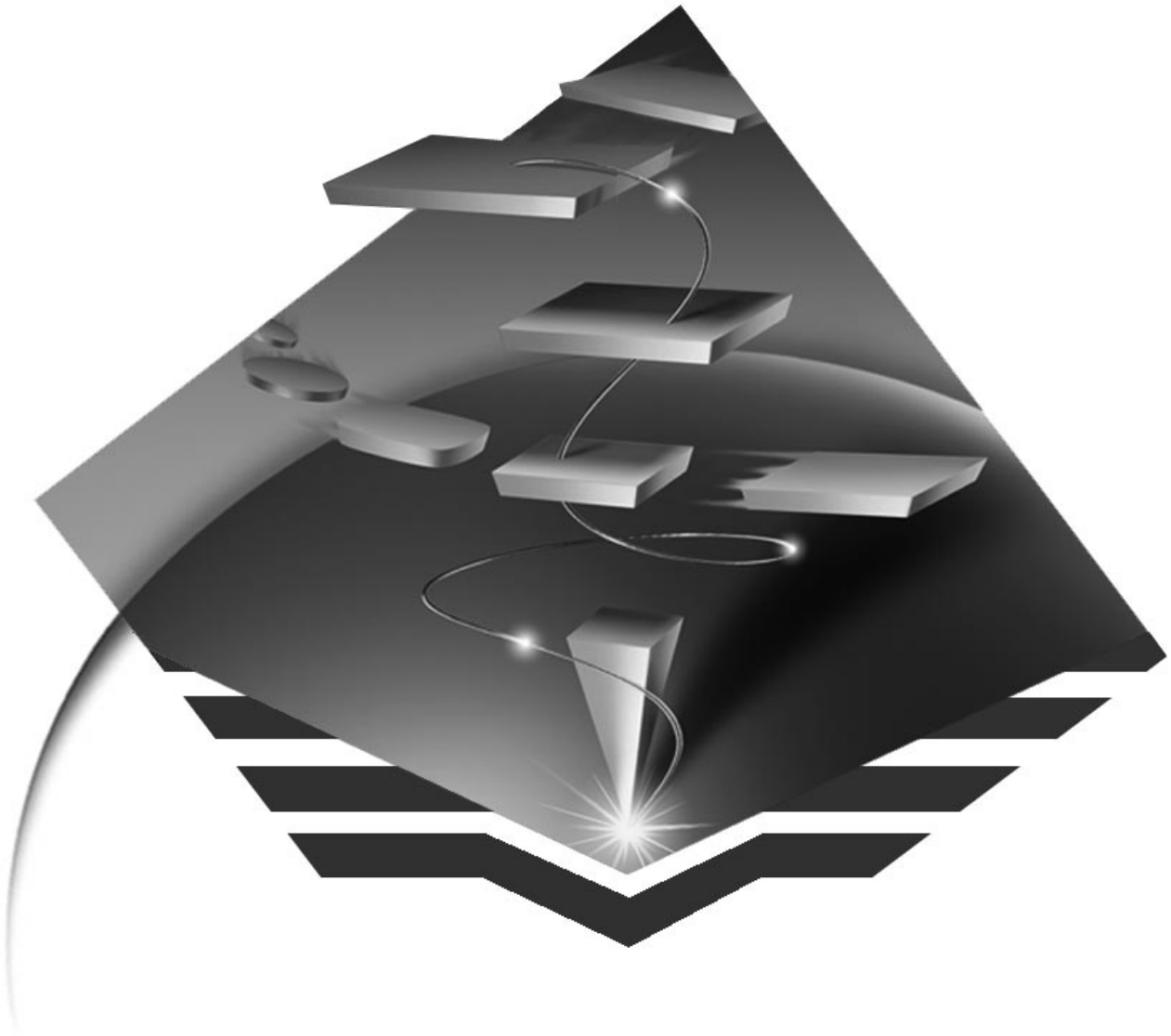
Communications Server
for Windows NT**

クライアント/サーバー コミュニケーション・プログラミング

バージョン 5.0



英文原典：SC31-8425-00
SC88-7727-00

Communications Server
for Windows NT**

クライアント/サーバー コミュニケーション・プログラミング

バージョン 5.0

英文原典：SC31-8425-00
SC88-7727-00

ご注意

本書の情報およびそれによってサポートされる製品を使用する前に、xvページの『特記事項』に記載する一般情報をお読みください。

本書は、Communications Server バージョン 5.0 に適用されます。

原　典：	SC31-8425-00 Communications Server for Windows** NT Client/Server Communinations Programming Version 5.0
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 1997.4

目次

図

表

特記事項

本書において、日本では発表されていない IBM 製品、プログラム、およびサービスについて言及または説明する場合があります。しかし、このことは、 IBM がこのような製品、プログラム、およびサービスを、日本で発表する意図があることを必ずしも示すものではありません。本書で IBM 製品、プログラム、またはサービスに言及している部分があっても、このことは 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。これらのプログラムまたは製品に代えて、IBM の知的所有権を侵害することのない機能的に同等のプログラムまたは製品を使用することができます。ただし、 IBM によって明示的に指定されたものを除き、これらの製品、プログラム、またはサービスに関連する動作の評価および検査は、お客様の責任で行っていただきます。

IBMは、本書で説明する主題に関する特許権(特許出願を含む）、商標権、または著作権を所有している場合があります。本書は、これらの特許権、商標権、および著作権について、本書で明示されている場合を除き、実施権、使用権等を許諾することを意味するものではありません。実施権、使用権等の許諾については、下記の宛先に、書面による照会状を送付してください。

〒106 東京都港区六本木 3 丁目 2-31

AP 事業所

IBM World Trade Asia Corporation

Intellectual Property Law & Licensing

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。

Site Counsel

IBM Corporation

P.O. Box 12195

3039 Cornwallis Road

Research Triangle Park, NC 27709-2195

USA

本プログラムに関する上記の情報は、適切な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。

本書に記載するライセンス・プログラムおよびそのライセンス資料のすべては、「IBM プログラム使用契約書」の契約条項にもとづいて弊社から提供されるものです。

本書は、実働面での使用を意図したものではなく、いかなる種類の保証も含まずそのままの形でお届けするものです。したがって、市販性および特定目的への適合性を含めたいかなる保証も認められません。

商標

以下の用語は、米国 IBM 社の商標です。

ACF/VTAM

Advanced Peer-to-Peer Networking

AFP

AIX

AIXwindows

Application System/400

APPN

AS/400

CallPath

CallPath/2

CallPath SwitchServer/2

CICS

CUA

C/2

IBM

IMS

NetView

Operating System/2

OS/2

RACF

SAA

SP

System/370

S/370

VM/ESA

VTAM

PC Direct は Ziff Communications Company の商標で、ライセンスにもとづき IBM Corporation が使用しています。

UNIX は、X/Open カンパニーリミテッドがライセンスしている米国ならびに他の国における登録商標です。

C-bus は Corollary, Inc. の商標です。

Microsoft、Windows、および Windows 95 ロゴ は、Microsoft Corporation の商標および登録商標です。

Java および HotJava は、Sun Microsystems, Inc. の商標です。

その他の会社、製品、およびサービス名（二重アスタリスク (**) で示されます）は、他社の商標またはサービス・マークです。

本書について

本書は 4 部に分かれています。

- 第1部 APPC APIでは、Communications Server拡張プログラム間通信機能 (APPC) インターフェースを使用するプログラムの開発方法について説明します。APPC は、論理装置 (LU) タイプ 6.2 用としてシステム・ネットワーク体系 (SNA) を実現したものを指します。本書では、特に断らないかぎり、APPC は Communications Server が提供する APPC を意味します。

  APPC は分散トランザクション処理能力を備えており、複数のプログラムが協調して処理機能を実行します。この能力にはプログラム間の通信も含まれているので、プロセッサー・サイクル、データベース、作業待ち行列、物理インターフェース（キーボードやディスプレイ）などの資源を、プログラム間で共用できます。
- 第2部 LUA APIでは、IBM 従来型 LU アプリケーション (LUA) インターフェースを使用して SNA LU タイプ 0、1、2、および 3 にアクセスするプログラムの開発方法について説明します（本書では、LUA は要求単位インターフェース (RUI) も指しています)。
- 第3部 共通サービス APIには、共通サービス API の要素である verb を収めてあります。
- 第4部 EHNAPPC APIでは、ルーター Windows API、機能、構造および戻りコードを説明します。

本書は、Communications Server が提供するクライアント・アプリケーションとサーバー・アプリケーションのユーザーを対象にしています。クライアント API は、Windows NT、Windows 95、 Windows 3.1 および OS/2 のプラットフォームで提供されます。また、*Windows* は Windows NT を指します。

本書の対象読者

本書は、APPC アプリケーションまたは LUA アプリケーションを作成するプログラマーおよび開発者を対象としています。

本書では、読者が *SNA Transaction Programmer's Reference Manual for LU Type 6.2*の内容を理解していることが前提になっています。

本書の使用方法

- 第1章 APPC の紹介では、拡張プログラム間通信機能 (APPC) について説明します。
- 第2章 APPC の基本概念では、APPC トランザクション・プログラムについて説明します。
- 第3章 接続マネージャーの使用では、接続マネージャーの使用方法について説明します。
- 第4章 トランザクション・プログラムの作成では、トランザクション・プログラムの作成方法について説明します。

- 第5章 APPC トランザクション・プログラムの実行では、APPC 拡張機能について説明します。
- 第6章 CPI-C プログラムの実行では、CPI-C プログラムについて説明します。
- 第7章 APPC 用のエントリー・ポイントでは、APPC API 用のプロシージャー・エントリー・ポイントについて説明します。
- 第8章 APPC verbでは、個々の APPC verb の構文について説明します。それぞれの verb 制御ブロックの構造と、個々のパラメーターに関する説明、および戻りコードの一覧を示してあります。
- 第9章 IBM 従来型 LU アプリケーションの基本概念では、本書での LUA プログラミングの基本概念について説明します。
- 第10章 LUA verb の機能では、各 LUA verb の機能を説明します。
- 第11章 LUA プログラムの実行では、LUA アプリケーション・プログラムの作成に関するいくつかの問題を説明します。
- 第12章 LUA エントリー・ポイントでは、LUA のプロシージャー・エントリー・ポイントを説明します。
- 第13章 LUA verbでは、各 LUA verb について詳しく説明します。
- 第14章 エントリー・ポイントでは、プロシージャー・エントリー・ポイントを説明します。
- 第15章 共通サービス verb (CSV)では、共通サービス verb を説明します。
- 第16章 EHNAPPC アプリケーション・プログラム・インターフェースでは、EHNAPPC API を説明します。
- 第17章 データ形式変更 Windows アプリケーション・プログラム・インターフェースでは、データ形式変更 Windows API を説明します。
- 付録 A. APPC 共通戻りコードでは、共通戻りコードについて説明します。
- 付録 B. APPC 会話状態の変化では、個々の APPC verb を発行できる会話の状態と、verb の完了時に起きる状態変化について説明します。

アイコン

本書では、異なるタイプの情報が探しやすいように、文中にアイコン（絵）を使用しています。

このアイコンは、その情報が基本 APPC verb に適用されることを示します。基本 verb については、第8章 APPC verbを参照してください。

このアイコンは、その情報がマップ式 APPC verb に適用されることを示します。 マップ式 verb については、第8章 APPC verbを参照してください。

また、特殊な情報を伝える必要がある場合、本書では、電話が鳴っている絵のアイコンで示しています。

本書で使用している表記規則

以下に示す表記規則は、Communications Server ライブラリーで共通に使用されているものです。 したがって、本書では使用されていないものも含まれています。

テキストの表記規則

Bold（ボールド）	ボールド書体は、プログラム内またはコマンド・プロンプトで使用できる verb、関数、およびパラメーターを示します。 これらの値は大文字小文字の区別があり、テキストに表記されているとおりに入力する必要があります。
Italics（イタリック）	イタリック書体は以下の場合に使用されます。 • ユーザーが値を指定する変数。 • リスト、チェック・ボックス、入力フィールド、押しボタン、メニュー選択項目などのウィンドウ制御項目の名前。 これらの名前は、ウィンドウに表示される通りにテキストに表記されています。 • 資料の表題。 • 文字はそのままの文字として、または語はそのままの語として使用されます。**例:** *a* と表記されている場合、これは *an* を意味するものではありません。
Bold italics（ボールド・イタリック）	ボールド・イタリック書体は語を強調するために使用されます。
UPPERCASE（大文字）	大文字は、プログラム内またはコマンド・プロンプトで使用できる定数、ファイル名、キーワード、およびオプションを示します。これらの値は、大文字または小文字のどちらで入力してもかまいません。
かぎ括弧（「」）	かぎ括弧はウィンドウに表示されるメッセージを示します。 たとえば、エミュレーター・セッションの操作員情報域 (OIA) に現れるメッセージなどです。
`Example type`（モノスペース）	モノスペース書体は、ユーザーがコマンド・プロンプトまたはウィンドウに入力する情報の例を示します。

数字の表記規則

2 進数	BX'xxxx xxxx' または BX'x' として表記されます。ただし、テキスト中では、「2 進数 xxxx xxxx の値は〜」のように表記される場合があります。
ビット位置	右端の位置の 0 (最小有効ビット) で始まります。
10 進数	4 桁を超える 10 進数はメートル法で表記します。3 桁ごとの区切りには、コンマではなくスペースを使用しています。たとえば、一万六千百四十七は、16 147 と表記します。
16 進数	テキスト中では、16 進数 xxxx または X'xxxx' の形で表します（たとえば、「隣接ノードのアドレスは 16 進数 5D で、X'5d'と指定します」のようになります)。

2 バイト文字セットのサポート

Communications Server は、1 文字が 2 バイトで表される 2 バイト文字セット (DBCS) をサポートします。 日本語、中国語、韓国語など、256 個のコード・ポイントで表せる記号より多くの記号を含む言語では、2 バイト文字セットが必要です。各文字にそれぞれ 2 バイトが必要なので、DBCS文字を入力、表示、印刷するには、DBCS をサポートするハードウェアおよびプログラムが必要です。

DBCS に適用される特殊情報がある場合は、該当の項の中にその旨を示してあります。

本書では、ASCII は、PC 単一バイト・コードを指します。 ASCII は、日本ではJISCIIとみなしてください。

関連情報

関連資料

SNA、APPN、または LU 6.2 アーキテクチャーについては、次の IBM 資料を参照してください。

- *IBM Systems Network Architecture: LU6.2 Reference: Peer Protocols*, SC31-6808
- *IBM Systems Network Architecture: APPN Architecture Reference*, SC30-3422
- *IBM Systems Network Architecture: Management Services*, SC30-3346
- *IBM Systems Network Architecture: Formats*, GA27-3136
- *IBM APPN Architecture and Product Implementations Tutorial*, GG24-3669
- *IBM AS/400 Advanced Peer-to-Peer Networking*, GG24-3287
- *コミュニケーション・マネージャー/2 システム管理プログラミング解説書*、SC88-5528
- *IBM コミュニケーション・マネージャー/2 拡張プログラム間通信 (APPC) プログラミングの手引きと解説書*、SC88-5521
- *S/370 解説書*、N:GA22-7000
- *SNA 解説書*、N:GC30-3073
- *IBM Systems Network Architecture: VTAM Programming for LU Type 6.2*, SC30-3400
- *SNA 概念と諸製品*、N:GC30-3072
- *IBM Systems Network Architecture Format and Protocol Reference Manual: Architecture Logic for LU Type 6.2*, SC30-3269
- *IBM Systems Network Architecture: Introduction to APPC*, GG24-1584
- *IBM Systems Network Architecture: Transaction Programmer's Reference Manual for LU Type 6.2*, GC30-3084
- *IBM Systems Network Architecture: Introduction to Sessions between Logical Units*, GC20-1869

- *IBM Systems Network Architecture Format and Protocol Reference Manual: Architectural Logic*, SC30-3112

第1部 APPC API

第1章 APPC の紹介

Communications Server は、ワークステーションに対する拡張分散ネットワーク機能 (APPN) エンド・ノード・サポートを提供しており、この機能を利用してワークステーションは、ネットワーク内の他のシステムとの間で融通性に富んだ通信を行うことができます。

Communications Serverは、*トランザクション・プログラム* (TP) と呼ばれる分散処理プログラム間の通信をサポートする拡張プログラム間通信機能 (APPC)を提供します。APPN は、この機能をネットワーキング環境にまで拡張します。トランザクション・プログラムは、APPC を備えている、ネットワーク内のどのノードにあってもかまいません。

Communications Server は、ローカル・エリア・ネットワーク (LAN) 環境での APPC スループットを向上させ、各種のプロトコル、たとえば、IBM トークンリング・ネットワーク、同期データ・リンク制御 (SDLC)、Twinaxial、イーサネットなどによって APPC をサポートします。

5ページの図 1は、Communications Serverによる APPC の機能構造を示しています。

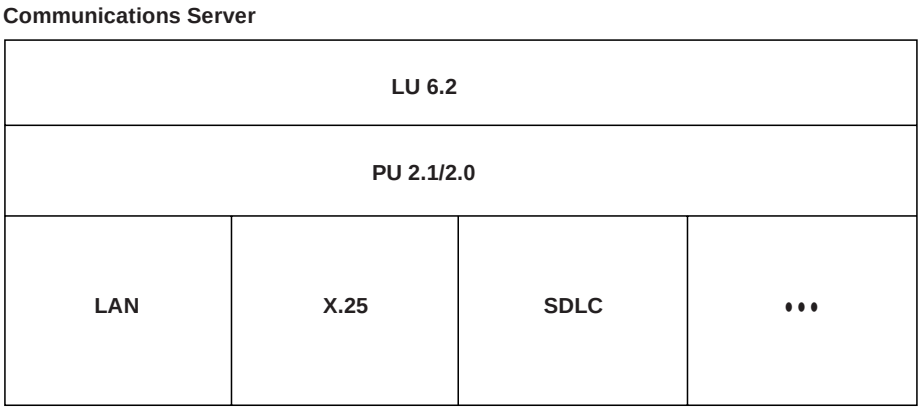


図 1. Communications Serverによる APPC の機能構造

SNA 通信サポート

Communications Serverは、システム・ネットワーク体系 (SNA) タイプ 2.1 ノードをサポートしています（SNA LU 6.2 以外の論理装置 [LU] に対するSNA タイプ 2.0 および SNA タイプ 2.1 サポートも含まれています)。 このサポートを利用して、他の多くの IBM SNA 製品と通信するためのプログラムを作成することができます。

プログラムを書く際に、基盤となっているネットワークに関する詳しい知識は必要ありません。必要な知識はパートナー LU の名前だけです。その LU の位置は知らなくてもかまいません。 SNA がパートナー LU の位置を判別し、データを送るための最適な経路を決定します。 基盤となるネットワークの変更、新しいアダプターの

追加、またはマシンの再配置によって、APPC プログラムが影響を受けることはありません。 ただし、プログラムによっては、交換 SDLC 接続によるリンク接続の確立が必要になる場合があります。

Communications Server は、始動時に、ローカル LU 定義および論理リンク定義を確立します。これらの定義は構成ファイルに格納されています。 システム管理アプリケーション・プログラミング・インターフェース (API) には、構成定義およびアダプターとリンクの活動化を制御する機能があります。
これらの機能については、*Communications Server: システム管理プログラミング*を参照してください。 ユーザーは、Communications Serverの実行中に構成およびノード操作の機能を使用することができます。構成とノード操作については、*Communications Server: 概説およびインストール* および *Communications Server: システム管理プログラミング*を参照してください。

SNA LU タイプ 6.2 サポート

LU 6.2 はプログラム間通信用のアーキテクチャーです。Communications Server は、LU 6.2 のすべての基本機能をサポートしています。 SNA LU 6.2 のオプション機能には次のものがあります。

- 基本会話とマップ式会話
- 半二重または全二重の会話スタイル
- 同期レベルの確認
- セッション・レベルおよび会話レベルでのセキュリティー・サポート
- 複数 LU
- 並行セッション。これには、リモート・システムを使用してセッション数を変更する機能を含みます。
- 並行複数リンク。各リンクがそれぞれ 1 つのサブエリア・ネットワークに接続できます（1 つの PU 定義につき 1 つずつ)。

第2章 APPC の基本概念

本書では、Communications Server がサポートする APPC API について説明します。説明内容は次の通りです。

- APPC API の構造の概要
- インターフェースを介して発行される verb の特定構文に関する参照情報

トランザクション・プログラムとは？

トランザクション・プログラムは、APPC 通信機能を使用するアプリケーション・プログラムの一部です。　アプリケーション・プログラムは、この機能を使用して、APPC をサポートする他のシステムのアプリケーション・プログラムと通信します。トランザクション・プログラムには 64 バイトの名前 (**tp_name**) が付きます。

トランザクション・プログラムは、次のいずれかの API を介して LU 6.2 サービスを使用することができます。

- APPC（拡張プログラム間通信機能）を使用すると、トランザクション・プログラムは、LU 6.2 セッション用に IBM が定義している構文と verb を使用して IBM SNA ネットワーク上で情報を交換することができます。
- CPI-C（共通プログラミング・インターフェース通信機能）を使用すると、トランザクション・プログラムは、LU 6.2 セッション用に IBM が SAA の共通構成要素に定義している構文を使用して、IBM SNA ネットワーク上で情報を交換することができます。 この API は多くのプラットフォームで実現されているので、CPI-C アプリケーションの移植は容易に行えます。

トランザクション・プログラムは、APPC の機能を呼び出すための APPC verb を発行します。 トランザクション・プログラムが APPC verb を発行する方法の詳細については、43ページの『第5章 APPC トランザクション・プログラムの実行』 を参照してください。トランザクション・プログラムは、CPI 通信の機能を呼び出すためのCPI 通信呼出しを発行することができます。 アプリケーション・プログラムは、CPI 通信呼出しを使用することにより、SAAの一貫性のあるコミュニケーション・サポートを活用できます。

CPI 通信呼出しについては、8ページの『CPI 通信トランザクション・プログラム』を参照してください。

APPC トランザクション・プログラム

APPC トランザクション・プログラムはアプリケーションではなく、アプリケーションの一部を構成します。 1 つのアプリケーションに、多数のトランザクション・プログラムを含めることができます。 どのトランザクション・プログラムにも、それぞれ固有の 8 バイトの識別番号 (**tp_id**) が付きます。

