エラー・メッセージ 193
エラー・ログ・データ 13, 76, 125, 148
オペレーター開始待機プログラム 41


## [カ行]


会話
  開始 5
  基本 5
  コンテンション 3
  終了 6, 29
  状態 7
  セキュリティー 14
  同期レベル 7
  マップ式 4
  割り振り 4
  割り振り解除 4, 7, 29, 74
  TP 側から見た会話 9
会話 ID 55, 98, 100
会話、複数 13
会話状態
  取得 82
  初期 10
  説明 7
  変化 9, 197
  変更 9
会話セキュリティー
  概要 14
  タイプ 130, 132
  パスワード 127
  ユーザー ID 132
会話タイプ
  基本 4
  設定 134
  マップ式 4
  Allocate コールによる 60
  Extract_Conversation_Type コールの使用時の 83
会話特性
  初期値 22, 98, 100
  シンボリック宛先名に関連した 136
  Accept_Conversation による設定 55
  Accept_Incoming による設定 57
  Allocate での考慮事項 60
会話の状態 82
会話の即時割り振り 62
確認 - 送信状態 8
確認 - 割り振り解除状態 8
確認状態 8
確認処理 6
確認要求
  応答 7, 68
  および Confirm コール 65
  受信 7, 70
  送信 7
キーボード 203
記号定数 54
基本会話
  タイプ 5
  特性 12
構成情報 36, 78, 136, 140
コンテキスト 78, 126
コンテンション勝者とコンテンション敗者 3
コンパイルとリンク 49


## [サ行]


サービス TP 5
最大バッファー・サイズ 89
サイド情報 77, 84, 136
サンプル・プログラム
  概要 181
  疑似コード 181
サンプル・プログラム、Java CPI-C の 185
自動開始待機プログラム 40
自動開始非待機プログラム 40
充てん会話特性 144
受信、データの
  コールの使用可能化 24
  データを待つ 157
  Receive コールによる 6, 104
受信準備タイプ 153
受信状態
  定義 8
  変更 10, 101
受信タイプ 157
ショートカット・キー 203
状態の変化 197
初期化 - 着呼状態 8

初期化状態 8
処理モード 155
身体障害 203
シンボリック宛先名 34, 77, 136
セキュリティー・タイプ 80
セッション、LU-LU 3
セッションの割り振りまで待機 159
送信 - 保留状態 8
送信、データの 97
  使用するコール 23
  Request_To_Send コールの使用 115
  Send_Data コールの使用 5, 117
送信状態
  定義 8
  変更 11
送信タイプ 161
送信要求通知
  有無のテスト 170
  Request_To_Send コールでの 117

## [タ行]

待機、セッションが割り振られるまで 159
データ
  受信 6
  送信 5
データの受信 104
データ・タイプ 54
データ・バッファー、サイズ 89
データ・レコード 5, 144
同期、パートナー・プログラムとの 68
同期レベル
  および Extract_Sync_Level 94
  設定 7, 163
トランザクション・プログラム (TP)
  パートナー TP 4
  呼び出し側 TP 4
  呼び出し対象 TP 4
  リモート TP 4
  ローカル TP 4

## [ハ行]

パートナー LU 4
パートナー LU 名 91, 152
パートナー TP 4
パートナー TP 名 164
パートナー・プログラム名 164
パスワード、会話セキュリティー 127
バッファー・サイズ 89
非ブロッキング操作 16
非ブロッキング・モード 16
複数セッション 3
フラッシュ、ローカル LU の送信バッファーの 5, 97
ブロッキング・コール、Windows 48
ブロッキング・モード 16
分散トランザクション処理 1
並列セッション 3
変換 (EBCDIC-ASCII) 113, 120
変換、ASCII と EBCDIC 間の 120
変換、ASCII と EBCDIC 間のデータの 25
報告、エラーの 121
保留 - 通知状態、Windows 8

## [マ行]

マップ式会話 4, 134
マルチプロセス 42
モード 3
モード名 90, 150
戻りコード 189
戻りコード、共通 191
戻り制御 159

## [ヤ行]

ユーザー ID、会話セキュリティー 87, 92, 132
呼び出し側 TP 4
呼び出し側プログラム、開始 40
呼び出し対象 TP 4
呼び出し対象プログラム
  オペレーター開始待機 41
  開始 40
  自動開始待機 40
  自動開始非待機 40

## [ラ行]

リセット状態 8
リモート LU 4
リモート TP 4
ローカル LU 4
ローカル TP 4
ログ・データ 76, 125, 148
論理装置 (LU)
  パートナー LU 4
  リモート LU 4
  ローカル LU 4
  LU 6.2 3
論理レコード 12, 144

## [ワ行]

割り振り、会話の
  エラー 62
  割り振りの確認 62
  Allocate コールの使用 60
割り振り解除、会話の 74
割り振り解除、パートナー・プログラムからの通知の受信 74
割り振り解除タイプ 76, 140