APPC は、アプリケーション内のトランザクション・プログラムを開始したり停止したりするための verb を提供します。また、APPC は、トランザクション・プログラムの機能を実行するための会話 verb のフルセットも提供しています。

トランザクション・プログラムは、アプリケーション・プログラムに関するアクションを実行するための要求を、verb の形で APPC に対して発行します。 verb は、トランザクション・プログラムが発行し APPC が実行する決められた要求形式です。プログラムは、一連の APPC verb を使用して、他のプログラムと通信します。 互いに通信する 2 つのプログラムは、別々のシステムにあっても同じシステムにあってもかまいません。どちらの場合も APPC API は同じです。

本書で説明する APPC verb のセットは、会話を記述できるプログラミング言語と考えてください。 APPC verb は制御ブロックの形でコーディングされ、各 verb の構文は厳密に定義されています。 ユーザー・プログラムは、verb 制御ブロックを APPC API に渡すことにより、APPC 機能へのアクセス権を取得します。

あるトランザクション・プログラムが他のトランザクション・プログラムとデータを交換するとき、両者をパートナー・トランザクション・プログラムと呼びます。

CPI 通信トランザクション・プログラム

CPI 通信トランザクション・プログラムは、APPC トランザクション・プログラムに似ています。どちらのタイプのトランザクション・プログラムも、APPC サポートを使用します。 ただし、CPI 通信トランザクション・プログラムは、verb を発行するのではなく、個々の機能に対する呼出しに適切なパラメーターを指定し、その呼出しを使用して各 CPI 通信機能を呼び出します。

ほとんどの CPI 通信呼出しは APPC verb に対応しています。 たとえば、アウトバウンド会話の割振りと受入れ（受信）を行う呼出し、および会話を使用してデータを送受信する呼出しは、それぞれに対応する APPC verb に似た機能を備えています。ただし、会話を割り振る前に会話を初期化する呼出しと、個々の会話特性を設定し抽出する呼出しは例外です。

Communications Server が CPI 通信プログラム用として提供するサポートの詳細は、*CPI Communications Reference* を参照してください。

クライアント・トランザクション・プログラム

通常、プログラムは、別のプログラムからのサービスが必要になるため、会話を開始します。 サービスを必要とするこのプログラムをクライアント・トランザクション・プログラムと呼びます。クライアント・トランザクション・プログラムは、LU 6.2 API を使用して会話を要求します。

クライアント・トランザクション・プログラムは、しばしば人間のユーザーによって開始されます。しかし、クライアント・トランザクション・プログラムは、実際は、他のプログラムの要求に応答するサーバー・トランザクション・プログラムであることもあります。 どの会話においても、クライアント・トランザクション・プ

ログラムは、常に、会話を開始する前に実行されている必要があります。 クライアント・トランザクション・プログラムの始動と終了は、直接には会話には関係しません。 クライアント・トランザクション・プログラムは、会話を開始し、会話が終了した後も実行し続けることができます。

サーバー・トランザクション・プログラム

サーバー・トランザクション・プログラムは、クライアント・トランザクション・プログラムが要求したサービスを送達します。

サーバー・トランザクション・プログラムは、クライアントが会話を開始するのを待ちながら、実行を続けることができます。 しかし、多くの場合、サーバー・トランザクション・プログラムは、単一のトランザクションを処理し、特定の 1 つの会話を処理するために APPC API によって開始されます。 このようなサーバー・トランザクション・プログラムは会話が要求されると実行を開始し、会話が終了すると終了します。

LU 6.2 アーキテクチャーの重要な機能は、クライアント・トランザクション・プログラムが要求したときにサーバー・トランザクション・プログラムを開始できることです。ユーザーは、このようなモデルにサーバー・プログラムを設計し、要求によってそのプログラムが開始されるようにすることができます。

論理装置とは？

どのトランザクション・プログラムも、*論理装置* (LU) を介して SNA ネットワークへのアクセスを取得します。 LU は、ユーザー・プログラムからの verb を受け入れ、それらの verb に従って働く SNA ソフトウェアです。 トランザクション・プログラムは、対応する LU に対して APPC verb を発行します。 これらの verb に従って、コマンドおよびデータがネットワークを介してパートナー LU に送られます。 LU は、トランザクション・プログラムとネットワークとの仲介として、トランザクション・プログラム間でのデータ交換を管理する役割も果たします。 1 つの LU が、複数のトランザクション・プログラムに対してサービスを提供することもできます。また、複数の LU がノード内で同時に活動状態になることも可能です。

LU のタイプ

Communications Server は LU タイプ 0、1、2、3、および 6.2 をサポートします。LU タイプ 0、1、2、3 は、ホスト・アプリケーション・プログラムと、いくつかの異なる種類の装置（端末やプリンターなど）との間の通信をサポートします。 このようなプログラムの作成方法の詳細は、第2部 LUA APIを参照してください。

LU 6.2 は、タイプ 5 のサブエリア・ノード、タイプ 2.1 の周辺ノード、またはその両方にある 2 つのプログラム間、およびそれらのプログラムと装置との間の通信をサポートします。 APPC は LU 6.2 アーキテクチャーを実現するものであり、これについては本書のこの部で説明します。

通信は、同じ LU タイプの LU 間でのみ行われます。 たとえば、LU 2 は別の LU 2 とは通信しますが、LU 3 とは通信しません。

従属 LU と独立 LU

従属 LU は、セッションを活動化する際にシステム・サービス制御点 (SSCP) に依存する LU です。 従属 LU は活動状態の SSCP-LU セッションを必要とし、LU はそのセッションを使用して、サブエリア・ノード内の LU との LU-LU セッションを開始します。 従属 LU は、サブエリア LU との間に一度に 1 つしかセッションを確立できません。 サブエリア・ノードのトランザクション・プログラムとの通信の場合、各従属 LU は一度に 1 つしか会話を持つことができず、また、各従属 LU は一度に1つのトランザクション・プログラムに対してしか通信をサポートすることができません。

独立 LU は、セッションを活動化する際に SSCP に依存しません。 独立 LU は、サブエリア・ノード内の他の LU との複数の並行セッションをサポートするので、複数の会話を持つことができ、サブエリア・トランザクション・プログラムとの通信に複数のトランザクション・プログラムをサポートすることができます。 周辺ノード間の LU もこのサポートを使用します。

従属 LU と独立 LU を区別する意味があるのは、周辺ノード内の LU とサブエリア・ノード内の LU の間のセッションについて述べるときだけです。 その他の場合は、タイプ 2.1 の周辺ノード間（たとえば 2 つのワークステーション間）で通信するとき、従属 LU と独立 LU のどちらも、複数の並行セッションおよび会話をサポートします。 Communications Server LU は、従属 LU との間には 1 つのセッションを、そして独立 LU との間には複数のセッションをサポートします。

LU 名とは？

LU は、システム・ネットワーク体系 (SNA) のネットワークにアクセスするためのポイントです。LU は、SNA ネットワーク全体を通じて構成（正式に記録）される名前とその他の特性をもちます。この構成は、ネットワーク管理者が構成し、構成ファイルに記録された静的な構成であることもあり、 また、ファイルまたはユーザー入力によって作成された動的な構成であることもあります。

会話をオープンするためには、クライアント・トランザクション・プログラムは、サーバー・トランザクション・プログラムの名前と、そのサーバー・トランザクション・プログラムが到達できる LU の名前の両方を指定する必要があります。 これらの名前は、クライアント・トランザクション・プログラムに組み込まれていることもあります。 あるいは、組み込まれずに、クライアント・トランザクション・プログラムの外部に記憶されたり、動的に指定されたりすることもあります。

セッションとは？

トランザクション・プログラムが互いに通信するには、それぞれの LU がセッション と呼ばれる相互関係で互いに接続されていることが必要です。 セッションは 2 つの LU を接続するもので、*LU-LU* セッションと呼ばれます。 11ページの図 2 はこの通信の関係を示しています。同じ 2 つの LU の間に確立された複数の並行セッションを、*並列* LU-LU セッションと呼びます。

セッションは、SNA ネットワーク内の一対の LU 間のデータの動きを管理するパイプとしての働きをします。 具体的には、セッションは、伝送するデータの量、データ・セキュリティー、ネットワーク経路指定、通信量の混雑状態などの事項を取り扱います。

図 2. 2 つの LU 間のセッション

セッションはそのセッションの LU により管理されます。 一般に、セッション特性はトランザクション・プログラムでは取り扱いません。 セッションの特性は、次の時点で定義します。

- システムを構成するとき
- 管理 verb を使用するとき

会話とは？

トランザクション・プログラム間の通信を *会話* と呼びます。 会話は LU-LU セッションを介して行われます。 会話は、会話を割り振る APPC verb または CPI 通信呼出しをトランザクション・プログラムが発行した時点で開始されます。 会話に関連した会話スタイルは、使用するデータ転送のスタイル、つまり両方向交互転送か両方向同時転送かを示します。 両方向交互スタイルのデータ転送を指定する会話を、*半二重* 会話と呼びます。 両方向同時スタイルのデータ転送を指定する会話は、*全二重* 会話と呼ばれます。

12ページの図 3 は、セット・アップされた後の会話を示しています。

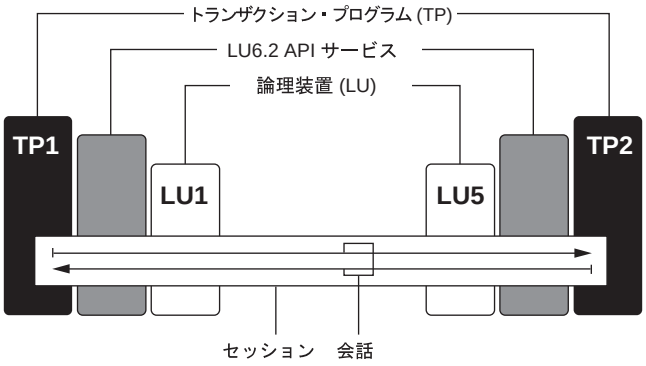


図 3. 会話の構成部分

半二重会話がセッションに割り振られると、その会話に接続されるトランザクション・プログラムの間で送受信関係が確立され、両方向交互データ転送が起こります。その場合、一度には一方向ずつ、両方向に情報が転送されます。 電話での会話の場合と同様に、1 つのトランザクション・プログラムが相手を呼び出し、一時点でどちらか一方のトランザクション・プログラムが話す形式で「会話」を交わし、どちらかのトランザクション・プログラムが打ち切るまでこの会話が続けられます。 一方のトランザクション・プログラムがデータを送信するための verb を発行し、相手のトランザクション・プログラムがデータを受信するための verb を発行します。 送信側トランザクションは、送信を終えると、会話の送信制御権を受信側のトランザクション・プログラムに渡すことができます。 どちらか一方のプログラムが会話を終了すべき時点を決定し、会話が終了したことを相手に知らせます。

全二重会話がセッションへ割り振られると、その会話へ接続されたトランザクション・プログラムは送信および受信状態で始動され、両方向同時データ転送が起こります。その場合、情報は同時に両方向で転送されます。 両方のトランザクション・プログラムは verb を発行してデータを同時に送信および受信することができ、送信制御の転送を必要としません。両方のトランザクション・プログラムが、データの転送を停止する準備ができたことを表示し、各トランザクション・プログラムが、パートナーから送られたデータを受け取ったとき、会話は終了します。 エラー条件が起こると、一方のトランザクション・プログラムは、会話の両側をただちに終わらせることができます。

会話では、制御情報とデータを交換することができます。 トランザクション・プログラムは、アプリケーションにとって最も適した会話スタイルを選択する必要があります。

13ページの図 4は、セッションを介して行われる 2 つのトランザクション・プログラム間の会話を示しています。

図 4. 2 つのトランザクション・プログラム間の会話

1 つのセッションがサポートするのは一度に 1 つの会話だけですが、1 つのセッションが順番に複数の会話をサポートすることはできます。 複数の会話がセッションを再利用することになるので、セッションは会話に比べて接続時間が長くなります。プログラムが会話を割り振ったときに、適用可能なすべてのセッションが使用中であった場合は、LU は、着信 Attach（割振り要求）を待ち行列に入れます。 そして、いずれかのセッションが使用可能になった時点で、割振りが完了します。 接続マネージャーについては、第3章 接続マネージャーの使用,

２つの LU は互に並列セッションを確立して、複数並行会話をサポートすることもできます。 並列セッションが生じるのは、どちらかのトランザクション・プログラムが会話を割り振ったときに、存在しているセッションが別の会話で使用中だった場合です。 LU は、新しいセッションを要求して割振りを完了することができます。

13ページの図 5 は、2 つの LU 間の 3 つの並列セッションを示しています。各セッションで、それぞれ会話が維持されています。

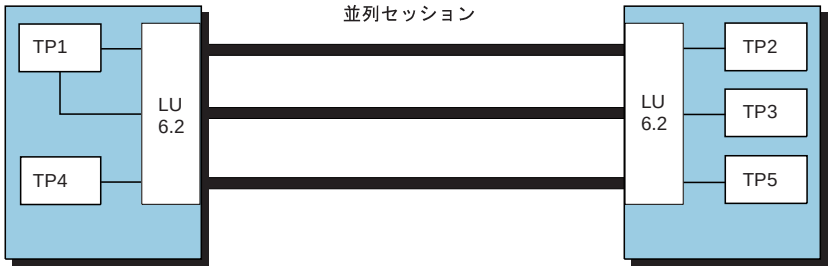


図 5. LU 間の並列セッション

APPC 会話は、*半二重*の会話です。 ある一時には、2 つのパートナー・トランザクション・プログラムのいずれか一方だけにデータを送信する権利があります。 そのトランザクション・プログラムは*送信状態*にあります。 もう一方のトランザクション・プログラムにはデータを受信する責任があります。 このプログラムは*受信状態*にあります。 トランザクション・プログラムは、指定の時間になると、この任務を交代します。 最初に会話がセット・アップされたときは、クライアント・トランザクションは送信状態になっており、サーバー・プログラムは受信状態になっています。

セッション、会話および LU の関係

LU 間の接続をセッションと呼びます。この接続は、中間ネットワーク・ノードを使用して渡すことができます。 ただし、LU 6.2 プログラムでは、接続の詳細を説明する必要はありません。 サーバー・トランザクション・プログラムがクライアント・トランザクション・プログラムと同じ部屋にあっても、または数千キロ離れたところにあっても、それはクライアント・トランザクション・プログラムにとっては何の違いもありません。 LU 6.2 API は、タイプ 6.2 の LU 間のセッションの開始と終了を行います。

セッションは、一時には会話は 1 つしか行えませんが、最初の会話が終了すると、そのセッションで別の会話を行うことができます。 LU 6.2 ソフトウェアは、会話が終了したときに、セッションを終了するか、またはオープンしたままにしておいて別の会話にそれを使用するかを判別します。

LU の中には、複数の並列セッションを処理できるものがあります。 各セッションはそれぞれ独立しています。14ページの図 6 は、マシン、LU、セッションおよびトランザクション・プログラムの間のいろいろな関係を示しています。

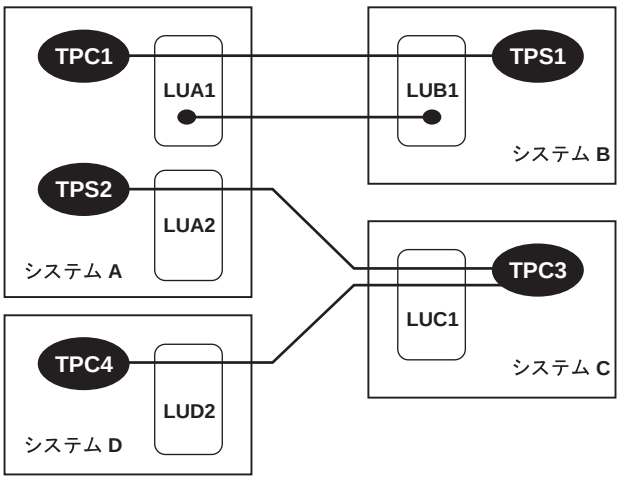


図 6. プログラムと LU との関係

14ページの図 6 は、システム A の LUA1 とシステム B の LUB1 との間の 2 つの並列セッションを説明しています。1 つのセッションでは、クライアント TPC1 とサーバー TPS1 の間の会話が行われています。 もう 1 つのセッションは、ここでは会話に使用されていません。

システム C では、LUC1 がやはり 2 つの並列セッションをサポートしています。これらのセッションはともにクライアント TPC3 で使用されています。この TPC3 クライアントでは、システム A のサーバー TPS2 との会話が行われていますが、システム D の TPC4 との会話も進行しています。この図は、トランザクション・プログラムでは会話は 1 つだけに限定されないことを示しています。また、この図を見ると、同一プログラムがクライアントにもサーバーにもなれることがわかります。 この会

話の例では、プログラム TPC4 がサービスを要求してプログラム TPC3 を開始したと考えられます。 そして、そのサービスを送達するために、TPC3 は TPS2 からのサービスを要求しています。

APPC 操作の例

15ページの表 1 は、LU 6.2 の操作を簡略用語を使用して説明しています。

表 1. LU 6.2 の操作

操作	行われる処理
Send	データ・ブロックを他のプログラムに送信する。
Receive	送信状態になっている場合は、バッファー内に出力データがあればそれを伝送し、受信状態に入る。 データが到着するのを待機し、データを受け取る。
Await confirmation	バッファー内に出力データがあればそれを伝送する。 パートナー・プログラムがすべてのデータを受信し、処理したことを確認するまで待つ。
Confirm	すべてのデータを受信し、処理したという確認を、パートナー・プログラムに送る。
Error	受信状態の場合は、バッファー内の入力データを除去し、送信状態に入る。 送信状態になっている場合は、バッファー内の出力データをすべて除去する。 パートナー・プログラムの現在の操作を特別な戻りコードを付けて終了させる。
Close	送信状態になっている場合は、バッファー内の出力データがあればそれを伝送する。 会話を終了する。

LU 6.2 API は、どちらの側においてもこれらのサービス（および他のサービス）を提供します。また、どちらの側でも、ユーザーがこれらの基本操作を組み合わせてパフォーマンスを向上させることができるようなサービスを提供します。以下の節では、会話のタイプを説明するときに、各 API の詳細な相違に言及しなくてもすむように、これらの用語を使用しています。 たとえば、15ページの表 1 で使用している用語 *Send* は、APPC verb の SEND_DATA または MC_SEND_DATA、または CPI-C 機能の CMSEND を指すものとします。

APPC 会話のタイプ

この節では、APPC 会話のタイプを説明します。

- 片方向
- 送達確認
- 照会
- データベース更新

片方向の会話

もっとも単純なタイプの会話である片方向会話では、クライアント・トランザクション・プログラムはデータをサーバーに渡し、サーバーはそれに注目します。16ペー

ジの表 2 はこれを要約したものです。

表 2. 片方向会話のアクション

クライアントのアクション	サーバーのアクション
1 つまたは複数のレコードを *Send*。	
Close。	レコードを *Receive*し、処理。 *Close*。

この最小タイプの会話は、データの送達が最重要事項ではないデータの場合に使用されます。たとえば、状況表示の定期的な更新、使用レベルの記録、または状態のログなどがこれに当たります。

送達確認会話

次に単純な会話のタイプは、送達確認会話です。この会話では、クライアント・トランザクション・プログラムはレコードを送信し、サーバーはその受信を確認します。16ページの表 3 はこれを要約したものです。

表 3. 送達確認会話のアクション

クライアントのアクション	サーバーのアクション
1 つまたは複数のレコードを *Send*。	
Await confirmation。	
	レコードを *Receive*し、処理。 レコードの *Confirm*。
Close。	*Close*。

このタイプの会話は、クレジット許可システム（クライアントはアカウント番号と購入金額を送信し、サーバーの確認がその売買を正当化する）で使用することができます。 たとえば、クライアント・トランザクション・プログラムはどのような種類のデータベース・レコードも送信することができ、サーバーはデータベースが更新されていることを確認できます。 クライアントが送信できるデータの量には上限がないので、このタイプの会話は、ファイル全体のデータをバッチ・モードで送信するのに使用することができます。 このタイプの会話では、クライアント・トランザクション・プログラムは確認だけを受信します。その他のデータを戻してもらう必要はありません。

Confirm 操作と *Send* 操作の相違は、*Confirm* が伝送するのはもっとも短い SNA メッセージ、すなわち、すべてのデータが受信され、処理されたという肯定応答だけであるという点です。

照会会話

照会会話では、クライアントは情報を求める要求を 1 つ送信し、サーバーは応答を1つ生成します。これを17ページの表 4 に要約してあります。(照会も応答も、任意の数の論理レコードで構成することができます。) このタイプの会話は、多くの種類のデータ処理アプリケーションで使用されます。

表 4. 照会会話のアクション

クライアントのアクション	サーバーのアクション
1 つまたは複数のレコードを *Send*。	
Receive。	
	レコードを *Receive*し、処理。
	1 つまたは複数のレコードから成る応答を *Send*。
すべての応答データが到着するまで *Receive*を続行。	*Close*。
Close。	

トランザクションをこのようなモデルに設計すると、サーバー・トランザクション・プログラムは非常に単純なものになります。 すなわち、各プログラムは 1 種類の照会インスタンス 1 つを処理し、終了します。 クライアント・トランザクション・プログラムは、必要な照会のタイプに応答できるサーバー・トランザクション・プログラムとの会話を要求します。 LU 6.2 API サービスは、そのようなサーバー・トランザクション・プログラムを探し、そのコピーを開始します。

データベース更新会話

データベース更新会話では、クライアント・トランザクション・プログラムは、データのコピーを要求し、それを修正した後、格納してもらうためにそれを戻します。サーバー・トランザクション・プログラムは、更新が完了するまで、クライアントが使用できないようにそのデータをロックします。17ページの表 5 は、この場合のクライアントとサーバーのアクションを要約したものです。

表 5. データベース更新会話のアクション

クライアントのアクション	サーバーのアクション
データ要求（レコード・キー）を *Send*。	
Receive。	
	キー値を *Receive*。
	レコードを取り出し、ロック。
	レコードのコピーを *Send*。
	Receive。
受信したレコードを処理。	
更新したレコードを *Send*。	
Await confirmation。	
	受信したレコードでデータベースを更新。
	更新を *Confirm*。
Close。	*Close*。

このプロセスを明確にするには、15ページの表 1 を参照してください。 クライアント・トランザクション・プログラムが最初に *Receive*を発行すると、次の 3 つが行われます。

- LU 6.2 送信バッファーが、クライアントの送信した残りの論理レコードを示してフラッシュされる。
- クライアント・トランザクション・プログラム（これは送信状態で開始される）が受信状態に切り替えられる。送信権がサーバー・トランザクション・プログラムに渡されます。

- クライアント・トランザクション・プログラムは、データの到着を待つ。(非ブロッキングの受信操作も利用できます。)

同じように、サーバーが発行した 2 番目の *Receive* がサーバー・バッファーをフラッシュし、送信権がクライアント・トランザクション・プログラムに戻されます。

エラーが発生した会話

会話では、エラーは避けることができないものです。したがって、トランザクション・プログラムには、エラーを検出し、それに応答する準備が必要です。 トランザクション・プログラムでは、エラーの検出を通知するために *Error* 命令が使用されます。15ページの表 1 を参照してください。 18ページの表 6 は、サーバーが照会のときに論理エラーを検出した場合の照会会話を要約したものです。

表 6. エラーが発生した照会会話

クライアントのアクション	サーバーのアクション
1 つまたは複数のレコードを *Send*。	
Receive。	
	いくつかの照会レコードを *Receive* し、処理。 エラーを検出。
	Error。
	診断エラー・メッセージを *Send*。
*Receive*の戻りコードが、パートナーによる *Error*を指示。	*Close*。
診断メッセージを *Receive*。ユーザーに表示。	
Close	

Error 操作の主な目的は、両方のトランザクション・プログラムの API バッファーに送信されなかった、または受信されなかったデータが入っている場合に、それをすべて除去することです。 *Error* 操作は、また、エラーを検出したトランザクション・プログラムに対して、パートナーに診断データを伝送できるように送信権を与えます。 トランザクション・プログラムには、この診断メッセージの内容と、それ以降の操作を指定しておく必要があります。

会話タイプ

Communications Server LU 6.2 では、マップ式会話と基本会話の 2 種類の会話がサポートされており、その各々に対して別のセットの verb が用意されています。 使用する会話タイプは、基本会話が提供する SNA *汎用データ・ストリーム (GDS)*に対する完全なアクセスが必要であるかどうかによって決まります。 GDS は、GDS 変数と呼ばれるものを定義しています。GDS 変数は、1 つまたは複数の論理レコードから構成されます。 各論理レコードは、その論理レコードの長さを指定する*論理長 (LL)* フィールドで始まります。GDS 変数の最初の論理レコードには、論理長フィールドの直後に*識別 (ID)*フィールドが続き、その GDS 変数のタイプを明らかにしています。

マップ式会話

交換されるデータの最終ユーザーになるトランザクション・プログラムには、マップ式会話を使用します。 マップ式会話を使用すると、拡張プログラム間通信を、使いやすいレコード・レベル方式で利用することができます。マップ式会話を使用するトランザクション・プログラムではデータを記述する GDS ヘッダーが必要でないため、プログラムでこれらのヘッダーを作成したり、解釈したりする必要がありません。 トランザクション・プログラムでマップ式会話を使用すると、Communications Server LU 6.2 が GDS 変数を作成し、解釈します。

マップ式会話を使用するプログラムでは、レコードはユーザーが設計するどのようなフォーマットにも交換されます。

- 各送信操作は、0 バイトから 65,535 バイトまでの指定された長さのレコードを扱います。 Communications Server はそのレコードを 1 つの GDS 変数にフォーマットします。
- 受信操作は、プログラムがどれだけのバッファー・スペースを割り振っているかによって、1 つの送信レコード（ヘッダー・フィールドのない GDS 変数）の全体または一部を戻します。 戻りコードは、パートナー・プログラムが送信したレコードの最後の部分が受信されたときを示します。

APPC API は、以下のタスクを完全に行います。

- 複数のレコードのブロッキングとバッファリング
- データの SNA GDS 変数へのフォーマット
- 受信側プログラムでのバッファリング
- ブロッキングの解除と Receive 操作への送達

基本会話

基本会話では、トランザクション・プログラムは 0 から 32,765 バイトまでの長さの論理レコードを交換します。

- 各送信操作は、0 バイトから 65,535 バイトまでの論理レコードを含むバッファーを扱います。 このバッファーは、1 つまたは複数の論理レコードおよびレコード部分を含むことができます。 論理レコードは、いくつかの送信呼出し間で分割することができます。
- 受信操作では、単一論理レコードを受け入れること、または、1 つまたは複数の論理レコードとレコード部分が入っている 1 つのバッファーを受け入れることができます。

要約

2 つのトランザクション・プログラムが LU 6.2 を使用して、会話でデータを交換します。 一方のプログラム、クライアント・トランザクション・プログラムは、通常はユーザーが開始します。 もう一方のプログラム、サーバー・トランザクション・プログラムは、クライアントによって自動的に開始されて、クライアントにサービ

スを提供します。 トランザクション・プログラムは、2 種類の API、すなわち、APPC または CPI-C のどちらかを使用します。これらは、スタイルは異なりますが、同一ではないが似ているサービス・セットをもっています。

会話は、2 つの LU の間のセッションで行われます。 LU は、トランザクション・プログラムが SNA ネットワークをアクセスできるポイントを表します。 セッションは、2 つの LU の間の接続を表します。このとき、それらの位置やその間の距離は関係しません。

第3章 接続マネージャーの使用

LU 6.2 の重要な機能の 1 つに、あるノードのプログラムが他のノードの対応するプログラムを始動できるという機能があります。 このようにプログラムを始動する着信要求を取り扱うのが、*接続*マネージャー です。

この章では、パートナー・プログラムの要求に応じて始動されるローカル・ワークステーションのプログラムについて記述します。 このようなローカル・プログラムを*リモート始動型* プログラムといいます。 セキュリティーと資源制御の観点から見て、どのプログラムをリモート開始できるかをワークステーション・ユーザーおよび管理者が制御できることが必要です。 データを破壊したり重大な局面でローカル・ワークステーションのメモリーを使用したりするようなプログラムを、リモート・ノードのユーザーが開始できるようにすべきではありません。接続マネージャーは、ローカル・ワークステーションでプログラムを開始するための着信要求を処理する門番のような働きをします。

接続マネージャーの名前は、一対の LU 間を流れる *Attach*（接続）と呼ばれる SNA メッセージからとられたものです。 Attach が流れるのは、パートナー LU を使用するプログラムが会話を始動したときです。 ローカル・ワークステーション内の LU 6.2 構成要素は、Attach を受け取ると、それを接続マネージャーに渡して処理を任せます。 このようにして受信される Attach は、*着信割振り要求* または*着信 Attach* と呼ばれます。 この章で、*着信割振り要求* という言葉は、パートナー LU が SNA Attach を生成したことを意味します。

接続マネージャーは次のことを行います。

- リモート・ノードがローカル・ワークステーション内のアプリケーションを始動できるようにする。 1 つのプログラムの複数のインスタンスを、連続して（待ち行列化して）始動することも、並列して（待ち行列化しないで）始動することもできます。
- リモート始動するプログラムにパラメーターを渡す。
- ウィンドウまたはバックグラウンドでプログラムを始動する。
- セキュリティー・ガイドラインに基づいて着信割振り要求を検査する。
- 着信割振り要求をクライアント・ワークステーションに転送する。
- 着信割振り要求の会話タイプ（基本かマップ式か）および同期レベルを検査する。
- サーバー・プログラムに対して、着信割振り要求およびローカルに発行された APPC **RECEIVE_ALLOCATE** verb または CPI 通信 Accept_Conversation または Accept_Incoming (CMACCP、CMACCI) 呼出しを保留するためのタイム・アウト値を指定する。

図 7 は接続マネージャーの機能を示しています。

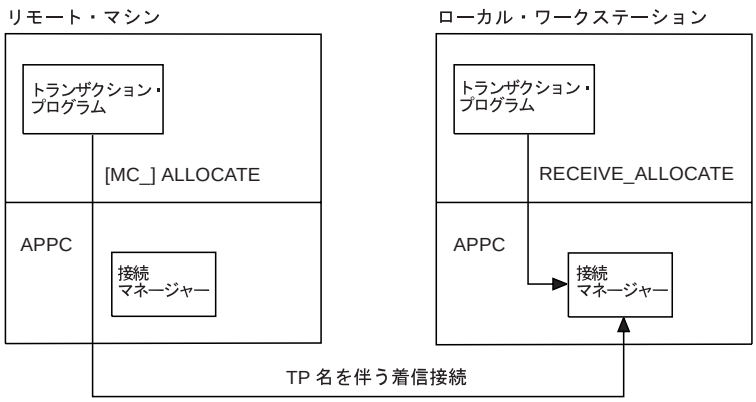


図 7. APPC での接続マネージャーの機能

互いに通信する一対のトランザクション・プログラムで、接続マネージャーを必要とするのは割振り要求を受信するノードだけです。 接続マネージャーは、次の 3 種類の入力を管理します。

- パートナー・トランザクション・プログラムからの着信割振り要求 (Attaches)。
- ローカル・プログラムからの APPC **RECEIVE_ALLOCATE** verb、または CPI 通信の CMACCP 呼出しおよび CMACCI 呼出し。
- トランザクション・プログラム、ユーザー ID、およびパスワードの構成定義。

TP 名 は、着信割振り要求の中の主要な情報の 1 つです。 接続マネージャーは、このトランザクション・プログラム名を使用して、ローカル・ワークステーションのどのプログラムを開始するのかを判断します。 両方のノードのプログラマーおよび管理者が、各トランザクション・プログラム名について合意に達していることが必要です。 割振り要求を発行するプログラムは、トランザクション・プログラム名を、APPC の **[MC_]ALLOCATE** verb または **[MC_]SEND_CONVERSATION** verb のパラメーターとして渡します。

Attach を受信すると、その Attach の中のトランザクション・プログラム名と、トランザクション定義によって定義されているトランザクション・プログラム名との突合せが行われます。一致する名前があった場合は、その定義に基づく名前の実行ファイルが開始されるか、またはクライアント・ワークステーションに送られます。一致する名前がなかった場合は、実行ファイルの名前は、Attach に指定されている名前に **.EXE** を付加したものとみなされます。

アプリケーションとトランザクション・プログラムとの相違

APPC では、*トランザクション・プログラム*という用語には特殊な意味があります。トランザクション・プログラムはアプリケーションではなく、アプリケーションの一部を構成しています。

アプリケーションが APPC の **RECEIVE_ALLOCATE** verb または **TP_STARTED** verb を正常に発行すると、トランザクション・プログラムが始動されます。いずれの方

法でも、そのトランザクション・プログラムを、APPC が認識すべき新しいトランザクション・プログラムとして識別します。 APPC は、トランザクション・プログラム用として一連のメモリー・ブロックを確保し、固有のトランザクション・プログラム識別子 **tp_id** を作成し、それを呼出し元プログラムに戻します。

アプリケーションは、トランザクション・プログラム内で発行するすべての会話verbで、取得した **tp_id** を指示する必要があります。アプリケーションが **TP_ENDED** verb を発行すると、APPCはそのトランザクション・プログラム用のバッファーを消去し、**tp_id** に無効にします。アプリケーションが終了すると、APPC はそのプロセスに関連している活動状態のトランザクション・プログラムをすべて終了します。 接続マネージャーは、割振り要求を受け取り、それが有効であると確認できた場合に、**RECEIVE_ALLOCATE** が保留中でなければ、着信トランザクション・プログラム名に対応するアプリケーションを開始します。接続マネージャーは、アプリケーション・プログラムを開始するのであって、トランザクション・プログラムを開始するのではないという点に注意してください。 一般的に、アプリケーションは次に、自分自身をトランザクション・プログラムとして確立する verb を発行します。 送信側ノードとローカル・ワークステーションが相互に合意すれば、接続マネージャーを構成して、ローカル・アプリケーション内の*任意の* アプリケーションを始動できます。

同様に、会話にはそれぞれ専用の識別子があり、会話を割り振るには、その前にトランザクション・プログラムが確立されていることが必要です。 複数の会話が 1 つの**tp_id** を、同時に（複数スレッドの場合など）または順次に（会話が順に続く場合）使用できます。トランザクション・プログラムが終了すると、APPC は活動状態にある会話の割振りをすべて解除します。

トランザクション・プログラム定義

Communications Serverは、 2 つの命名レベルを使用して、リモート始動するプログラムを識別します。

- パートナー・トランザクション・プログラムによって認識されている 64 文字のローカル・プログラム名 (**tp_name**)
- 始動するローカル・プログラムのファイル指定 (filespec)

この 2 つの名前を使用することで、融通性に富んだ再構成が可能となり、ワークステーション間での APPC プログラムの移植性が強化されます。

TP 名 パートナー・トランザクション・プログラムが、割振り要求でローカル・ワークステーションの接続マネージャーに送る名前です。

パートナー・トランザクション・プログラムとローカル・プログラムは、どちらもトランザクション・プログラム名を認識していることが必要です。トランザクション・プログラム名は、ローカル LU のプログラムで使用する **RECEIVE_ALLOCATE** の指定パラメーターです。パートナー・トランザクション・プログラムは、APPC の **[MC_]ALLOCATE** verb または **[MC_]SEND_CONVERSATION** verb でトランザクション・プログラム名を指定します。

パス名 トランザクション・プログラム・ファイル指定（パス名）は、ローカルで始動するプログラムの名前を示します。 トランザクション・プログラム・ファイル指定には、実行ファイルのドライブ、パス、ファイル名、および拡張子が含まれます。

複数のトランザクション・プログラム定義で、同じトランザクション・プログラム・ファイル指定を指定することができます。 接続マネージャーは、プログラムの１つのインスタンスを実行するのか複数のインスタンスを実行するのかを判断する必要があるので、該当のトランザクション・プログラムを指定するすべての 定義の中で、そのトランザクション・プログラム・ファイル指定を、待ち行列型または非待ち行列型のいずれかとして構成しておくことが必要です。 たとえば、**MYTP.EXE** を指定する定義が「待ち行列－接続マネージャー始動」として構成されているとすれば、他のトランザクション・プログラム定義の中で、**MYTP.EXE** を非待ち行列型として構成することはできません。ただし、トランザクション・プログラムのファイル指定には大文字小文字の区別があります。

両マシンのトランザクション・プログラム名の識別

接続マネージャーにより識別されたプログラムが開始できない場合は、接続マネージャーは割振り要求を拒否します。割振り要求を発行したプログラムには、接続マネージャーがプログラムを開始できなかったことを示す通知が送られます。

ユーザーまたは管理者は、Communications Serverの構成時にトランザクション・プログラムを定義し、定義済みトランザクション・プログラム名のリストを作成します。省略時値をそのまま使用する場合を除き、パートナーから受け入れる個々の固有トランザクション・プログラム名ごとに、ローカル（受入れ側）ワークステーション内にトランザクション・プログラム定義が必要です。 トランザクション・プログラム定義には、トランザクション・プログラムに関する情報が入っています。 同様に、構成時には、LU 6.2 会話セキュリティー情報をもとにして、セキュリティー情報のリスト（許容されるパスワードおよびユーザー ID）が作成されます。 Communications Server の構成については、*Communications Server: 概説およびインストール*を参照してください。以下に、トランザクション・プログラムを定義するために指定する必要のある構成データを説明します。

会話属性の定義

会話パラメーター **sync_level**、**conv_type**、および **security_rqd** は、接続マネージャーがプログラムを始動する方法に対して直接的な影響を与えるものではありません。しかし、接続マネージャーは、着信割振り要求を待ち行列に入れる前、または対応する **RECEIVE_ALLOCATE** verb について検査する前に、着信割振り要求を拒否すべきかどうかを、これらのパラメーターに基づいて判断します。

同期レベル

sync_level を定義するときに、トランザクション・プログラムが確認処理用の verb およびパラメーターをサポートするかどうかを指定してください。
この種の APPC verb には、**[MC_]CONFIRM** と **[MC_]CONFIRMED** があります。**[MC_]ALLOCATE**、**[MC_]SEND_CONVERSATION**、**[MC_]PREPARE_TO_RECEIVE**、および **[MC_]DEALLOCATE** には、確認処理用のパラメーターがいくつかあります。共通プログラミング・インターフェース通信 (CPIC) のユーザーの場合は、**Set_Sync_Level** (CMSSL) 呼出しにより **sync_level** を設定できます。

着信割振り要求には、トランザクション・プログラムが確認処理用の verb またはパラメーターを発行するかどうかを示すフィールドが含まれています。接続マネージャーは、着信割振り要求のこのフィールドを、トランザクション・プログラム定義のリスト内の構成値と比較して検査します。 両者の値が一致しない場合は、接続マネージャーは着信割振り要求を拒否します。構成選択項目には次のものがあります。

NONE トランザクション・プログラムは、どの会話でも確認処理に関連した verb を一切発行しません。

CONFIRM
トランザクション・プログラムは、会話の中で確認処理を行うことができます。トランザクション・プログラムは、確認に関する verb を発行し、その戻り値を認識することができます。トランザクション・プログラムに確認処理用の verb が含まれている場合は、それと互換性のあるセッションを確保するために、**sync_level**(CONFIRM) を定義してください。

EITHER
トランザクション・プログラムは、パートナーが確認処理を指定していてもいなくても、そのパートナーとの会話に参加できます。構成しているトランザクション・プログラムで確認処理が必要な場合は、EITHER を選択しないでください。

会話のタイプとスタイル

conv_type パラメーターは、始動するプログラムの会話タイプおよび会話スタイルの両方を指定するために使用します。会話タイプ属性は、始動するプログラムが、データの送受信時に基本レコードまたはマップ式レコードのどちらをサポートするのかを決定します。 会話スタイル属性は、始動するプログラムが半二重会話をサポートするかどうかを決定します。 接続マネージャーは、トランザクション・プログラムが基本 verb とマップ式 verb のどちらを使用しているのか、そして半二重と全二重のどちらを使用するかをチェックします。

会話タイプは次のいずれかです。

BASIC
トランザクション・プログラムは、会話の場合に基本会話 verb のみを発行します。

MAPPED
トランザクション・プログラムは、会話の場合にマップ式会話 verb のみを発行します。

EITHER
トランザクション・プログラムは、会話の場合に、着信割振り要求で何が到着するかに応じて、基本会話 verb またはマップ式会話 verb のいずれかを発行します。

二重のタイプは次のとおりです。

HALF トランザクション・プログラムは半二重会話のみをサポートします。

FULL トランザクション・プログラムは全二重会話のみをサポートします。

EITHER
トランザクション・プログラムは全二重会話または半二重会話のいずれかをサポートします。

着信割振り要求での会話セキュリティー

トランザクション・プログラム定義では、着信割振り要求でパスワードおよびユーザー ID を提供する必要があることを指定できます。 パスワードおよびユーザーID は、**[MC_]ALLOCATE** verb と **[MC_]SEND_CONVERSATION** verb、または CPIC 呼出し Set_Conversation_Security_UserID (CMSCSU) および Set_Conversation_Security_PassWord (CMSCSP)の任意指定パラメーターです。ローカル・トランザクション・プログラム定義で会話セキュリティーを指定してある場合は、接続マネージャーは、着信割振り要求のパスワードおよびユーザー ID の妥当性をチェックします。 ユーザー ID およびパスワードが存在しない場合、またはそれらがパスワードとユーザー ID の有効な組合せに一致しない場合は、接続マネージャーは割振り要求を拒否します。

パスワードおよびユーザー ID を伴う着信割振り要求が到着した場合は、トランザクション・プログラム定義で会話セキュリティーが不要なことを指定してあっても、接続マネージャーはその割振り要求の妥当性をチェックします。 そして、パスワードおよびユーザー ID がリスト内の有効な組合せのどれにも一致しない場合は、その割振り要求を拒否します。 つまり、到着した割振り要求にパスワードまたはユーザー ID が含まれている場合、それが無視されることは決してありません。

発信割振り要求での会話セキュリティー

リモート始動トランザクション・プログラム（他のトランザクション・プログラムにより始動されるプログラム）は、３番目のトランザクション・プログラムに会話を割り振る前に、ユーザー ID とパスワードの妥当性をチェックすることができます。このような場合は、**[MC_]ALLOCATE** verb および **[MC_]SEND_CONVERSATION** verb の **security**(SAME) パラメーターにより、会話セキュリティーがすでにチェック済みであることを示すことができます。 ２番目の Attach は、最初の会話を開始した１番目の Attach から、自動的にユーザー ID を入手します。

APPC は現行ユーザー ID を入手し、ユーザー ID がチェック済みであることを示す標識とともにその ID を送信することができます。 **[MC_]ALLOCATE** verb または **[MC_]SEND_CONVERSATION** verb のどちらかで **security**(SAME) パラメーターを使用するローカル始動トランザクション・プログラム用の Attach では、パートナーは妥当性チェック済みの標識の受入れができることが必要です。

ユーザー ID とパスワードの使用については、*Communications Server: システム管理プログラミング*を参照してください。

Communications Server での接続マネージャーの使用

以下の節では、Communications Server マシンに置かれているプログラムの始動方法を説明します。

接続マネージャーの始動

ユーザーは、SNA ノードが活動状態にあるときに、接続マネージャーを始動したり停止したりできます。始動された接続マネージャーは、着信 Attach の処理を始めます。 停止の際には、接続マネージャーは待ち行列内にある Attach をすべて除去します。 該当する verb については、*Communications Server: システム管理プログラミング*を参照してください。

接続マネージャーを始動する必要があるのは、リモート始動するトランザクション・プログラムを実行するノード内だけです。 そのノード内のすべてのトランザクション・プログラムが会話を始動する（つまりすべてのプログラムが APPC の **[MC_]ALLOCATE** verb または **[MC_]SEND_CONVERSATION** verb を発行する）場合は、接続マネージャーを始動する必要はありません。 Communications Serverのノード操作機能を使用することで、許可ユーザーはいつでも接続マネージャーを始動または停止することができます。 許可プログラムは、ノード操作 verb である Enable Attach Manager および Disable Attach Manager を使用して、接続マネージャーを開始または停止します。