# A


Accept_Conversation 55
Accept_Incoming 57
AIX アプリケーション
  コンパイルとリンク 42
AIX アプリケーションのコンパイル 42
AIX アプリケーションのリンク 42
Allocate コール 60
ASCII-EBCDIC データ変換 25


# C


Cancel_Conversation 62
Check_For_Completion 64
CM_ALLOCATION_FAILURE_NO_RETRY 191
CM_ALLOCATION_FAILURE_RETRY 191
CM_CALL_NOT_SUPPORTED 191
CM_CONVERSATION_TYPE_MISMATCH 191
CM_DEALLOCATED_ABEND 191
CM_DEALLOCATED_ABEND_SVC 195
CM_DEALLOCATED_ABEND_TIMER 195
CM_DEALLOCATED_NORMAL 192
CM_OK 192
CM_OPERATION_INCOMPLETE 192
CM_OPERATION_NOT_ACCEPTED 192
CM_PARAMETER_ERROR 193
CM_PIP_NOT_SPECIFIED_CORRECTLY 196
CM_PRODUCT_SPECIFIC_ERROR 193
CM_PROGRAM_ERROR_NO_TRUNC 193
CM_PROGRAM_ERROR_PURGING 193
CM_PROGRAM_ERROR_TRUNC 194
CM_PROGRAM_PARAMETER_CHECK 194
CM_PROGRAM_STATE_CHECK 194
CM_RESOURCE_FAILURE_NO_RETRY 194
CM_RESOURCE_FAILURE_RETRY 194
CM_SECURITY_NOT_VALID 194
CM_SVC_ERROR_NO_TRUNC 196
CM_SVC_ERROR_PURGING 196
CM_SVC_ERROR_TRUNC 196
CM_SYNC_LVL_NOT_SUPPORTED_LU 194
CM_SYNC_LVL_NOT_SUPPORTED_PGM 194
CM_TPN_NOT_RECOGNIZED 195
CM_TP_NOT_AVAILABLE_NO_RETRY 194
CM_TP_NOT_AVAILABLE_RETRY 195
CM_UNSUCCESSFUL 195
Confirm コール 65
confirmed 68
Convert_Incoming コール 71
Convert_Outgoing コール 72
CPI-C コール
  概要 4
  機能別の要約 21
CPI-C 用の関数コール、Windows 固有 47


# D


Deallocate コール 74
Delete_CPIC_Side_Information 77


# E


EBCDIC と ASCII の間の変換 113
EBCDIC-ASCII データ変換 25
Extract_Conversation_Context 78
Extract_Conversation_Security_Type 80
Extract_Conversation_Security_User_ID 81, 82
Extract_Conversation_State 82
Extract_Conversation_Type 83
Extract_CPIC_Side_Information 84
Extract_Local_LU_Name 87
Extract_Maximum_Buffer_Size 89
Extract_Mode_Name 90
Extract_Partner_LU_Name 91
Extract_Security_User_ID 92
Extract_Sync_Level 94
Extract_TP_Name 95


# F


Flush 97


# I


Initialize_Conversation 98
Initialize_For_Incoming 100


# J


Java CPI-C
  アプリケーションのコンパイルとリンク 46
  アプリケーションの実行 46
  クラス 43
  使用例 44
  定数 43
  パラメーター型 44
  プログラムの作成 43


# L


Linux アプリケーション
  コンパイルとリンク 42
Linux アプリケーションのコンパイル 42
Linux アプリケーションのリンク 42
LU 名、パートナー 91
LU-LU セッション 3

# P

Prepare_To_Receive 101

# R

Receive 104
Release_Local_TP_Name 114
Request_To_Send 115

# S

Send_Data 117
Send_Error 121
Set_Conversation_Context 126
Set_Conversation_Security_Password 127, 130
Set_Conversation_Security_Type 130, 132
Set_Conversation_Security_User_ID 132, 134
Set_Conversation_Type 134
Set_CPIC_Side_Information 136
Set_Deallocate_Type 140
Set_Error_Direction 142
Set_Fill 144
Set_Local_LU_Name 146
Set_Log_Data 148
Set_Mode_Name 150
Set_Partner_LU_Name 152
Set_Prepare_To_Receive_Type 153
Set_Processing_Mode 155
Set_Receive_Type 157
Set_Return_Control 159
Set_Send_Type 161
Set_Sync_Level 163
Set_TP_Name 164
Specify_Local_TP_Name 166
Specify_Windows_Handle 168

# T

Test_Request_to_Send_Received 170
TP 間の通信 2
TP 通信 2
TP 名 95

# W

Wait_For_Conversation 171
WinCPICCleanup コール 174
WinCPICIsBlocking コール 174
WinCPICStartup コール 176
Windows に関する考慮事項 46

プログラム番号: 5765-E51

Printed in Japan

SC88-6954-02

日本アイ・ビー・エム株式会社
〒106-8711　東京都港区六本木3-2-12