接続マネージャーによるプログラムの始動

接続マネージャーは、ワークステーション上のプログラムを始動するときに、定義済みトランザクション・プログラム・リスト中の **load_type** フィールドを使用して、プログラムをどのように実行するかを決定します。リモート始動するプログラムは、次のいずれかで始動するように構成できます。

Console
ウィンドウを表示するか、または全画面 DOS アプリケーションとして実行されるアプリケーション。

Background
プログラムはバックグラウンド・プロセス（切り離されたプロセス）内で開始されます。 バックグラウンド・プロセスは、キーボード、マウス、またはディスプレイに対する入力呼出しまたは出力呼出しを発行するものであっ

てはなりません。 プログラムが完全にデバッグされていて、対話方式によるユーザー入力がまったく必要でない場合は、このオプションを使用すると最大のパフォーマンスが得られます。

接続マネージャーがプログラムを開始できない場合（たとえば Communications Server が十分なメモリーを用意できない場合）は、接続マネージャーは着信割振り要求を拒否します。

トランザクション・プログラムが **RECEIVE_ALLOCATE** 呼出しを発行し、まだ定義されていないトランザクション・プログラム名を指定した場合は、システムはそのトランザクション・プログラムの暗黙定義を実行し、各パラメーターに省略時値を割り当てます。

使用される省略時値は次のとおりです。

Attach タイムアウト	= 0	（タイムアウトは適用されない）
受信割振りタイムアウト	= 0	（タイムアウトは適用されない）
接続マネージャーの動的ロード	= Yes	（トランザクション・プログラムは接続マネージャーによりロードできる）

上記で述べたような状況で **RECEIVE_ALLOCATE** 呼出しを発行し、これらの省略時値が適用された場合、指定したトランザクション・プログラムへの接続が試行されるか、またはユーザーがその呼出しを取り消すまで、呼出しは完了しません。

着信割振り要求と RECEIVE_ALLOCATE Verb との突合せ

ローカル・ワークステーションでリモート始動されたプログラムは、通常、APPC の **RECEIVE_ALLOCATE** verb を発行して、トランザクション・プログラムと会話の両方を開始します。APPC の **RECEIVE_ALLOCATE** verbは、リモート・トランザクション・プログラムが APPC の **[MC_]ALLOCATE** または **[MC_]SEND_CONVERSATION** verb に指定したものと同じトランザクション・プログラム名を指定します。APPC は、**RECEIVE_ALLOCATE** verb を接続マネージャーに渡して、処理を任せます。接続マネージャーは、**RECEIVE_ALLOCATE** verb が受信した Attach と一致していることを確認すると（さらに幾つかのクロスチェックを行った上で）、会話の始動が可能であることを APPC に知らせます。この時点で、接続マネージャーは会話への関与を打ち切ります。

トランザクション・プログラムの構成時に、同じプログラムに関する複数の着信割振り要求の取扱いについて、２つの選択項目のいずれかを指定できます。 つまり、同じプログラムの複数のインスタンスをローカル・ワークステーションで同時に実行するか（*非待ち行列* 操作）、または同じプログラムのインスタンスを一度に１つずつ実行することができます（*待ち行列* 操作）。 これらの値は、**queued** パラメーターおよび **dynamic load** パラメーターで構成できるものであり、これには次のようなオプションがあります。

- 非待ち行列－接続マネージャー始動
- 待ち行列－接続マネージャー始動
- 操作員始動

非待ち行列型プログラム

プログラムを非待ち行列型として構成してある場合は、着信割振り要求が送られてくるたびに、接続マネージャーは、着信トランザクション・プログラム名に対応するプログラムのそれぞれ異なるインスタンスをロードし実行します。

接続マネージャーは、有効な着信割振り要求を無期限に保持し、始動したプログラムがそれに一致する **RECEIVE_ALLOCATE** verb を発行するまで待ちます。そのプログラムが **RECEIVE_ALLOCATE** verb を発行できなかった場合（たとえば、**RECEIVE_ALLOCATE** verb より前でプログラムがループしている場合など）には、接続マネージャーは、プロセスが終了するまでその割振り要求を保留します。

待ち行列型プログラム

待ち行列型プログラムの始動には次の 2 通りの方法があります。

接続マネージャー始動
接続マネージャーがプログラムを始動します。

操作員始動
操作員またはワークステーション内の別のプログラムがプログラムを始動します。

接続マネージャーは、定義済みトランザクション・プログラム・リスト内の各待ち行列型トランザクション・プログラム名ごとに、それぞれ 2 つずつ待ち行列を維持します。 一方の待ち行列は着信割振り要求用で、もう一方は **RECEIVE_ALLOCATE** verb 用です。たとえば、着信割振り要求が 1 つ到着すると、接続マネージャーは、それに対応するローカル・プログラムを始動するか、または操作員にメッセージを送ります。 ノードは、接続マネージャーによって始動されたプログラムが一致する **RECEIVE_ALLOCATE** verb を発行するか、またはタイムアウトが発生するまで、着信割振り要求を保持します。ノードは、**incoming_alloc_timeout** パラメーターに構成されている値を使用して、タイムアウトの発生時点を決定します。その後は、同じトランザクション・プログラムまたは別のトランザクション・プログラムを対象とする新たな割振り要求を受け入れることができます。 他のプログラムは、一致する **RECEIVE_ALLOCATE** verb が発行されるか、またはタイムアウトが発生するまで、それぞれの待ち行列内で待機します。

ローカル・プログラムは、一致する割振り要求が到着する前に、**RECEIVE_ALLOCATE** verb を発行しておくことができます。接続マネージャーは、その **RECEIVE_ALLOCATE** verb を該当の待ち行列に入れ、パートナー LU からの割振り要求が到着するのを待ちます。各待ち行列ごとにタイムアウト値があります。**rcv_alloc_timeout** パラメーターに、タイムアウト前に **RECEIVE_ALLOCATE** verb が待ち行列内で待機できる時間の長さを指定できます。接続マネージャーは、待ち行列内の **RECEIVE_ALLOCATE** verb を、戻りコード ALLOCATE_NOT_PENDING とともに、関連のプログラムに戻します。**RECEIVE_ALLOCATE** verb のタイムアウト値を 0 にすると、プログラムは待ち行列内に割振り要求があるかどうかをチェックして、要求が 1 つもなければ他の処理を続けることができます。

RECEIVE_ALLOCATE verb は非ブロッキング verb として発行できます。したがって、トランザクション・プログラムは、単一プロセス内の単一スレッドから複数の会話にサービスを提供することができます。

RECEIVE_ALLOCATE を非ブロッキング verb として発行すると、接続マネージャーはトランザクション・プログラムにただちに制御を戻します。したがって、トランザクション・プログラムは、指定した着信割振り要求の到着を待つ間、待機状態のままになっている必要はありません。代わりに、トランザクション・プログラムは他の作業を行うことができ、指定した着信割振り要求をいつ待つのかを選択することができます。

トランザクション・プログラムは、それぞれ異なる会話に対して複数の非ブロッキング **RECEIVE_ALLOCATE** verb を、待ち行列に入れることができます。待ち行列に入れることのできる verb の最大数は、資源の制約によってのみ制限されます。非ブロッキング **RECEIVE_ALLOCATE** verb は、一致する割振り要求が到着するまで、または、verb のタイムアウトが生じる（つまり **rcv_alloc_timeout** の値に達する）まで、接続マネージャーの **RECEIVE_ALLOCATE** verb 待ち行列内に留まっています。

待ち行列内のプログラムが、トランザクション・プログラムにとって有効な **RECEIVE_ALLOCATE** verb 呼出しを発行すると、接続マネージャーはそのトランザクション・プログラムを識別する情報を保管します。待ち行列内のプログラムが終了すると、接続マネージャーは、そのトランザクション・プログラムに関する割振り要求の待ち行列を調べます。 待ち行列が空でなければ、接続マネージャーは、プログラムの新しいインスタンスを開始するか、または、操作員にプログラムの開始を求めるメッセージを送ります。

ユーザーは、各トランザクション・プログラムに対して、着信割振り要求の最大サイズを構成する必要があります。 待ち行列内の**RECEIVE_ALLOCATE** verb の最大数は、資源の制約により制限されます。

次の 2 つのケースは、待ち行列操作を要約して示しています。

ケース 1:

所定のトランザクション・プログラムについて **RECEIVE_ALLOCATE** verb または CPI 通信の CMACCP 呼出しが発行される前に、 1 つまたは複数の着信割振り要求が到着します。 接続マネージャーは、**RECEIVE_ALLOCATE** verb が発行されるまで（ただし、構成済みのタイムアウト値に指定されている時間の範囲内で）、着信割振り要求を待ち行列に入れておきます。 最初の着信割振り要求が**RECEIVE_ALLOCATE** verb を満たします。

ケース 2:

所定のトランザクション・プログラムについて着信割振り要求が到着する前に、**RECEIVE_ALLOCATE** verb が発行されます。 接続マネージャーは、着信割振り要求が到着するまで（ただし、構成済みのタイムアウト値に指定されている時間の範囲内で）、**RECEIVE_ALLOCATE** verb を待ち行列に入れておきます。 場合によっては、着信割振り要求が到着する前に、複数の **RECEIVE_ALLOCATE** verb が発行され、待ち行列に入れられることもあり

ます。 新しい着信割振り要求のそれぞれが、待ち行列内の次の **RECEIVE_ALLOCATE** verb を満たします。

31ページの表 7は、パラメーター値 **queued**および **dynamic load** に関連した verb および着信割振り要求の要約を示しています。

表 7. verb 処理とトランザクション・プログラム名構成

verb 処理	トランザクション・プログラム操作		
	非待ち行列 接続マネージャー始動	操作員始動	待ち行列 接続マネージャー始動
保留中の **RECEIVE_ALLOCATE** verb がある場合の着信割振り要求。	発生しません。保留中の **RECEIVE_ALLOCATE**verb の待ち行列がありません。	**RECEIVE_ALLOCATE** verb は処理されます。	**RECEIVE_ALLOCATE** verb は処理されます。
保留中の **RECEIVE_ALLOCATE** verb がない場合の着信割振り要求。	他のプログラム・インスタンスをロードし実行します。 着信割振り要求を保持します。 **RECEIVE_ALLOCATE** verb を待ちます。	待ち行列がいっぱいでない限り、着信割振り要求を待ち行列に入れます。 **RECEIVE_ALLOCATE** が発行されるか、または指定された時間が経過するまで待ちます。	プログラムが始動されていない場合は、そのプログラムをロードし実行します。 待ち行列がいっぱいでない限り、着信割振り要求を待ち行列に入れます。 **RECEIVE_ALLOCATE** が発行されるか、または指定された時間が経過するまで待ちます。
保留中の着信割振り要求がある場合の **RECEIVE_ALLOCATE** verb。	**RECEIVE_ALLOCATE** verb は処理されます。	**RECEIVE_ALLOCATE** verb は処理されます。	**RECEIVE_ALLOCATE** verb は処理されます。
保留中の着信割振り要求がない場合の **RECEIVE_ALLOCATE**verb。	発生しません。非待ち行列操作用の保留割振り要求は時間切れになることはありません。	**RECEIVE_ALLOCATE** verb を保留します。 着信割振り要求が到着するか、または指定された時間が経過するまで待ちます。	**RECEIVE_ALLOCATE** verb を保留します。 着信割振り要求が到着するか、または指定された時間が経過するまで待ちます。
トランザクション・プログラム操作の終了。	なにも起こりません。	なにも起こりません。	保留中の割振り要求がある場合は、プログラムを再ロードします。該当する割振り要求がない場合は、次回の着信割振り要求の到着時に再ロードします。

Communications Server SNA API クライアントでの接続マネージャーの使用

以下の節では、SNA API マシンに置かれているプログラムの始動方法を説明します。

SNA API クライアントのためのトランザクション・プログラムの定義

クライアント・マシンに置かれているトランザクション・プログラムをリモートに始動するには、Communications Server とクライアント・マシンの両方にトランザクション・プログラム定義が必要です。 サーバーで必要になるトランザクション・プログラムの情報には、次のものがあります。

- トランザクション・プログラム名
- 会話タイプ
- 同期レベル
- 会話セキュリティーが必要であるかどうかの別

Communications Server は、着信割振りが到着したときに、この情報をチェックします。さらに、着信割振り要求を受信するローカル LU は、その要求をクライアント・マシンに送れるようになっている必要があります。

クライアント接続マネージャーには、定義されたトランザクション・プログラムが必要です。これによって、要求されたプログラムを始動する方法がわかります。クライアントに必要なトランザクション・プログラム情報には、次のものがあります。

- トランザクション・プログラム名
- 着信割振り要求を受信するローカル LU
- プログラムのパス名
- トランザクション・プログラムに渡す必要のあるパラメーター

これらの定義が完全で、クライアント接続マネージャーが始動すると、クライアント・マシンに置かれたトランザクション・プログラム着信割振りは、クライアントに送られて、処理されます。

SNA API クライアント接続マネージャーの始動

クライアント接続マネージャーを始動するには、SNA クライアント用 Communications Server フォルダーにある接続マネージャー・アイコンをクリックします。 これによって、接続マネージャーは構成済みの Communications Server に接続され、そのクライアントに定義されているトランザクション定義のリストが送信されます。

第4章 トランザクション・プログラムの作成

この章では、APPC を使用するトランザクション・プログラムを計画し、作成する際に考慮するべき問題について説明します。 トランザクション・プログラムを作成するときは、いくつかの設計案の中から選択する必要があります。 以下に、考慮する必要のある設計上の問題をリストします。

- 基本会話とマップ式会話のいずれを選択するか
- 確認付きの会話を開始するか、確認なしの会話を開始するか
- セキュリティー機能の使用について
- ASCII 名とデータを変換する用意（必要な場合）

この章の前半では、アプリケーション・プロトコル、会話の状態、Communications Server サポート・タスク、およびデータ・フォーマットについての基礎知識を説明します。 この章の後半では、トランザクション・プログラムを作成するために必要な具体的な要件について述べます。

アプリケーション・プロトコル

Communications Server LU 6.2 を使用すると、プログラムとプログラムの間で通信を行うことができます。 ユーザー・プログラムの設計は、ユーザーが定義するプロトコル、およびそのプログラムが行いたい通信のタイプによって決まります。

ユーザーが自分のプログラムのために定義する規則の他に、Communications Server LU 6.2 は会話の使用時にユーザー・プログラムが守らなければならない規則を定義しています。 これらの規則を実施するため、Communications Server LU 6.2 は、ユーザーの会話の状態を管理し、会話が正しい状態にあるときだけユーザー・プログラムで特定の操作を行えるようにしています。 次に例を示します。

- ユーザー・プログラムは、送信の許可をもっていない場合は、データを送信できません。
- ユーザー・プログラムは、パートナー・プログラムが送信の許可をもっていない場合は、データを受け取れません。
- 会話の割振りが解除された後は、ユーザー・プログラムは会話を使用できません。

Communications Server LU 6.2 は33ページの表 8 に示してある会話状態を管理し、実施します。

表 8. 会話状態

状態	定義
リセット	会話は存在しない。
送信	プログラムは、データを送信し、確認を要求し、または会話を割振り解除することができる。
Receive	プログラムは、パートナー・プログラムからの情報を受け取ることができる。
Confirm	プログラムは確認要求に応答することができる。

利用可能な Communications Server LU 6.2 サービス

この節では、ユーザーのトランザクション・プログラムが他のトランザクション・プログラムと通信するために使用できる Communications Server LU 6.2 サービスについて説明します。

会話の割振り

ローカル LU に対して、パートナー LU 内のパートナー・トランザクション・プログラムと会話を開始するように要求します。

対応する verb: ALLOCATE および MC_ALLOCATE

データの送信

データをパートナー・プログラムに送信します。

対応する verb: SEND_DATA および MC_SEND_DATA

内部バッファーのデータを強制的に送信

LU に対して、それが内部バッファーに保持しているすべてのデータをパートナー・プログラムに送信するように強制します。

注: 通常は、ユーザーはこのサービスを使用して LU にデータの送信を強制する必要はありません。 LU は、内部バッファーが一杯になると、またはユーザー・プログラムによる送信が終了したと判断すると、内部バッファーに記憶していたデータを、自動的に送信します。

対応 する verb: FLUSH および MC_FLUSH

データの受信

データをパートナー・プログラムから受信します。

対応 verb: RECEIVE_AND_WAIT、RECEIVE_IMMEDIATE、
MC_RECEIVE_AND_WAIT および MC_RECEIVE_IMMEDIATE

送信許可の要求

データを送信する許可をパートナー・プログラムに要求します。

対応する verb: REQUEST_TO_SEND および MC_REQUEST_TO_SEND

送信許可の付与

パートナー・プログラムにデータ送信許可を与えます。

対応する verb: PREPARE_TO_RECEIVE および MC_PREPARE_TO_RECEIVE

要求確認

パートナー・プログラムに対して、すべてのデータが受信され、首尾よく処理されたことを確認するよう要求します。

対応する verb: CONFIRM および MC_CONFIRM

確認の受入れまたは拒否

確認要求に対する応答を送信します。

対応する verb: CONFIRMED、MC_CONFIRMED、SEND_ERROR および MC_SEND_ERROR

情報が利用可能であることを通知する要求

会話が受け取れる情報があるときは、LU がイベントを通知するように要求します。

対応する verb: RECEIVE_AND_POST

エラーの報告

エラーが発生したことを報告します。

対応する verb: SEND_ERROR および MC_SEND_ERROR

会話属性の入手

会話の属性を入手します。 この属性には次のものが入っています。

- ローカル LU の名前
- パートナー LU の名前
- セッションの伝送サービス・モードの名前
- 会話がサポートする確認プロトコルのタイプ
- 会話のタイプ

対応する verb: GET_ATTRIBUTES、MC_GET_ATTRIBUTES および GET_TYPE

会話の割振り解除

パートナー・プログラムとの会話を終了します。

対応する verb: DEALLOCATE および MC_DEALLOCATE

会話タイプの選択

この節では、基本会話かマップ式会話かの選択を行うときに考慮すべき問題を説明します。

会話タイプの一貫性

使用する会話タイプは、ALLOCATE verb で指示しますが、全会話にわたって一貫している必要があります。 いくつかの要求には基本会話 verb を使用し、他の要求には

マップ式会話を使用することはできません。 Communications Server LU 6.2 は、1 つの会話の中でタイプが一方から他方に変更されると、その verb を拒否します。 リモートに開始されたトランザクション・プログラムは、会話タイプを判別するために GET_TYPE verb を発行することができます。

基本会話には基本会話 verb だけが使用できます。マップ式会話を使用するプログラムでは、基本 verb またはマップ式 verb のいずれを使用することもできます。ただし、基本またはマップ式のどちらか一方のタイプの verb だけを使用する必要があります。

ユーザーは、マップ式と指示した会話に対して、基本会話 verb だけを使用した独自のマップ式会話サポートを準備することができます。 独自のマップ式会話サポートを提供することにした場合は、ユーザー・プログラムで、マップ式会話のフォーマットとプロトコルに準拠する必要があります。

マップ式会話のフォーマットとプロトコルについては、*SNA Format and Protocol Reference Manual: Architecture Logic for LU Type 6.2* および *Systems Network Architecture LU 6.2 Reference: Peer Protocols* を参照してください。

データの送信

論理レコードを複数含む、または部分論理レコードを含むバッファーからデータを送信することによって、プログラムのパフォーマンスを最適化する必要があるときは、基本会話を使用してください。 基本会話を使用すると、ユーザー・プログラムでは 1 つの要求によって複数の論理レコードを送信できるため、プログラムの実行効率を上げることができます。

基本会話を使用するためには、各論理装置レコードの初めに 2 バイトの*論理長フィールド（LL フィールド）*を置く必要があります。このフィールドについては、次のことがいえます。

- LL フィールドの最後の 15 ビットには、その論理レコードの長さに等しい 2 進値が入ります。この値には 2 バイトの長さフィールドも含まれます。 15 ビットという制約によって、この値の最大値は 32,767（32,765 バイトに長さフィールドの2 バイトを加えたもの）です。 32,767 よりも大きい値を使用すると、Communications Server LU 6.2 はそのエラーを検出できず、とにかく LL フィールドの最後の 15 ビットを使用してしまいます。

  指定できる最小の値は 2（データが続かない LL フィールド）です。 2 よりも小さい値を使用すると、Communications Server LU 6.2 はエラーであると指示します。
- Communications Server LU 6.2 は、LL フィールドの最初のビットは無視します。このビットは連結標識です。 連結標識がセットされているときは、トランザクション・プログラムは、続く論理レコードからのデータを、その時点までに受け取ったデータの後に付加する必要があります。 この連結プロセスは、トランザクション・プログラムが、連結標識がセットされていないレコードを受信するまで続行してください。 この定義によって、32,767 バイトより長い、より高いレベルのレコード（GDS 変数）を使用することができます。
- ユーザーは、使用する PC のバイト値の反転を処理する必要があります。

PC は、すべての数値について、低位 (有効性のより少ない) バイトを低位アドレスに記憶する方法で、16 または 32 ビット値を記憶します。 したがって、トランザクション・プログラムが論理メッセージの長さを計算し、その値を LL フィールドの値として格納すると、この 2 バイトはメモリーでは、低位バイトが最初に表れ、使用する PC はそのバイトをその順序（正しくない順序）で通信回線に送ります。

トランザクション・プログラムは、LL フィールドも含めて、トランザクション・レベルの全データについて正しい順序（高位バイトを最初に）で送信する責任があります。

部分論理レコードまたは複数の論理レコードを送信する必要がない場合は、マップ式会話を使用します。 マップ式会話 verb によってデータを送信するときは、Communications Server LU 6.2 は、バッファーには完全なより高いレベルのレコード（GDS 変数）が 1 つだけ入っているものと想定します。 マップ式会話サポートでは、長さフィールドは自動的に正しいバイト反転順序で提供され、また、必要に応じて連結論理レコードが使用されます。

データの受信

複数の論理レコードを 1 つのバッファーに受信する必要があるときは、基本会話を使用します。 このオプションを使用すると、ユーザー・プログラムでは複数のレコードを 1 つの要求で受信することができ（BUFFER オプション）、プログラムの実行効率を上げることができます。

基本会話機能では、Communications Server LU 6.2 は、2 バイトの LL フィールドには触らずに論理レコードをそのままバッファーに入れます。 バイトは、通常の IBM 互換の PC 順序とは逆になっています。

ユーザーは、verb の戻りフィールドを調べて、完全な論理レコードが受信されたか、受信された場合は、次の論理レコードがどこから始まるかを判別する必要があります。 Communications Server LU 6.2 は、後続のデータ受信要求が出されたときに、不完全な論理レコードの残りを提供します。

1 つの要求で、より高い/ユーザー・レベルのレコードを 1 つ受信したい場合は、マップ式会話を使用してください。 マップ式会話 verb によってデータを受信すると、プログラムが完全なより高い/ユーザー・レコードを受信するか、またはバッファーが一杯になると、Communications Server LU 6.2 は受信操作を終了します。 プログラムが完全な論理レコードを受信しない内にバッファーが一杯になると、Communications Server LU 6.2 は、戻りコードを戻します。

プログラムは、続けてデータ受信要求を発行して、より高い/ユーザー・レベル・レコードの残りを受け取ることができます。 Communications Server LU 6.2 マップ式会話サポートでは、長さフィールドがとり除かれ、必要に応じて自動的に論理レコードが連結されます。

エラーおよび異常終了の報告

以下の場合には、基本会話を使用してください。

- ユーザー・プログラムが検出したエラーと、ユーザー・プログラムを使用しているアプリケーションが検出したエラーを区別したい場合
- ユーザー・プログラムに起因する異常終了と、ユーザー・プログラムを使用するアプリケーションに起因する異常終了を区別したい場合

エラーを報告するとき、または LU サービス・プログラムとの会話を異常に終了するとき、基本会話 verb を使用すると、どちらのプログラムがエラーを検出したかを判別することができます。 パートナー LU がパートナー・プログラムに戻りコード付きでエラーを報告すると、戻りコードの値によって Communications Server LU 6.2 がそのエラーを検出した場所がわかります。

ユーザー・プログラムが検出したエラーと、他のアプリケーションが検出したエラーを区別する必要がない場合は、マップ式会話を使用してください。 マップ式会話 verb では、ユーザー・プログラムがエラーを検出したものとみなされます。

エラー・ログ・データ・レコードの送信

エラーを検出したとき、または会話を異常に終了したときにログ・レコードを送信するのであれば、基本会話を使用します。 基本会話 verb を使用すると、エラーを報告するとき、または会話を異常に終了するときにエラー・ログ GDS 変数を指定することができます。 Communications Server LU 6.2 は、ローカル・ログとパートナーLU にこのログ・レコードを送信し、そのログ内にこれを記録できるようにします。この機能は、ユーザー・プログラムが重大なエラーまたは回復不能なエラーを検出したとき、問題を判別するためにそのイベントをプログラムに記録させたい場合に有益です。

エラー・ログ GDS 変数を送信する場合は、そのレコードのフォーマットは SNA が定義しているフォーマットに一致している必要があります。 GDS 変数フォーマットについての詳細は、*IBM Systems Network Architecture Formats* を参照してください。

エラーを検出したり、会話を異常に終了するときに、ログ・レコードを送信する必要がないならば、マップ式会話を使用します。 マップ式会話 verb では、ユーザー・プログラムはエラー・データをログに記録して問題を判別する必要がないものとみなされます。

タイム・アウトに起因する異常終了

タイム・アウトが原因でユーザー・プログラムが会話を終了終了したことを示すには、基本会話を使用します。 会話を異常に終了するとき、基本会話 verb を使用すると、許容された時間内にパートナー・プログラムが必要な処理を行わなかったために、ユーザー・プログラムが異常に終了したことを、ユーザー・プログラムで示すことができます。 Communications Server LU 6.2 がこのエラーをパートナー・トランザクション・プログラムに報告するとき、戻りコード値は、この異常終了がタイム・アウトに起因するものであることを示しています。

異常終了の原因を報告する必要がない場合は、マップ式会話を使用します。 マップ式会話 verb では、重大なエラーまたは回復不能なエラーによってユーザー・プログラムが異常終了を要求したものとみなされます。

確認の要求

確認要求は、パートナー・プログラムがそれまでに送信されたすべてのデータを受信したかどうかを判別するための効率的な方法です。 会話中に確認を要求するつもりならば、割振りトランザクションは、会話に割振りを要求するときにそのことを示す必要があります。

確認を要求しない会話 verb を使用する場合は、確認サービスをサポートする会話の割振りは要求しないでください。

1 つのトランザクション・プログラムで、確認要求を使用する会話と確認要求を使用しない会話の両方を行うようにすることができます。

トランザクション・プログラム名の選択

トランザクション・プログラムに名前を付けるときは、その最初の文字が EBCDIC のブランク (X'40') よりも大きい EBCDIC コードをもつ名前を選んでください。 最初の文字が X'40' よりも小さい EBCDIC コードの文字になっているトランザクション・プログラム名は、サービス・トランザクション・プログラム用に確保されています。 トランザクション・プログラム名には 64 文字まで使用することができます。

セキュリティー機能の使用

Communications Server LU 6.2 は、パートナー LU 検証とエンド・ユーザー検証の2種類のセキュリティー機能の 1 つを提供します。パートナー LU 検証は、セッションが活動化したときに行われるセッション・レベルのセキュリティー・プロトコルです。 また、エンド・ユーザー検証は、会話の開始時に行われる会話レベルのセキュリティー・プロトコルです。

パートナー LU の検証（セッション・レベル・セキュリティー）

パートナー LU 検証は、2 つの LU の間のセキュリティー情報の交換によって行われます。 この交換は、セッション・レベル・セキュリティーと呼ばれます。 このセキュリティー・レベルは、一般に、通信ネットワークのセキュリティーが物理的に保たれない場合に必要になります。 ローカル LU とリモート LU は、それぞれパスワードを提供し、Communications Server LU 6.2 はパスワード検証のために暗号化を行います。必須ではありませんが、各 LU ペアは固有のパスワードを持つようにおすすめします。

エンド・ユーザーの検証（会話レベル・セキュリティー）

エンド・ユーザー検証を使用すると、要求されたアプリケーション・サブシステムは、要求されたトランザクション・プログラムとそのリソースにアクセスする前に、リクエスターの識別を検証することができます。 交換されるセキュリティー情報には、ユーザー ID とパスワードを入れることができます。 会話レベル・セキュリティーが提供するユーザー ID は、監査およびアカウンティングの目的にも使用することができます。

会話レベル・セキュリティーにおいては、要求側のトランザクション・プログラムが ALLOCATE verb にセキュリティー情報を提供し、リモート・アプリケーション・サブシステムが検証を行います。 要求側のトランザクション・プログラムが正しいユーザー ID とパスワードを提供しなかった場合は、リモート・アプリケーション・サブシステムはその要求を拒否します。

会話レベル・セキュリティーを要求する中間トランザクション・プログラム（他のトランザクション・プログラムによって開始されたトランザクション・プログラム）を使用して、会話レベル・セキュリティーを必要とする別のトランザクション・プログラムをアクセスすることができます。この場合、すでに検証済みの標識が、追加のトランザクション・プログラムのための割振り要求にセットされます。 中間トランザクション・プログラムを開始した、最初の要求で保管されたユーザー ID は、2 番目の要求に自動的に提供されます。

EBCDIC と ASCII の変換

Communications Server LU 6.2 では、それ自体とトランザクション・プログラム（またはアプリケーション・サブシステム）との間のインターフェースは、verb の指定があるところでは EBCDIC 文字を使用するものとみなされます。 ユーザー・プログラムは、データまたはパラメーターをシステム・サービス制御点 (SSCP) またはパートナー LU に送信することができ、そこでそのデータまたはパラメーターは EBCDIC で表現されているテーブル値と比較されます。 これらの値には、トランザクション・プログラム名、ALLOCATE に提供されているパートナー LU 名、モード名、ユーザー ID およびユーザー・パスワードが含まれます。

着信する名前は EBCDIC なので、ユーザー・プログラムではそれらを ASCII に変換する必要があります（アプリケーション・サブシステムがそれらを ASCII で格納するのであれば)。したがって、アプリケーション・サブシステムとトランザクション・プログラムの両方で、ASCII と EBCDIC の変換が行えるようにしておく必要があります。

2 つの PC 間でプライベート・プロトコルを定義することによって、この変換を避けてはなりません。 プライベート・プロトコルを使用すると、予期できない困難が生じたり、他のタイプのコンピューターとの通信ができなくなることがあります。たとえば、LU またはネットワーク名に ASCII 『k』 (X'4B') を使用すると、完全修飾名に使用される EBCDIC のピリオド (.) も値 X'4B' をもつため、プロトコル・エラーが起きたり、または名前が認識されないことになります。

LU 名、モード名、ネット名、およびパートナー LU 名は 8 文字の EBCDIC 文字とする必要があります（名前が 8 文字より短い場合は、名前の右側をブランクで埋め込みます）。これらの名前は、タイプ A の記号ストリングで、以下の任意の組合せから構成されます。ただし、その最初の文字は数字であってはなりません。

- 英字の大文字
- 数字
- 特殊文字の $、# および @

トランザクション・プログラムがデータを変換する必要があるかどうかは、パートナー・トランザクション・プログラム間のプライベートな取り決めによって決まります。 ユーザー・プログラムが、通常は EBCDIC を使用するノードと通信するのであれば、データを適切に EBCDIC に変換する必要があります。

便利なように、Communications Server LU 6.2 は CONVERT verb を提供しています。これは ASCII コードを EBCDIC に、または EBCDIC コードを ASCII に変換します。さらに詳しくは、275ページの『CONVERT』を参照してください。

第5章 APPC トランザクション・プログラムの実行

この章では、提供されているダイナミック・リンク・ライブラリー (DLL) ファイルを使用して、APPC トランザクション・プログラムを実行する方法について説明します。

Communications Server が提供する APPC は、Windows マシンの Microsoft** NT SNA Server とのバイナリー互換性を備えたものとして設計されており、OS/2 コミュニケーション・マネージャー/2 バージョン 1.0 の APPC インターフェースに似ています。

トランザクション・プログラムの作成

Communications Server は、APPC verb を処理するダイナミック・リンク・ライブラリー (DLL) ファイルを提供します。

DLL は再入可能であり、複数のアプリケーション・プロセスおよびスレッドで、同時に DLL を呼び出すことができます。

APPC verb は、簡単で分かりやすい言語インターフェースを備えています。ユーザー・プログラムは、*verb 制御ブロック (VCB)* と呼ばれるメモリー・ブロック内のフィールドに必要な値を入力します。 次にユーザー・プログラムは、APPC DLL を呼び出し、verb 制御ブロックを指すポインターを渡します。 APPC は、操作が完了すると、VCB のフィールドを使用し修正した後で、そのフィールドを戻します。これで、ユーザー・プログラムは、戻されたパラメーターを verb 制御ブロックから読み取ることができます。

43ページの表 9 は、APPC プログラムのコンパイルとリンクに必要な、提供されるヘッダー・ファイルとライブラリーのソース・モジュールの使用についてまとめたものです。 ヘッダー・ファイルの中には、他の必須ヘッダー・ファイルを含んでいるものがあります。

表 9. APPC のためのヘッダー・ファイルとライブラリー

オペレーティング・システム	ヘッダー・ファイル	ライブラリー	DLL 名
WINNT & WIN95	WINAPPC.H	WAPPC32.LIB	WAPPC32.DLL
WIN3.1	WINAPPC.H	WINAPPC.LIB	WINAPPC.DLL
OS/2	APPC_C.H	APPC.LIB	APPC.DLL

サポートされるオプション・セット

Communications Server は次の APPC オプション・セットをサポートします。 各オプション・セットの詳しい説明については、LU タイプ 6.2 のための *SNA Transaction Programmer's Reference*を参照してください。

101 LU 送信バッファーのフラッシュ

102 属性の取得

103 通知テストを伴う受信時の通知（**RECEIVE_AND_POST** verb によって使用できる機能）

104 待機を伴う受信時の通知（**RECEIVE_AND_POST** verb によって使用できる機能）

105 受信準備

106 即時受信

109 トランザクション・プログラム名およびインスタンス識別子の取得

110 会話タイプの取得

112 全二重会話および急送データ

オプション 112 は、サーバーにのっているアプリケーションに対してだけサポー トされます。

113 非ブロッキング・サポート

201 競合勝者セッションの待ち行列割振り

203 セッションの即時割振り

204 同じ LU にあるプログラム間の会話

205 待ち行列化割振りまたはセッション解放時

211 セッション・レベルの LU-LU 検査

212 ユーザー ID 検査

213 プログラム提供のユーザー ID およびパスワード

214 ユーザー ID 許可

241 PIP データ送信

242 PIP データ受信

243 アカウンティング

244 長期ロック

245 送信要求受信テスト

247 ユーザー制御データ

251 変換および会話相関子の抽出

290 システム・ログへのデータのロギング

291 マップ式会話 LU サービス構成要素

401 高信頼度片方向ブラケット

501 **CHANGE_SESSION_LIMIT** verb

502 **ACTIVATE_SESSION** verb

504 **DEACTIVATE_SESSION** verb

505 **LU-***definition* verb

601 **MIN_CONWINNERS_TARGET** パラメーター

602 **RESPONSIBLE(TARGET)** パラメーター

603 **DRAIN_TARGET(NO)** パラメーター

604 **FORCE** パラメーター

605 LU-LU セッション限度

606 ローカル認識の LU 名

607 非解釈 LU 名

610 最大 RU サイズ限度

612 競合勝者の自動活動化限度

613 ローカル最大 (LU、mode) セッション限度

616 CPSVCMG モード名サポート

全二重 VCB

これは、サーバーにのっているアプリケーションに対してだけサポートされます。

全二重会話に必要になるフォーマット 1 VCB の定義を明らかにするため、および急送データを送受信するため、トランザクション・プログラムは WINAPPC.H ファイルを組み込む前に WINAPPC_FORMAT_1 と呼ばれるコンパイラー定数を定義する必要があります。これは次のようにして、C 言語で行うことができます。

```
#define  WINAPPC_FORMAT_1
#include  <winappc.h>
```

この定数が定義されていない場合は、VCB のフォーマット 0 のバージョンのみを、アプリケーションからアクセスします。

待ち行列レベルの非ブロッキング

これは、サーバーにのっているアプリケーションに対してだけサポートされます。

Communications Server APPC API は待ち行列レベルの非ブロックキングをサポートしています。 このサポートは、APPC エントリー・ポイントを介して提供されます。

非ブロッキング操作では、verb の処理をすぐに完了できない場合にアプリケーションに制御を戻せるので、アプリケーションは、未完了の verb が完了したことを示す通知を受け取るまで、他の処理を続けることができます。 つまり、待ち行列レベルの非ブロッキングを使用すると、アプリケーションは複数の異なる待ち行列について非ブロッキング verb を発行でき、それらの verb を Communications Server に同時

に処理させることができるわけです。 また、アプリケーションは、特定の待ち行列に対して、前の verb の完了を待つことなく一連の非ブロッキング verb を発行することができます。

Communications Server は非ブロッキング verb 用として 6 つの待ち行列を維持しています。

- 割振り待ち行列（1 つの活動トランザクション・プログラムにつき 1 つずつ）
- 送信/受信待ち行列 (1 つの会話につき 1 つずつ、半二重のみ)
- 送信待ち行列 (1 つの全二重会話につき 1 つずつ）
- 受信待ち行列（1 つの全二重会話につき 1 つずつ）
- 急送待ち行列（1 つの会話につき 1 つずつ）
- 受信急送待ち行列（1 つの会話につき 1 つずつ）

6 つの待ち行列タイプはすべて、無制限数の verb を保留できます。 Communications Serverが他の（ブロッキングまたは非ブロッキング）verb を処理中の場合は、非ブロッキング verb は待ち行列に入れられます。 割振り待ち行列内の verb は並行して処理されますが、その他の待ち行列内の verb は、Communications Serverが受信した順序で、一度に 1 つずつ処理されます。

非ブロッキング・モードで verb を処理したい場合は、アプリケーションで **opext** フィールドに **AP_NON_BLOCKING** フラグをセットすることにより、それを Communications Server に通知します。 アプリケーションは、非同期の verb の完了をアプリケーションに通知するために使用されるイベント・ハンドルを、任意の非ブロッキング verb とともに与えることができます。
このハンドルは、**SECONDARY_RC**フィールドにセットして Communications Server に渡されます。 ハンドルを指定していない場合は、同じ待ち行列内の次の verb 完了のハンドルが指定された時点で、前の verb の完了がアプリケーションに通知されます。

同じ待ち行列上にあり、ハンドルを指定していない verb の完了後にイベントが通知された時点で、先行するハンドルなしのすべての verb の完了が保証されます。

非ブロッキング verb がフラグ **AP_OPERATION_INCOMPLETE_FLAG** を戻すと、それが **opext** フィールドにセットされます。

割振り待ち行列上に非ブロッキング・モードで発行できる APPC verb には、次のものがあります。

(MC_)ALLOCATE

(MC_)SEND_CONVERSATION

送受信待ち行列上に非ブロッキング・モードで発行できる APPC verb には、次のものがあります。

(MC_)CONFIRM

(MC_)CONFIRMED

(MC_)DEALLOCATE

(MC_)FLUSH

(MC_)PREPARE_TO_RECEIVE

(MC_)RECEIVE_AND_WAIT

(MC_)RECEIVE_IMMEDIATE

(MC_)SEND_DATA

(MC_)SEND_ERROR

送信待ち行列上に非ブロッキング・モードで発行できる (全二重会話の場合) APPC verb には、次のものがあります。

(MC_)DEALLOCATE

(MC_)FLUSH

(MC_)SEND_DATA

(MC_)SEND_ERROR

受信待ち行列上に非ブロッキング・モードで発行できる (全二重会話の場合) APPC verb には、次のものがあります。

(MC_)RECEIVE_AND_WAIT

(MC_)RECEIVE_IMMEDIATE

受信急送待ち行列上に非ブロッキング・モードで発行できる (全二重会話の場合) APPC verb には、次のものがあります。

(MC_)RECEIVE_EXPEDITED_DATA

送信急送待ち行列上に非ブロッキング・モードで発行できる APPC verbs には、次のものがあります。

(MC_)REQUEST_TO_SEND

(MC_)SEND_EXPEDITED_DATA

次の APPC verb は常に非同期に処理されますが、どの待ち行列にも関連付けられません。

(MC_)RECEIVE_AND_POST

(MC_)TEST_RTS_AND_POST

非ブロッキング・モードでは発行できない (そして、アプリケーションが非ブロッキング・フラグをセットしていてもブロッキング・モードで処理される) Communications Server APPC verb には、次のものがあります。

(MC_)GET_ATTRIBUTES

GET_TP_PROPERTIES

GET_TYPE

RECEIVE_ALLOCATE

TEST_RTS

TP_ENDED

TP_STARTED

CNOS

ALLOCATE verb または **RECEIVE_ALLOCATE** verb が正常に戻るまで（つまり Communications Server が AP_PARAMETER_CHECK および AP_BAD_CONV_ID を戻

すまで）、アプリケーションは、送受信待ち行列または急送待ち行列に対して verb を非ブロッキング・モードで発行することはできません。

送受信待ち行列または急送待ち行列に対して非ブロッキング verb が発行されたときに、同じ待ち行列上に現在未完了の状態の他の（ブロッキングまたは非ブロッキングの）verb がある場合は、新規発行の verb はその待ち行列に追加され、未完了のverb が完了した時点で処理されます。

他の verb（同じ会話用の）が未完了の状態にあるときに、ブロッキング verb を発行すると、Communications Server は、**primary_rc** として AP_TP_BUSY を戻し、そのverb を拒否します。 この点では **RECEIVE_AND_POST** はブロッキング verb として扱われますが、**TEST_RTS_AND_POST** は、同じ会話で他の verb が未完了状態にあっても発行することができる（そしてどの非ブロッキング verb 待ち行列にも入らない）という点に注意してください。同じ待ち行列に verb が存在しない場合に発行されたブロッキング verb は、他の待ち行列に verb が存在する場合でも正常として処理されます。 TEST_RTS、GET_ATTRIBUTES、GET_STATE、およびGET_TYPE は待ち行列と関連付けられず、いつでも実行される可能性があり、決して AP_TP_BUSY を戻さないことに注意してください。

省略時のローカル LU

Communications Server は、従属 LU 6.2 と独立 LU 6.2 の両方省略時のローカル LU をサポートします。 省略時 LU は、**lu_alias**フィールドをブランクにして **TP_STARTED** verb (88ページの『TP_STARTED』 を参照) を出すと使用されます。 独立 LU 6.2 の場合、省略時 LU は制御点 LU です。 従属 LU 6.2 の場合、ローカル LU プールが使用されます。 **DEFINE_LOCAL_LU** verb についての詳細は、*Communications Server: システム管理プログラミング*を参照してください。Communications Server は、次の規則に従って、省略時プールから LU を選択するか、または制御点 LU を使用します。

- 省略時のローカル LU プールのメンバーとして LU が構成されている場合は、Communications Server は、そのプールから使用されていない LU を選択します。プール内のすべての LU が使用中である場合は、**TP_STARTED** verb は失敗します。
- 省略時のローカル LU プールのメンバーとして構成されている LU がない場合は、Communications Server は制御点 LU を使用します。

第6章 CPI-C プログラムの実行

この章では、CPI-C インターフェースのための Communications Server サポートについて述べます。ここでは、主要な次の事項について説明します。

- CPI-C プログラムのコンパイルとリンクの手法
- CPI-C プログラムの作成と実行の方法
- Communications Server がサポートする CPI-C バージョンの機能

Communications Server が提供する CPIC は、Windows マシンの Microsoft** NT SNA Server とのバイナリー互換性を備えたものとして設計されており、OS/2 コミュニケーション・マネージャー/2 バージョン 1.0 の CPIC インターフェースに似ています。

CPIC プログラムの作成

Communications Server は、CPIC 呼出しを処理するダイナミック・リンク・ライブラリー (DLL) ファイルを提供します。

DLL は再入可能であり、複数のアプリケーション・プロセスおよびスレッドで、同時に DLL を呼び出すことができます。

49ページの表 10 は、CPIC プログラムのコンパイルとリンクに必要な、提供されるヘッダー・ファイルとライブラリーのソース・モジュールの使用についてまとめたものです。 ヘッダー・ファイルの中には、他の必須ヘッダー・ファイルを含んでいるものがあります。

表 10. CPIC のためのヘッダー・ファイルとライブラリー

オペレーティング・システム	ヘッダー・ファイル	ライブラリー	DLL 名
WINNT & WIN95	WINCPIC.H	WCPIC32.LIB	WCPIC32.DLL
WIN3.1	WINCPIC.H	WINCPIC.LIB	WINCPIC.DLL
OS/2	CPIC_C.H	CPIC16.LIB または CPIC32.LIB	CPIC.DLL

CPI-C のバージョン

CPI-C インターフェースには、これまでにいくつかのバージョン変更と拡張が加えられてきました。以下の 2 つの理由から、これらのバージョンについても知っておくことが必要です。

- 既存のプログラムを維持または移植する場合、どの機能呼出しが移植可能であるか、また、バージョンを変えるならばどのような変更が必要になるかを知っている必要があります。
- 新規プログラムを作成する場合、特定のバージョンに依存するコードを使用するときは旧バージョンについての知識が必要です。

Communications Server CPI-C 適合クラス・サポート

以下の CPI-C 2.1 適合クラスがサポートされています。これらの適合クラスは、IBM 資料の *Common Programming Interface Communications CPI-C Reference*バージョン2.1 (SC26-4399-08) に定義されています。

クライアント / サーバー・アプリケーションでサポートされるクラスの詳細については、*電話が鳴っている絵を参照してください。* 

会話適合クラスによって、プログラムは半二重会話を開始したり、終了したりすることができます。

スターター・セット呼出し:

CMACCP
Accept_Conversation

CMALLC
Allocate

CMDEAL
Deallocate

CMINIT
Initialize_Conversation

CMRCV
Receive

CMSEND
Send_Data

拡張機能呼出し:

CMCFM
Confirm

CMCFMD
Confirmed

CMECS
Extract_Conversation_State

CMECT
Extract_Conversation_Type

CMEMBS
Extract_Maximum_Buffer_Size

CMEMN
Extract_Mode_Name

CMESL
Extract_Sync_Level

CMFLUS
Flush

CMPTR
Prepare_To_Receive

CMRTS
Request_To_Send

CMSERR
Send_Error

CMSCT
Set_Conversation_Type

CMSDT
Set_Deallocate_Type

CMSF Set_Fill

CMSLD
Set_Log_Data

CMSMN
Set_Mode_Name

CMSPTR
Set_Prepare_To_Receive_Type

CMSRT
Set_Receive_Type

CMSRC
Set_Return_Control

CMSST
Set_Send_Type

CMSSL
Set_Sync_Level
必須 sync_level 値:
CM_NONE または CM_CONFIRM

CMSTPN
Set_TP_Name

CMTRTS
Test_Request_To_Send_Received

LU 6.2 適合クラスを使用すると、プログラムは以下のような 6.2 固有のサービスを使用することができます。

CMEPLN
Extract_Partner_LU_Name

CMSED
Set_Error_Direction

CMSPLN
Set_Partner_LU_Name

会話レベルの非ブロッキング適合クラスを使用すると、呼出しが即時に完了できない場合にプログラムは制御をとり戻すことができます。

CMCANC
Cancel_Conversation

CMSPM
Set_Processing_Mode

CMWAIT

Wait_For_Conversation

サーバー適合クラスでは、プログラムは CPI-C に複数のトランザクション・プログラム名を登録し、複数の着信会話を受け入れ、異なるクライアントのためのコンテキストを管理することができます。

CMACCI

Accept_Incoming

CMECTX

Extract_Conversation_Context

CMEXTX Extract_Conversation_Context は Windows 3.1 クライアントにはサポートされません。

CMETPN

Extract_TP_Name

CMRLTP

Release_Local_TP_Name

CMINIC

Initialize_For_Incoming

CMSLTP

Specify_Local_TP_Name

データ変換適合クラス・ルーチンを使用すると、文字ストリングの符号化をローカル・コードから EBCDIC へ、またはその逆に変更するためにローカル・ルーチンを呼ぶことができます。

CMCNVI

Convert_Incoming

CMCNVO

Convert_Outgoing

セキュリティー適合クラスを使用すると、サイド情報に入っている、またはプログラムが直接に設定したアクセス・セキュリティー情報を使用する会話を確立することができます。

CMESUI

Extract_Security_User_ID

CMSCSP

Set_Conversation_Security_Password

CMSCST

Set_Conversation_Security_Type
必須の conversation_security_type 値:
CM_SECURITY_NONE
CM_SECURITY_PROGRAM
CM_SECURITY_PROGRAM_STRONG
CM_SECURITY_SAME

CMSCSU

Set_Conversation_Security_User_ID

呼出しが完了できない場合に制御をとり戻すための**待ち行列レベルの非ブロッキング**。

CMCANC

Cancel_Conversation

CMSQPM

Set_Queue_Processing_Mode

CMWCMP

Wait_For_Completion

Queue-Level Non-Blocking は Windows 3.1 クライアントにはサポートされません。

呼出しを完了できない場合に制御をとり戻すための**コールバック機能**。

CMCANC

Cancel_Conversation

CMSQCF

Set_Queue_Callback_Function

コールバック機能は Windows 3.1 クライアントや OS/2 にはサポートされません。

2 次情報を使用すると、2 次エラー戻り情報を抽出することができます。

CMESI

Extract_Secondary_Information

2 次情報は Windows 3.1 クライアントにはサポートされません。

以下のクラスは、Communications Server でだけサポートされます。

全二重を使用すると、全二重会話にアクセスすることができます。

CMESRM

Extract_Send_Receive_Mode

CMSSRM

Set_Send_Receive_Mode

急送データは、急送データをパートナー・プログラムと交換します。

CMRCVX

Receive_Expedited_Data

CMSNDX

Send_Expedited_Data

以下の適合クラスはサポートされていません。

OSI TP サービス

回復可能トランザクション（資源回復インターフェース用）

非連鎖トランザクション（回復可能トランザクション用）

分散セキュリティー（分散セキュリティー・サーバーのユーザー・セキュリティー・サービス)

ディレクトリー（分散ディレクトリーに格納されているユーザー指示情報）

CPI-C 機能

54ページの表 11 には Communications Server がサポートするすべての CPI-C 機能をリストしてあります。古いプログラムを維持しているとき、または既存システムと互換性のあるプログラムを作成する必要があるときは、この表を参照してください。

表 11. Communications Server クライアントがサポートする CPI-C 機能

機能	長い名前	Windows 3.1 クライアント	Windows 95 および Windows NT クライアント	OS/2 クライアント	Windows NT Server
cmaccp	Accept_Conversation	x	x	x	x
cmacci	Accept_Incoming	x	x	x	x
cmallc	Allocate	x	x	x	x
cmcanc	Cancel_Conversation	x	x	x	x
cmcfm	Confirm	x	x	x	x
cmcfmd	Confirmed	x	x	x	x
cmcnvi	Convert_Incoming	x	x	x	x
cmcnvo	Convert_Outgoing	x	x	x	x
cmdeal	Deallocate	x	x	x	x
xcmdsi	Delete_CPIC_Side_Information	-	-	-	x
cmectx	Extract_Conversation_Context	-	x	x	x
xcecst	Extract_Conversation_Security_Type	x	x	x	-
cmecst	Extract_Conversation_Security_Type	x	x	x	x
cmecs	Extract_Conversation_State	x	x	x	x
cmect	Extract_Conversation_Type	x	x	x	x
xcmesi	Extract_CPIC_Side_Information	x	x	x	x
cmembs	Extract_Maximum_Buffer_Size	x	x	x	x
cmemn	Extract_Mode_Name	x	x	x	x
cmepln	Extract_Partner_LU_Name	x	x	x	x
cmesi	Extract_Secondary_Information	-	x	x	x
cmesui	Extract_Security_User_ID	x	x	x	x
cmecsu	Extract_Security_User_ID	x	x	x	x
xcecsu	Extract_Security_User_ID	x	x	x	x
cmesrm	Extract_Send_Receive_Mode	-	-	-	x
cmesl	Extract_Sync_Level	x	x	x	x
xceti	Extract_TP_ID	-	x	x	x
cmetpn	Extract_TP_Name	x	x	x	x
cmflus	Flush	x	x	x	x
cminit	Initialize_Conversation	x	x	x	x
xcinct	Initialize_Conversation_For_TP	-	x	x	x

表 11. Communications Server クライアントがサポートする CPI-C 機能 (続き)

機能	長い名前	Windows 3.1 クライアント	Windows 95 および Windows NT クライアント	OS/2 クライアント	Windows NT Server
cminic	Initialize_For_Incoming	x	x	x	x
cmptr	Prepare_To_Receive	x	x	x	x
cmrcv	Receive	x	x	x	x
cmrcvx	Receive_Expedited	-	-	-	x
cmrltp	Release_Local_TP_Name	x	x	x	x
cmrts	Request_To_Send	x	x	x	x
cmsend	Send_Data	x	x	x	x
cmsndx	Send_Expedited	-	-	-	x
cmserr	Send_Error	x	x	x	x
cmscsp	Set_Conversation_Security_Password	x	x	x	x
xcscsp	Set_Conversation_Security_Password	x	x	x	x
cmscst	Set_Conversation_Security_Type	x	x	x	x
xcscst	Set_Conversation_Security_Type	x	x	x	x
cmscsu	Set_Conversation_Security_User_ID	x	x	x	x
xcscsu	Set_Conversation_Security_User_ID	x	x	x	x
cmsct	Set_Conversation_Type	x	x	x	x
xcmssi	Set_CPIC_Side_Information	-	-	-	x
cmsdt	Set_Deallocate_Type	x	x	x	x
cmsed	Set_Error_Direction	sx	x	x	x
cmsf	Set_Fill	x	x	x	x
cmsld	Set_Log_Data	x	x	x	x
cmsmn	Set_Mode_Name	x	x	x	x
cmspln	Set_Partner_LU_Name	x	x	x	x
cmsptr	Set_Prepare_To_Receive_Type	x	x	x	x
cmspm	Set_Processing_Mode	x	x	x	x
cmsqcf	Set_Queue_Callback_Function	-	x	x	x
cmsqpm	Set-Queue_Processing_Mode	-	x	x	x
cmsrt	Set_Receive_Type	x	x	x	x
cmsrc	Set_Return_Control	x	x	x	x
cmssrm	Set_Send_Receive_Mode	-	-	-	x
cmsst	Set_Send_Type	x	x	x	x
cmssl	Set_Sync_Level	x	x	x	x
cmstpn	Set_TP_Name	x	x	x	x
cmsltp	Specify_Local_TP_Name	x	x	x	x
xchwnd*	Specify_Windows_Handle	x	x	-	x
xcstp	Start_TP	-	x	x	x
cmtrts	Test_Request_To_Send_Received	x	x	x	x
cmwcmp	Wait_For_Completion	-	x	x	x
cmwait	Wait_For_Conversation	x	x	x	x
xcendt	End_TP	-	x	x	x
WinCPICCleanup*		x	x	-	x
WinCPICIsBlocking*		x	-	-	-
WinCPICSetBlockingHook*		x	-	-	-
WinCPICStartup*		x	x	-	x
WinCPICUnhookBlockingHook*		x	-	-	-

***:** Microsoft Windows の WOSA 機能

表 11. Communications Server クライアントがサポートする CPI-C 機能 (続き)

機能	長い名前	Windows 3.1 クライアント	Windows 95 および Windows NT クライアント	OS/2 クライアント	Windows NT Server
x: サポートされる機能 **-:** サポートされない機能					

サービス TP 名の指定

この機能は、クライアントに対してだけサポートされます。

サービス・トランザクション・プログラムの名前を CMSTPN および CMSLTP 機能で指定するときは、特別な規則に従う必要があります。 通常は、CPI-C 機能によって標準の TP を指定します。 サービス・トランザクション・プログラムは、他のプログラムまたはユーザーに共通ネットワークおよびシステム・サービスを提供する、特殊化されたトランザクション・プログラムです。 サービス・トランザクション・プログラムの例としては、スケジューラー・プログラム、ディレクトリー・サービス、およびスプーラがあります。

CMSTPN および CMSL トランザクション・プログラム機能によってサービス・トランザクション・プログラムの名前を指定するための規則は、次のとおりです。

- 2 バイトから 5 バイトの ASCII 文字の名前を指定する。
- その名前の最初のバイト、たとえば 0x23 を ASCII 文字の 2 バイトに指定し直す。
  - 名前の最初のバイトを 2 つのニブル、たとえば、2 と 3 に分割し、各 ASCII バイトの低位ニブルにそれらを指定する。
  - 各 ASCII バイトの高位ニブルを 1 にセットする。これで、サービス TP の名前を指定していることが指示されます。 先の例では、最初の 2 バイトが 0x12 と 0x13 になります。
- ASCII文字による名前のこの後の3バイトまでの残りをゼロにします。たとえば、007とします。

したがって、0x23 007 というサービス・トランザクション・プログラムの名前は0x12 0x13 007 と指定されます。

第7章 APPC 用のエントリー・ポイント

この章では、APPC用のプロシージャー・エントリー・ポイントについて説明します。

APPC

これは、すべての APPC verb 用の同期エントリー・ポイントとして使用できます。また、このエントリー・ポイントを使用して非ブロッキング verb を発行することもできます。そのためには、2次戻りコード・フィールドにイベント・ハンドルを入れ、**opext** フィールド内で待ち行列レベルの非ブロッキング・フラグ (AP_NON_BLOCKING) をセットします。

構文

```
void WINAPI APPC(long)
```

入力パラメーターは verb 制御ブロックを指すポインターです。

戻り値

１次戻りコードおよび２次戻りコードを調べて、エラーの有無を確認してください。

使用上の注意

関連情報: 62ページの『WinAsyncAPPCEx( )』

これは、OS/2 プログラムにサポートされる唯一のエントリー・ポイントです。

WinAsyncAPPC( )

これは、すべての APPC verb 用の非同期エントリー・ポイントです。 アプリケーションは、Windows メッセージによる完了通知を選択する場合に、このエントリー・ポイントを使用します。 Communications Serverは、既存のアプリケーションとの互換性を確保するために、このエントリー・ポイントを提供しています。

構文

```
HANDLE WINAPI WinAsyncAPPC(HWND hWnd,
                           long vcb)
```

パラメーター

	説明
hwnd	完了メッセージを受け取るウィンドウ・ハンドル。
vcb	verb 制御ブロックへのポインター。

戻り値

戻り値は、非同期要求が正常に完了したかどうかを示します。要求が成功した場合は、実際の戻り値はハンドルです。 関数が成功しなかった場合は、Communications Server は 0を戻します。

使用上の注意

ブロック可能な APPC verb には次のものがあります。

- **[MC_]ALLOCATE**
- **[MC_]CONFIRM**
- **[MC_]CONFIRMED**
- **[MC_]DEALLOCATE**
- **[MC_]FLUSH**
- **[MC_]PREPARE_TO_RECEIVE**
- **RECEIVE_ALLOCATE**
- **[MC_]RECEIVE_AND_WAIT**
- **[MC_]REQUEST_TO_SEND**
- **[MC_]SEND_CONVERSATION**
- **[MC_]SEND DATA**
- **[MC_]SEND_ERROR**
- **TP_ENDED**
- **TP_STARTED**

WinAsyncAPPC エントリー・ポイントでは verb の取消しができますが、待ち行列レベルの非ブロッキングはサポートしていません。 APPC エントリー・ポイントは待ち行列レベルの非ブロッキングをサポートしていますが、アプリケーションが verb を取り消すことはできません。

このエントリー・ポイントは待ち行列レベルの非ブロッキングをサポートしていません。非同期インターフェースで待ち行列レベルの非ブロッキング・フラグ AP_NON_BLOCKING を指定してあっても、Communications Server はそれを無視します。 非同期エントリー・ポイントを使用しているときは、 1 つのアプリケーションで未完了状態のままにしておける関数は、 1 つの会話につき一度に 1 つだけです。このときに第 2 の関数を開始しようとすると、エラー・コード AP_CONV_BUSY が戻されます。アプリケーションが、イベント・ハンドルにより非同期完了の通知を受け取る必要がある場合は、**WinAsyncAPPCEx** エントリー・ポイントまたは **APPC** エントリー・ポイントのどちらでも使用できます。 ただし、**RECEIVE_AND_POST** および **RECEIVE_AND_WAIT** は例外です。 非同期サポートを十分に活用できるようにするために、Communications Serverは、非同期に発行された **RECEIVE_AND_WAIT** verbを、**RECEIVE_AND_POST** verb と同じ働きに変更します。 つまり、非同期の **RECEIVE_AND_POST** または **RECEIVE_AND_WAIT** が未完了状態にあるときに、アプリケーションは同じ会話で次のような verb を発行することができます。

- **REQUEST_TO_SEND**
- **GET_TYPE**
- **GET_ATTRIBUTES**
- **TEST_RTS**
- **DEALLOCATE** (AP_ABEND_PROG、AP_ABEND_SVC、または AP_ABEND_TIMER)
- **SEND_ERROR**
- **TP_ENDED**

結果として、サーバーなどのアプリケーションが、非同期の **RECEIVE_AND_WAIT** を使用してデータを受信できるようになります。 **RECEIVE_AND_POST** または **RECEIVE_AND_WAIT** が未完了の状態にあっても、アプリケーションは **SEND_ERROR** および **REQUEST_TO_SEND** を使用できます。

非同期操作が完了すると、アプリケーションのウィンドウ *hWnd* は、『**WinAsyncAPPC**』 を入力ストリングとする **RegisterWindowMessage** から戻されたメッセージを受け取ります。 *wParam* 引数には、元の関数呼出しから戻された非同期のタスク・ハンドルが入っています。 *lParam* 引数には元の VCB ポインターが入っており、これは最終戻りコードを判別するために使用できます。

WinAPPCCancelAsyncRequest を使用すると、アプリケーションは非同期 APPC アクションをどれでも取り消すことができますが、それに伴って関連する会話またはトランザクション・プログラムも終了します。 未完了の操作がある場合は、戻りコード AP_CANCELED を返します。

この関数が正常に終了すると、Communications Serverは、操作が完了したときまたは会話が取り消されたときに、**WinAsyncAPPC()** メッセージをアプリケーションに渡します。

関連情報:

62ページの『WinAsyncAPPCEx( )』

64ページの『WinAPPCCancelAsyncRequest()』

WinAsyncAPPCEx( )

これは、すべての APPC verb 用の非同期エントリー・ポイントです。 この呼出しは、同じスレッドで複数のセッションを処理できるようにするために使用します。

このエントリー・ポイントを使用するのは、イベントを介してアプリケーションに完了を通知する必要があり、アプリケーションで未完了の verb を取り消す機能が必要な場合です。それ以外の場合は、APPC 待ち行列レベルの非ブロッキング・エントリー・ポイントを使用してください。

構文

```
HANDLE WINAPI WinAsyncAPPCEx(HANDLE handle,
                             long vcb);
```

パラメーター

説明

handle
アプリケーションが待機するイベントのハンドル。

vcb verb 制御ブロックへのポインター。

戻り値

戻り値は、非同期要求が成功したがどうかを示します。この関数が成功した場合は、実際の戻り値はハンドルです。 関数が成功しなかった場合は、Communications Server は `0` を戻します。

使用上の注意

この verb は、Win32** API で **WaitForMultipleObjects** とともに使用するためのものです。

ブロック可能な APPC verb には次のものがあります。

- **[MC_]ALLOCATE**
- **[MC_]CONFIRM**
- **[MC_]CONFIRMED**
- **[MC_]DEALLOCATE**
- **[MC_]FLUSH**
- **[MC_]PREPARE_TO_RECEIVE**
- **RECEIVE_ALLOCATE**
- **[MC_]RECEIVE_AND_WAIT**
- **[MC_]REQUEST_TO_SEND**
- **[MC_]SEND_CONVERSATION**
- **[MC_]SEND_DATA**

- **[MC_]SEND_ERROR**
- **TP_ENDED**
- **TP_STARTED**

このエントリー・ポイントは待ち行列レベルの非ブロッキングをサポートしていません。 非同期インターフェースで待ち行列レベルの非ブロッキング・フラグ AP_NON_BLOCKING を指定してあっても、Communications Server はそれを無視します。 非同期エントリー・ポイントを使用しているときは、1 つのアプリケーションで未完了状態のままにしておける関数は、1 つの会話につき一度に 1 つだけです。 このときに第 2 の関数を開始しようとすると、エラー・コード AP_CONV_BUSY が戻されます。

WinAsyncAPPCEx エントリー・ポイントでは verb の取消しができますが、待ち行列レベルの非ブロッキングはサポートしていません。 **APPC** エントリー・ポイントは待ち行列レベルの非ブロッキングをサポートしていますが、アプリケーションが verb を取り消すことはできません。 ただし、**RECEIVE_AND_POST** および **RECEIVE_AND_WAIT** は例外です。 非同期サポートを十分に活用できるようにするために、Communications Serverは、非同期に発行された **RECEIVE_AND_WAIT** verb を、**RECEIVE_AND_POST** verb と同じ働きに変更します。 つまり、非同期の **RECEIVE_AND_POST** または **RECEIVE_AND_WAIT** が未完了状態にあるときに、アプリケーションは同じ会話で次のような verb を発行することができます。

- **REQUEST_TO_SEND**
- **GET_TYPE**
- **GET_ATTRIBUTES**
- **TEST_RTS**
- **DEALLOCATE** (AP_ABEND_PROG、AP_ABEND_SVC、または AP_ABEND_TIMER)
- **SEND_ERROR**
- **TP_ENDED**

結果として、特にサーバー・アプリケーションなどのようなアプリケーションが、非同期の **RECEIVE_AND_WAIT** を使用してデータを受信できるようになります。 **RECEIVE_AND_POST** または **RECEIVE_AND_WAIT** が未完了の状態にあっても、アプリケーションは **SEND_ERROR** および **REQUEST_TO_SEND** を使用できます。

非同期操作が完了すると、Communications Server は、イベントを通知することにより、アプリケーションに操作が完了したことを知らせます。 アプリケーションは、この通知を受け取ると、1 次戻りコードおよび 2 次戻りコードを調べて、エラー条件の有無を確認します。

関連情報:

59ページの『WinAsyncAPPC( )』

64ページの『WinAPPCCancelAsyncRequest()』

58ページの『APPC』

WinAPPCCancelAsyncRequest()

この関数は、未完了の **WinAsyncAPPC** に基づく要求を取り消します。

構文

```
int WINAPI WinAPPCCancelAsyncRequest(HANDLE handle);
```

パラメーター

説明

handle

指定パラメーター。取り消したい要求のハンドルを指定します。

戻り値

戻り値は、非同期要求が取り消されたかどうかを示します。値が 0 の場合は、Communications Server は要求を取り消しています。 取り消していない場合は、値は次に示すエラー・コードのいずれかになります。

WAPPCINVALID

指定した非同期タスク ID が無効でした。

WAPPCALREADY

取り消そうとしている非同期ルーチンはすでに完了しています。

使用上の注意

アプリケーション・プログラムは、**WinAPPCCancelAsyncRequest()** 呼出しを発行し、ハンドル内の初期関数から戻される非同期イベントを指定することにより、**WinAsyncAPPC** 関数の 1 つを使用して発行した非同期タスクを、それが完了する前に取り消すことができます。

未完了の verb が会話に関連する verb（たとえば **SEND_DATA** または **RECEIVE_AND_WAIT**）である場合は、Communications Server はその verb を除去し、セッションを非活動化します。 その verb がトランザクション・プログラムに関連する verb（たとえば **RECEIVE_ALLOCATE** または **TP_STARTED**）である場合は、Communications Serverはそのトランザクション・プログラムを終了します。 どちらの場合も、Communications Serverはできるだけ完全に会話とセッションを非活動化しますが、送信バッファー、確認待ち、またはその他の保留アクションをフラッシュすることはしません。 この呼出しは同期呼出しです。 上記で述べた処理が完了すると、Communications Server は、取り消した verb についての完了メッセージを通知します。

既存の非同期 **WinAsyncAPPC** ルーチンを取り消すのに失敗し、エラー・コード WAPPCALREADY が戻された場合は、そのルーチンはすでに完了しています。 アプリケーションは、結果の通知を処理済みであるか、または完了通知を処理していな

いかのどちらかです。 APPC 待ち行列レベルの非ブロッキング・エントリー・ポイントを介して発行された非同期 verb を取り消すことはできません。

関連情報: 59ページの『WinAsyncAPPC( )』

WinAPPCCancelBlockingCall( )

この関数は、この関数用のスレッドに関する未完了のブロッキング操作を取り消します。 Communications Serverは、未完了のブロックされた呼出しを取り消した場合、エラー・コード AP_CANCELLED を生成します。 この呼出しは、ブロッキング・フック関数の中でのみ使用するものです。 Communications Serverは、既存のアプリケーションとの逆方向の互換性を確保するために、この関数を提供しています。

構文

```
Int WINAPI WINAPPCCancelBlockingCall(void);
```

戻り値

戻り値は、取消し要求が成功したかどうかを示します。値が 0 の場合は、Communications Serverは要求を取り消しています。 取り消していない場合は、値は次に示すエラー・コードになります。

WAPPCINVALID
未完了のブロッキング呼出しはありません。

使用上の注意

未完了の verb が会話に関連する verb（たとえば **SEND_DATA** または **RECEIVE_AND_WAIT**）である場合は、Communications Server はその verb を除去し、セッションを非活動化します。 その verb がトランザクション・プログラムに関連する verb（たとえば **RECEIVE_ALLOCATE** または **TP_STARTED**）である場合は、Communications Serverはそのトランザクション・プログラムを終了します。 どちらの場合も、Communications Serverはできるだけ完全に会話とセッションを非活動化しますが、送信バッファー、確認待ち、またはその他の保留アクションをフラッシュすることはしません。 この呼出しは同期呼出しです。 上記に述べた処理が完了すると、関数は終了します。

マルチスレッド・アプリケーションは、複数のブロッキング操作を同時に未完了のままにしておくことができますが、ただしそれは１スレッドにつき１つに限ります。複数の未完了呼出しを個々に区別するために、**WinAPPCCancelBlockingCall()** は、現行の、もしくは呼び出すアプリケーション・スレッド上に未完了の操作があれば、それを取り消します。それがない場合は、この関数は失敗します。APPC は、未完了の操作がある間は、呼出し元のアプリケーション・スレッドを延期します。その結果、アプリケーションが **WinAPPCSetBlockingHook** を使用してスレッド用のブロッキング・フックを登録してある場合を除き、ブロッキング操作を開始したスレッドが制御を再獲得することはありません（したがって、**WinAPPCCancelBlockingCall()** に対する呼出しを発行することはできません）。

これは、Windows 95 および Windows NT SNA API クライアントではサポートされません。

WinAPPCCleanup( )

この関数は、アプリケーションを終了し、APPC API からアプリケーションの登録を取り消します。

構文

```
BOOL WINAPI WinAPPCCleanup(void);
```

戻り値

戻り値は、登録の取消しが成功したかどうかを示します。　値が 0 以外であれば、Communications Serverは正常にアプリケーションの登録を取り消しています。 アプリケーションの登録を取り消していない場合は、Communications Serverは値 `0` を戻します。

使用上の注意

WinAPPCCleanup() は、APPC API からCommunications Server・アプリケーションの登録を取り消すために使用します。

Communications Server は、まだ活動状態にある会話を終了し、トランザクション・プログラムを終了します。 この関数は、アプリケーションが所有しているすべてのトランザクション・プログラムについて **TP_ENDED(HARD)** を発行するのと同じです。

関連情報: 69ページの『WinAPPCStartup( )』

WinAPPCIsBlocking( )

この関数は、前のブロッキング要求の終了を待っている間にスレッドが実行中かどうかを判別します。 Communications Serverは、既存のアプリケーションとの逆方向の互換性を確保するために、この関数を提供しています。

構文

```
BOOL WlNAPI WinAPPCIsBlocking(void);
```

戻り値はこの関数の結果を示します。 値が 0 以外である場合は、未完了のブロッキング呼出しが完了を待っています。 値 0 は、未完了のブロッキング呼出しがないことを意味します。

使用上の注意

Communications Server DLL は、 1 スレッドにつき複数のブロッキング呼出しを禁止します。この状況が生じると、Communications Server DLL はAP_THREAD_BLOCKING を戻します。 ブロッキング呼出しを実行しているスレッドは、ブロッキング・フック関数が設定されている場合以外は再入しません。 この場合、 **WinAPPCIsBlocking** は、ブロッキング・フック関数内からのみ TRUE を戻します。

関連情報:

66ページの『WinAPPCCancelBlockingCall( )』

70ページの『WinAPPCSetBlockingHook( )』

72ページの『WinAPPCUnhookBlockingHook( )』

これは、Windows 95 および Windows NT SNA API クライアントではサポートされません。

WinAPPCStartup( )

この関数を使用すると、アプリケーションで、必要なCommunications Serverのバージョンを指定し、Communications Serverからバージョン情報を検索することができます。この呼出しは必須ではありません。

構文

```
int WINAPI WinAPPCStartup(WORD wVersionRequired,
                          LPWAPPCDATA wappcdata);
```

パラメーター

説明

wVersionRequired

必要なCommunications Serverのサポートのバージョンを指定します。 高位バイトはリリース番号（改訂番号）を示し、低位バイトはバージョン番号を示します。

wappcdata

APPC API のバージョンおよび API 実現の説明を戻します。

戻り値

戻り値は、Communications Serverが正常にアプリケーションを登録したかどうか、そして指定したバージョン番号をサポートするかどうかを示します。値が `0` であれば、Communications Serverは指定したバージョンをサポートしており、アプリケーションを正常に登録します。 その他の場合は、戻り値は次に示すうちのいずれかです。

WAPPCSYSNOTREADY

基盤のネットワーク・サブシステムが、ネットワーク通信用として作動可能な状態ではありません。

WAPPCVERNOTSUPPORTED

この特定のCommunications Server実施は、要求されたCommunications Serverのサポートのバージョンをサポートしません。

WAPPCINVALID

Communications Serverは、指定されたバージョンを判別できませんでした。

使用上の注意

WinAPPCStartup() は、API の将来のリリースの互換性を確保するためのものです。この DLL はバージョン J1.0 をサポートしています。

関連情報: 67ページの『WinAPPCCleanup( )』

WinAPPCSetBlockingHook( )

この関数を使用すると、APPC を実現する APPC API によって APPC 関数呼出しをブロックすることができます。

Communications Server は、既存のアプリケーションとの互換性を確保するために、この関数を提供しています。

構文

```
FARPROC WINAPI WinAPPCSetBlockingHook(FARPROC IpBlockFunc);
```

パラメーター

説明

IpBlockFunc

導入するブロッキング関数のプロシージャー・インスタンス・アドレスを指定します。

戻り値

戻り値は、すでに導入済みのブロッキング関数のプロシージャー・インスタンスを示します。 **SetBlockingHook** 関数を呼び出すアプリケーションまたはライブラリーは、この戻り値を保管し、必要があればこの値を復元できるようにしなければなりません （ネストが重要でない場合は、アプリケーションは、**WinAPPCSetBlockingHook()** から戻された値を破棄し、必要なときに **WinAPPCUnhookBlockingHook** を使用して省略時のメカニズムを復元することもできます)。

使用上の注意

ブロッキング関数は、WM_QUIT メッセージを受け取ると FALSE を戻します。したがって、Communications Serverは、制御をアプリケーションに戻し、アプリケーションがそのメッセージを処理して正常に終了できるようにすることができます。 その他の場合は、この関数は TRUE を戻します。

省略時値ではブロッキング・フックはありません。 アプリケーションがブロッキング・フックを設定しない限り、アプリケーション・スレッドは、ブロッキング呼出しからの戻りを無限に待ちます。

ブロッキング・フック関数が TRUE を戻さなかった場合は、Communications Server は、そのブロッキング verb を、 1次戻りコード AP_CANCELLED とともにアプリケーションに戻します。

この関数はスレッド単位で実行されます。 特定のスレッドについて、この関数によりブロッキング・メカニズムを置き換えても、他のスレッドが影響を受けることはありません。

関連情報:

66ページの『WinAPPCCancelBlockingCall( )』

68ページの『WinAPPCIsBlocking( )』

72ページの『WinAPPCUnhookBlockingHook( )』

これは、Windows 95 および Windows NT SNA API クライアントではサポートされません。

WinAPPCUnhookBlockingHook( )

この関数は、導入済みのブロッキング・フックがある場合に、それを削除します。

Communications Serverは、既存のアプリケーションとの逆方向の互換性を確保するために、この関数を提供しています。

構文

```
BOOL WINAPI WinAPPCUnhookBlockingHook (void);
```

戻り値

戻り値はこの関数の結果を示します。 Communications Serverが省略時のメカニズムを正常に再導入した場合は、戻り値は 0 以外の値です。 Communications Serverが省略時のメカニズムを再導入しなかった場合は、この値は 0 です。

使用上の注意

この関数が呼び出された後は、このアプリケーション・スレッドは、これ以降のすべてのブロッキング呼出しが完了するまで、無限に待機します。

関連情報: 70ページの『WinAPPCSetBlockingHook( )』

これは、Windows 95 および Windows NT SNA API クライアントではサポートされません。

GetAppcConfig( )

この関数は実現されていません。 しかし、逆方向の互換性の確保のためにエントリー・ポイントが提供されています。有効なパラメーターのセットを指定すると、Communications Server は APPC_CFG_SUCESS_NO_DEFAULT_REMOTE を戻し、RemLu バッファーの最初のバイトに NULL ターミネーターを入れます。

Communications Server は APPN ノードになることができるので、多くの場合、この呼出しは必要ありません。 パートナー LU の名前は ALLOCATE 上で指定でき、LU の探索が開始されます。 しかし、アプリケーションはノード操作員機能 (NOF) インターフェースを使用して、この情報を検索することができます。 インターフェースの使用方法については、*Communications Server: システム管理プログラミング*を参照してください。

GetAppcReturnCode( )

この関数は、VCB 内の 1 次戻りコードおよび 2 次戻りコードを印刷可能ストリングに変換します。 この関数は、APPC アプリケーションが使用するための標準のエラー・ストリングを提供しています。

構文

```
int WINAPI GetAppcReturnCode (struct appc_hdr *vcb,
                              UINT buffer_length,
                              unsigned char *buffer_addr);
```

パラメーター

説明

vcb 指定パラメーター。verb 制御ブロックのアドレスを指定します。

buffer_length

指定パラメーター。**buffer_addr** が指すバッファーの長さを指定します。この長さの推奨値は 256 です。

buffer_addr

指定パラメーター/戻りパラメーター。NULL 文字で終了する定型ストリングを入れるバッファーのアドレスを指定します。

戻り値

0x20000001

パラメーターが無効です。この関数は、指定した verb 制御ブロックからの読取り、または指定したバッファーへの書込みができませんでした。

0x20000002

指定したバッファーが小さすぎます。

使用上の注意

buffer_addr に戻されるエラー・ストリングは、改行文字 (**\n**) で終わりません。

第8章 APPC verb

この章では、APPC API を介して受け渡される各 verb の構文を示し、各 verb で使用されるパラメーターについて説明します。

また、APPC 全二重および半二重の会話に提供される APPC 基本会話およびマップ式会話 verb のための情報も載せてあります。 この章を読み進むとわかるように、基本 verb とマップ式 verb はよく似ています。そこで、両方を合わせて 1 つの章で説明しています。 しかし、違いもいくつかあります。 違いについては、下記のように示してあります。

- 基本 verb にだけ適用される情報には、この印が付けてあります。 
- マップ式 verb にだけ適用される情報には、この印が付けてあります。 

さらにはっきりさせるために、 会話 verb が基本にもマップ式にもなる場合は、次のように示してあります。

[MC_]VERBNAME

verb 制御ブロック

この項では、各 verb のフィールドおよび操作に関する一般事項について説明します。

共通フィールド

各 VCB には、次に示すような共通フィールドがあります。

opcode
verb 操作コード: verb の名前が入っている識別フィールド。

format
VCB のフォーマットを識別します。 VCB の現行バージョンを指定するためにこのフィールドをどの値に設定するかについては、それぞれの verb の項で個別に説明します。

primary_rc
1 次戻りコード。 各 verb に戻される可能性のある値を示します。

secondary_rc
2 次戻りコード。 これは、1 次戻りコードが提供する情報の補足情報です。各 verb に戻される可能性のある値を示します。 VCB によっては、上記に加えて次のフィールドも含まれることがあります。

opext verb 拡張コード。 これは verb 操作コードが提供する情報の補足情報です。verb シグナルを非ブロッキング・モードで処理する場合は、

AP_NON_BLOCKING フラグをセットする必要があります。 以下に説明するシグナルにはこれらの共通フィールドも含まれていますが、個別には説明してありません。

TP Identifiers

個々の活動トランザクション・プログラムに、それぞれ 8 バイトのトランザクション・プログラム識別子が割り当てられます。 この識別子は Communications Server によって割り当てられます。

トランザクション・プログラム識別子は、**TP_ENDED** verb の経路指定のため、そして会話 verb での相関係数として使用されます。

以下の節で、各シグナルの verb 制御ブロックについて説明します。

APPC API サポート

サポートされる verb

Communications Serverは、APPC API で以下の verb をサポートしています。

制御 verb

GET_TP_PROPERTIES
GET_TYPE
RECEIVE_ALLOCATE
TP_ENDED
TP_STARTED

会話 verb

[MC_]ALLOCATE
[MC_]CONFIRM
[MC_]CONFIRMED
[MC_]DEALLOCATE
[MC_]FLUSH
[MC_]GET_ATTRIBUTES
[MC_]PREPARE_TO_RECEIVE
[MC_]RECEIVE_AND_POST
[MC_]RECEIVE_AND_WAIT
[MC_]RECEIVE_EXPEDITED_DATA
[MC_]RECEIVE_IMMEDIATE
[MC_]REQUEST_TO_SEND
[MC_]SEND_CONVERSATION
[MC_]SEND_DATA
[MC_]SEND_ERROR

[MC_]SEND_EXPEDITED_DATA
[MC_]TEST_RTS
[MC_]TEST_RTS_AND_POST

制御 verb 制御ブロック

GET_TP_PROPERTIES

GET_TP_PROPERTIES は、トランザクション・プログラムに関連した属性を戻します。

VCB 構造

```
typedef struct get_tp_properties
{
 unsigned short    opcode;             /* verb operation code            */
 unsigned char     opext;              /* verb extension code            */
 unsigned char     format;             /* format                         */
 unsigned char     reserv2[2]          /* verb format                    */
 unsigned short    primary_rc;         /* primary return code            */
 unsigned long     secondary_rc;       /* secondary return code          */
 unsigned char     tp_id[8];           /* TP identifier                  */
 unsigned char     tp_name[64];        /* TP name                        */
 unsigned char     lu_alias[8];        /* LU alias                       */
 luw_id_overlay    luw_id;             /* LUW identifier                 */
 unsigned char     fqlu_name[17];      /* fully qualified LU name        */
 unsigned char     reserv3[10];        /* reserved                       */
 unsigned char     user_id[10];        /* user id                        */
} GET_TP_PROPERTIES;
typedef struct luw_id_overlay{
 unsigned char     fqlu_name_len;      /* fully qualified LU name length */
 unsigned char     fqlu_name[17];      /* fully qualified LU name        */
 unsigned char     instance[6];        /* instance number                */
 unsigned char     sequence[2];        /* sequence number                */
} LUW_ID_OVERLAY;
```

指定パラメーター

トランザクション・プログラムは、次のパラメーターを Communications Server に提供します。

opcode

AP_GET_TP_PROPERTIES

tp_id ローカル・トランザクション・プログラムの識別子。 このパラメーターの値は、ローカルで始動したトランザクション・プログラムでは **TP_STARTED** verb から戻された値であり、リモートから始動されたトランザクション・プログラムでは **RECEIVE_ALLOCATE** verb から戻された値です。

opext AP_BASIC_CONVERSATION

format

VCB のフォーマットを識別します。 上記に示した VCB のバージョンを指定するには、このフィールドに 1 をセットしてください。

戻りパラメーター

verb が正常に実行された場合は、Communications Serverは次のパラメーターを戻します。

primary_rc

AP_OK

tp_name

ローカル・トランザクション・プログラム（つまりこの verb を発行するトランザクション・プログラム）の名前。 Communications Serverはこのフィールドの文字セットをチェックしません。

lu_alias

トランザクション・プログラムに関連付けられているローカル LU の別名。これは、8バイトの ASCII 文字ストリングです。 8 バイトのすべてが意味を持つため、すべてのバイトをセットする必要があります。

luw_id フィールドは、プロテクトされていない会話（**sync_level** が AP_NONE または AP_CONFIRM_SYNC_LEVEL である会話）に関連付けられている作業論理単位識別子です。 **luw_id_overlay** には次のパラメーターが含まれています。

luw_id_overlay.fq_lu_name_len

作業論理単位に関連付けられている完全修飾 LU 名の長さ。

luw_id_overlay.fq_lu_name

作業論理単位に関連付けられている完全修飾 LU 名。この名前は長さが 17 バイトで、17 バイトに満たない場合は右側に EBCDIC のブランクが埋め込まれます。この名前は、EBCDIC　　のピリオドで連結された２つのタイプA EBCDIC 文字ストリングです（１つの名前の長さは最大８バイトで、ブランクを含んでいてはなりません。ネットワーク ID がない場合は、ピリオドを省略してください)。名前の長さが 17 バイトに満たないときは、名前の後にすぐに **instance** および **sequence** が続きます（これは、LUW_ID_OVERLAY 構造のフィールドを使用して、**instance** または **sequence** のどちらにもアクセスできないことを意味します)。

luw_id_overlay.instance

作業論理単位インスタンス番号。 これは、長さが６バイトの２進ストリングです。

luw_id_overlay.sequence

作業論理単位順序番号。 これは、長さが２バイトの２進ストリングです。

luw_id_overlay.fq_lu_name_len が 17 バイトに満たないときは、**luw_id_overlay** の右側（**instance** および **sequence** の後）に EBCDIC のブランクが埋め込まれます。

luw_id_overlay.fq_lu_name

トランザクション・プログラムに関連付けられているローカル LU の完全修飾名。 この名前は長さが 17 バイトで、17 バイトに満たない場合は右側に EBCDIC のブランクが埋め込まれます。この名前は、EBCDIC のピリオドで連結された２つのタイプA EBCDIC 文字ストリングです （１つの名前の長さは最大８バイトで、ブランクを含んでいてはなりません。ネットワークIDがない場合は、ピリオドを省略してください)。

user_id

トランザクションの開始プログラムのユーザー ID。 これは 10 バイトのタ

イプ AE EBCDIC 文字ストリングで、10 バイトに満たない場合は右側に EBCDIC のスペースが埋め込まれます。

パラメーター・エラーが原因で verb が実行されなかった場合は、Communications Serverは次のパラメーターを戻します。

primary_rc

AP_PARAMETER_CHECK

secondary_rc

AP_BAD_TP_ID

次に示す 1 次戻りコード (**primary_rc**) が生成される条件については、付録 A. APPC 共通戻りコードに説明してあります。

AP_TP_BUSY

AP_UNEXPECTED_SYSTEM_ERROR

GET_TYPE

GET_TYPE verb は、特定の会話の会話タイプ（基本またはマップ式）を戻します。

VCB 構造

```
typedef struct get_type
{
 unsigned short    opcode;             /* verb operation code     */
 unsigned char     opext;              /* verb extension code     */
 unsigned char     format;             /* format                  */
 unsigned short    primary_rc;         /* primary return code     */
 unsigned long     secondary_rc;       /* secondary return code   */
 unsigned char     tp_id[8];           /* TP identifier           */
 unsigned long     conv_id;            /* conversation identifier */
 unsigned char     conv_type;          /* conversation type       */
 unsigned char     conv_style;         /* conversation style      */
} GET_TYPE;
```

指定パラメーター

トランザクション・プログラムは、次のパラメーターを Communications Server に提供します。

opcode

AP_GET_TYPE

opext AP_BASIC_CONVERSATION

format

VCB のフォーマットを識別します。 上記に示した VCB のバージョンを指定するには、このフィールドに 1 をセットしてください。

tp_id ローカル・トランザクション・プログラムの識別子。 このパラメーターの値は、ローカルで始動したトランザクション・プログラムでは **TP_STARTED** verb から戻された値であり、リモートから始動されたトランザクション・プログラムでは **RECEIVE_ALLOCATE** verb から戻された値です。

conv_id

会話識別子。 このパラメーターの値は、ローカルで始動したトランザクション・プログラムでは **ALLOCATE** verb から戻された値であり、リモートから始動されたトランザクション・プログラムでは **RECEIVE_ALLOCATE** verb から戻された値です。

戻りパラメーター

verb が正常に実行された場合は、Communications Serverは次のパラメーターを戻します。

primary_rc

AP_OK

conv_type

conv_id で識別される会話の会話タイプ。

AP_BASIC_CONVERSATIONAP_MAPPED_CONVERSATION

conv_style

conv_id で識別される会話の会話スタイル。 このフィールドには、フォーマット 1 バージョンの VCB が必要です。 フォーマット 1 VCB のアクセスの詳細は、 45ページの『全二重 VCB』を参照してください。

AP_HALF_DUPLEXAP_FULL_DUPLEX

パラメーター・エラーが原因で verb が実行されなかった場合は、Communications Serverは次のパラメーターを戻します。

primary_rc

AP_PARAMETER_CHECK

secondary_rc

AP_BAD_TP_ID

AP_BAD_CONV_ID

次に示す 1 次戻りコード (**primary_rc**) が生成される条件については、付録 A. APPC 共通戻りコードに説明してあります。

AP_TP_BUSY

AP_UNEXPECTED_SYSTEM_ERROR

RECEIVE_ALLOCATE

RECEIVE_ALLOCATE verb は、**ALLOCATE** verb または **MC_ALLOCATE** verb を発行したパートナー・トランザクション・プログラムとの会話のために、新しいトランザクション・プログラムを確立するのに必要な情報を要求します。

VCB 構造

```
typedef struct receive_allocate
{
 unsigned shor    opcode;              /* verb operation code     */
 unsigned char    opext;               /* verb extension code     */
 unsigned char    format;              /* format                  */
 unsigned shor    primary_rc;          /* primary return code     */
 unsigned long    secondary_rc;        /* secondary return code   */
 unsigned char    tp_name[64];         /* TP name                 */
 unsigned char    tp_id[8];            /* TP identifier           */
 unsigned long    conv_id;             /* conversation identifier */
 unsigned char    sync_level;          /* sync Level              */
 unsigned char    conv_type;           /* conversation type       */
 unsigned char    user_id[10];         /* user ID                 */
 unsigned char    lu_alias[8];         /* LU alias                */
 unsigned char    plu_alias[8];        /* partner LU alias        */
 unsigned char    mode_name[8];        /* mode name               */
 unsigned char    reserv3[2];          /* reserved                */
 unsigned long    conv_group_id;       /* conversation group ID   */
 unsigned char    fqplu_name[17];      /* fully qualified partner LU name */
 unsigned char    pip_incoming;        /* received PIP data       */
 unsigned char    conversation_style;  /* conversation style      */
 unsigned char    reserv4[3];          /* reserved                */
 unsigned char    password[10];        /* security password       */
 unsigned char    reserv5[2];          /* reserved                */
 unsigned char    dload_id[8];         /* user ID                 */
} RECEIVE_ALLOCATE;
```

指定パラメーター

トランザクション・プログラムは、次のパラメーターを Communications Server に提供します。

opcode

AP_RECEIVE_ALLOCATE

opext AP_BASIC_CONVERSATION

format

VCB のフォーマットを識別します。 上記に示した VCB のバージョンを指定するには、このフィールドに 0 をセットしてください。

tp_name

呼び出されたトランザクション・プログラムの名前。Communications Serverはこのフィールドの文字セットをチェックしません。

戻りパラメーター

verb が正常に実行された場合は、Communications Serverは次のパラメーターを戻します。

primary_rc
AP_OK

tp_id ローカル・トランザクション・プログラムの識別子。 この値は、Communications Serverがトランザクション・プログラムに割り当てます。トランザクション・プログラムは、後続のすべての APPC verb で、この識別子をCommunications Serverへ渡します。

conv_id
会話識別子。 この値によって、2 つのトランザクション・プログラム間に確立される会話が識別されます。

sync_level
会話の同期レベル。

AP_CONFIRM_SYNC_LEVEL

AP_NONE

conv_type
conv_id で識別される会話の会話タイプ。

AP_BASIC_CONVERSATIONAP_MAPPED_CONVERSATION

user_id
パートナー・トランザクション・プログラムから提供されるユーザー ID。これは 10 バイトのタイプ AE EBCDIC 文字ストリングで、10 バイトに満たない場合は右側に EBCDIC のスペースが埋め込まれます。

lu_alias
ローカル LU を認識させるための別名。 これは、8 バイトの ASCII 文字ストリングです。 8 バイトのすべてが意味を持つため、すべてのバイトをセットする必要があります。

plu_alias
ローカル・トランザクション・プログラムにパートナー LU を認識させるための別名。これは、8 バイトの ASCII 文字ストリングです。 8 バイトのすべてが意味を持つため、すべてのバイトをセットする必要があります。

mode_name
構成時に定義したネットワーキング特性のセットの名前。これは、8 バイトの英数字のタイプ A EBCDIC ストリング（英字で始まるもの）で、8 バイトに満たない場合は右側に EBCDIC のスペースが埋め込まれます。

conv_group_id
この会話で使用しているセッションの会話グループ識別子。

fqplu_name

パートナー LU の完全修飾 LU 名。この名前は長さが 17 バイトで、17 バイトに満たない場合は右側に EBCDIC のブランクが埋め込まれます。 この名前は、
EBCDIC のピリオドで連結された 2 つのタイプA EBCDIC 文字ストリングです （1 つの名前の長さは最大 8 バイトで、ブランクを含んでいてはなりません。 ネットワーク ID がない場合は、ピリオドを省略してください)。

pip_incoming

パートナー・トランザクション・プログラムが、[MC_]ALLOCATE 要求でプログラム初期設定パラメーター (PIP) を提供するかどうかを指定します。AP_YES または AP_NO にセットします。AP_YES の場合は、この会話で最初に発行される **[MC_]RECEIVE_*** verb で PIP データが受信されます。

conv_style

conv_id で識別される会話の会話スタイル。

AP_HALF_DUPLEXAP_FULL_DUPLEX

パスワード (password)

user_id に関連付けられているパスワード。 これは 10 バイトのタイプ AE EBCDIC 文字ストリングで、10 バイトに満たない場合は右側に EBCDIC のスペースが埋め込まれます。 これは Security = Program (AP_PGM または AP_PGM_STRONG) の場合は必須で、その他の場合は任意選択です。

パラメーター・エラーが原因で verb が実行されなかった場合は、Communications Serverは次のパラメーターを戻します。

primary_rc

AP_PARAMETER_CHECK

secondary_rc

AP_UNDEFINED_TP_NAME

次に示す 1 次戻りコード (**primary_rc**) が生成される条件については、付録 A. APPC 共通戻りコードに説明してあります。

AP_UNEXPECTED_SYSTEM_ERROR

TP_ENDED

TP_ENDED verb は、指定したトランザクション・プログラムが終了したことを Communications Server に通知します。

VCB 構造

```
typedef struct tp_ended
{
 unsigned short    opcode;             /* verb operation code     */
 unsigned char     opext;              /* verb extension code     */
 unsigned char     format;             /* format                  */
 unsigned short    primary_rc;         /* primary return code     */
 unsigned long     secondary_rc;       /* secondary return code   */
 unsigned char     tp_id[8];           /* TP identifier           */
 unsigned char     type;               /* type of TP ended        */
} TP_ENDED;
```

指定パラメーター

トランザクション・プログラムは、次のパラメーターを Communications Server に提供します。

opcode

AP_TP_ENDED

opext AP_BASIC_CONVERSATION

format

VCB のフォーマットを識別します。 上記に示した VCB のバージョンを指定するには、このフィールドに 0 をセットしてください。

tp_id ローカル・トランザクション・プログラムの識別子。 このパラメーターの値は、ローカルで始動したトランザクション・プログラムでは **TP_STARTED** verb から戻された値であり、リモートから始動されたトランザクション・プログラムでは **RECEIVE_ALLOCATE** verb から戻された値です。

type **TP_ENDED** のタイプ。

AP_HARDAP_SOFTAP_ABENDAP_CANCEL

タイプが AP_ABEND の場合は、Communications Server は**TP_ENDED** 信号に応答しません。

戻りパラメーター

verb が正常に実行された場合は、Communications Server は次のパラメーターを戻します。

primary_rc

AP_OK

戻りパラメーター

パラメーター・エラーが原因で verb が実行されなかった場合は、Communications Serverは次のパラメーターを戻します。

primary_rc

AP_PARAMETER_CHECK

secondary_rc

AP_BAD_TP_ID

AP_BAD_TYPE

次に示す 1 次戻りコード (**primary_rc**) が生成される条件については、付録 A. APPC 共通戻りコードに説明してあります。

AP_TP_BUSY

AP_UNEXPECTED_SYSTEM_ERROR

TP_STARTED

TP_STARTED verb は、着信割振り要求ではなくローカル・コマンドの結果として始動されたトランザクション・プログラムに対して、プログラムが資源を要求していることをCommunications Serverに通知します。

VCB 構造

```
typedef struct tp_started
{
 unsigned short    opcode;             /* verb operation          */
 unsigned char     opext;              /* verb extension          */
 unsigned char     format;             /* format                  */
 unsigned short    primary_rc;         /* primary return code     */
 unsigned long     secondary_rc;       /* secondary return code   */
 unsigned char     lu_alias[8];        /* LU alias                */
 unsigned char     tp_id[8];           /* TP identifier           */
 unsigned char     tp_name[64];        /* TP name                 */
} TP_STARTED;
```

指定パラメーター

トランザクション・プログラムは、次のパラメーターを Communications Server に提供します。

opcode
AP_TP_STARTED

opext AP_BASIC_CONVERSATION

format
VCB のフォーマットを識別します。 上記に示した VCB のバージョンを指定するには、このフィールドに 0 をセットしてください。

lu_alias
ローカル LU を認識させるための別名。 このパラメーターを 0 にセットした場合は、Communications Serverは制御点 LU を使用します。 これは、8 バイトの ASCII 文字ストリングです。 8 バイトのすべてが意味を持つため、すべてのバイトをセットする必要があります。 ブランクの **lu_alias** フィールドも受け入れられます。 その場合は制御点 LU が使用されます。

tp_name
呼び出されたトランザクション・プログラムの名前。Communications Serverはこのフィールドの文字セットをチェックしません。

戻りパラメーター

verb が正常に実行された場合は、Communications Serverは次のパラメーターを戻します。

primary_rc
AP_OK

tp_id ローカル・トランザクション・プログラムの識別子。 この値は、Communications Server がトランザクション・プログラムに割り当てます。トランザクション・プログラムは、後続のすべての APPC verb で、この識別子をCommunications Serverへ渡します。

パラメーター・エラーが原因で verb が実行されなかった場合は、Communications Serverは次のパラメーターを戻します。

primary_rc

AP_PARAMETER_CHECK

secondary_rc

AP_INVALID_LU_NAME

AP_INVALID_ENABLE_POOL

次に示す 1 次戻りコード (**primary_rc**) が生成される条件については、付録 A. APPC 共通戻りコードに説明してあります。

AP_UNEXPECTED_SYSTEM_ERROR

会話 verb

[MC_]ALLOCATE

[MC_]ALLOCATE verb は、ローカルのトランザクション・プログラムが発行します。この verb は、ローカル LU とパートナー LU との間のセッションを割り振り、（**RECEIVE_ALLOCATE** verb とともに機能して）ローカルのトランザクション・プログラムとリモートのトランザクション・プログラムとの間の会話を確立します。

ALLOCATE verb は、基本会話またはマップ式会話のいずれかを確立できます。**ALLOCATE** verb を使用してマップ式会話を確立すると、トランザクション・プログラムは、基本会話 verb を使用して、マップ式会話パートナー・トランザクション・プログラムと通信することができます。

Communications Serverは、この verb が正常に実行されると、識別子(**conv_id**) を生成します。 この識別子は、すべての他の APPC 会話 verb に必要なパラメーターです。

VCB 構造:

```
typedef struct allocate
{
 unsigned short    opcode;             /* verb operation code     */
 unsigned char     opext;              /* verb extension code     */
 unsigned char     format;             /* format                  */
 unsigned short    primary_rc;         /* primary return code     */
 unsigned long     secondary_rc;       /* secondary return code   */
 unsigned char     tp_id[8];           /* TP identifier           */
 unsigned long     conv_id;            /* conversation identifier */
 unsigned char     conv_type;          /* conversation type       */
 unsigned char     sync_level;         /* sync level              */
 unsigned char     reserv3[2];         /* reserved                */
 unsigned char     rtn_ctl;            /* return control                */
 unsigned char     conversation_style; /* conversation style            */
 unsigned long     conv_group_id;      /* conversation group identifier */
 unsigned long     sense_data;         /* sense data                    */
 unsigned char     plu_alias[8];       /* partner LU alias         */
 unsigned char     mode_name[8];       /* mode name                */
 unsigned char     tp_name[64];        /* partner TP name          */
 unsigned char     security;           /* security level           */
 unsigned char     reserv5[11];        /* reserved                 */
 unsigned char     pwd[10];            /* security password        */
 unsigned char     user_id[10];        /* security user_id         */
 unsigned short    pip_dlen;           /* PIP data length          */
 unsigned char    *pip_dptr;           /* pointer to PIP data      */
 unsigned char     reserv5a;           /* reserved                 */
 unsigned char     fqplu_name[17];     /* fully qualified partner LU    */
                                       /* name                     */
 unsigned char    reserv6[8];          /* reserved                 */
} ALLOCATE;

typedef struct mc_allocate
{
 unsigned short    opcode;             /* verb operation code     */
 unsigned char     opext;              /* verb extension code     */
 unsigned char     format;             /* format                  */
```

```
 unsigned short    primary_rc;         /* primary return code     */
 unsigned long     secondary_rc;       /* secondary return code   */
 unsigned char     tp_id[8];           /* TP identifier           */
 unsigned long     conv_id;            /* conversation identifier */
 unsigned char     reserv3;            /* reserved                */
 unsigned char     sync_level;         /* sync level              */
 unsigned char     reserv4[2];         /* reserved                */
 unsigned char     rtn_ctl;            /* return control          */
 unsigned char     conversation_style; /* conversation style      */
 unsigned long     conv_group_id;      /* conversation group identifier */
 unsigned long     sense_data;         /* sense data              */
 unsigned char     plu_alias[8];       /* partner LU alias        */
 unsigned char     mode_name[8];       /* mode name               */
 unsigned char     tp_name[64];        /* partner TP name         */
 unsigned char     security;           /* security level          */
 unsigned char     reserv6[11];        /* reserved                */
 unsigned char     pwd[10];            /* security password       */
 unsigned char     user_id[10];        /* security user_id        */
 unsigned short    pip_dlen;           /* PIP data length         */
 unsigned char    *pip_dptr;           /* pointer to PIP data     */
 unsigned char     reserv6a;           /* reserved                */
 unsigned char     fqplu_name[17];     /* fully qualified partner LU */
                                       /* name                    */
 unsigned char     reserv7[8];         /* reserved                */
} MC_ALLOCATE;
```

指定パラメーター: トランザクション・プログラムは、次のパラメーターを Communications Server に提供します。

opcode

AP_B_ALLOCATE 

AP_M_ALLOCATE 

format

VCB のフォーマットを識別します。 上記に示した VCB のバージョンを指定するには、このフィールドに 0 をセットしてください。

opext AP_BASIC_CONVERSATION または AP_MAPPED_CONVERSATION。非ブロッキング操作の場合は、このフラグと AP_NON_BLOCKING を OR で結ぶことができます。

tp_id ローカル・トランザクション・プログラムの識別子。

このパラメーターの値は、ローカルで始動したトランザクション・プログラムでは **TP_STARTED** verb から戻された値であり、リモートから始動されたトランザクション・プログラムでは **RECEIVE_ALLOCATE** verb から戻された値です。

conv_type 

割り振る会話のタイプ。

AP_BASIC_CONVERSATION
AP_MAPPED_CONVERSATION

ALLOCATE verb がマップ式会話を確立する場合は、ローカル・トランザクション・プログラムは基本会話 verb を発行できますが、その場合データ・レコードを論理レコードに、そして論理レコードをデータ・レコードに変換する独自のマッピング層を提供する必要があります。 パートナー・トランザクション・プログラムは、マッピング層を提供することによって基本会話 verb を発行することができ、また、マップ式会話 verb を使用することもできます（パートナー・トランザクション・プログラムが使用している APPC の機能で、マップ式会話 verb がサポートされている場合)。 詳細は、*IBM Systems Network Architecture: LU 6.2 Reference: Peer Protocols* を参照してください。

sync_level
会話の同期レベル。

AP_CONFIRM_SYNC_LEVEL
AP_NONE

rtn_ctl
ローカル・トランザクション・プログラムからのセッション要求を処理するローカル LU が、ローカル・トランザクション・プログラムに、いつ制御を戻すかを指定します。

AP_IMMEDIATE
AP_WHEN_SESSION_ALLOCATED
AP_WHEN_SESSION_FREE
AP_WHEN_CONV_GROUP_ALLOC
AP_WHEN_CONWINNER_ALLOC
AP_WHEN_CONLOSER_ALLOC

conversation_style
conv_id で識別される会話の会話スタイル。

duplex_type
割り振られる会話の、二重のタイプ。

AP_HALF_DUPLEXAP_FULL_DUPLEX

これは、サーバーにのっているアプリケーションに対してだけサポートされます。

conv_group_id

割り振られるセッションの会話グループ識別子。 このパラメーターが提供されるのは、**rtn_ctl** を AP_WHEN_CONV_GROUP_ALLOC にセットした場合だけです。

plu_alias

ローカル・トランザクション・プログラムにパートナー LU を認識させるための別名。これは、 8 バイトの ASCII 文字ストリングです。 8 バイトのすべてが意味を持つため、すべてのバイトをセットする必要があります。 この名前は、構成時に確立されたパートナー LU の名前に一致している必要があります。 このフィールドをすべて 0 にセットした場合は、Communications Server は **fqplu_name** フィールドを使用して、必要なパートナー LU を指定します。

mode_name

構成時に定義されるネットワーキング特性の名前。これは、8 バイトの英数字のタイプ A EBCDIC ストリング（英字で始まるもの）で、8 バイトに満たない場合は右側に EBCDIC のスペースが埋め込まれます。

tp_name

呼び出されたトランザクション・プログラムの名前。Communications Serverはこのフィールドの文字セットをチェックしません。 ローカルのトランザクション・プログラムで **ALLOCATE** verb により指定されている **tp_name** の値は、パートナーのトランザクション・プログラムで **RECEIVE_ALLOCATE** verb により指定されている **tp_name** の値に一致している必要があります。

security

パートナー・トランザクション・プログラムへのアクセスの妥当性をチェックするために、パートナー LU が必要とする情報を指定します。

AP_NONE

呼び出されたトランザクション・プログラムは会話セキュリティーを使用しません。

AP_PGM

呼び出されたトランザクション・プログラムは会話セキュリティーを使用します。このセキュリティーにはユーザー ID とパスワードが必要です。

AP_SAME

呼び出されたトランザクション・プログラムは、会話セキュリティーを使用し、チェック済み標識を受け入れるように構成されます。 ユーザー ID は、チェック済み標識と共に送られ、パスワードが必要でないことを、呼び出されたプログラムに通知します。

AP_PGM_STRONG

AP_PGM と同じですが、パートナー LU へのセッションがパスワード置換をサポートしているときのみ、ALLOCATE は正常に実行されます。

注: [MC_]ALLOCATE が AP_SAME のセキュリティー・タイプを指定しているが、ユーザー ID とパスワードを指定していない場合は、前の SET_TP_PROPERTIES verb （もしあれば）で指定されたユーザー ID とパスワードが使用されます。 [MC_]ALLOCATE にユーザー ID とパスワードがある場合、これらは常に SET_TP_PROPERTIES verb に指定されたものに代わって使用されます。

pwd **user_id** に関連付けられているパスワード。 これは 10 バイトのタイプ AE EBCDIC 文字ストリングで、10 バイトに満たない場合は右側に EBCDIC のスペースが埋め込まれます。 これは Security = Program (AP_PGM または AP_PGM_STRONG) の場合は必須で、その他の場合は任意選択です。

user_id

パートナー・トランザクション・プログラムにアクセスするために必要なユーザー ID。 これは 10 バイトのタイプ AE EBCDIC 文字ストリングで、10 バイトに満たない場合は右側に EBCDIC のスペースが埋め込まれます。 これは Security = Program (AP_PGM または AP_PGM_STRONG) の場合は必須で、その他の場合は任意選択です。

pip_dlen

パートナー・トランザクション・プログラムに渡すプログラム初期設定パラメーター (PIP) の長さ。 有効範囲は 0 から 32767 です。

pip_dptr

PIP データが入っているバッファーのアドレス。 このパラメーターは、**pip_dlen** が 0 より大きい場合にのみ使用します。

fqplu_name

パートナー LU の完全修飾 LU 名。 この名前は長さが 17 バイトで、17 バイトに満たない場合は右側に EBCDIC のブランクが埋め込まれます。 この名前は、EBCDIC のピリオドで連結された 2 つのタイプA EBCDIC 文字ストリングです （1 つの名前の長さは最大 8 バイトで、ブランクを含んでいてはなりません。 ネットワーク ID がない場合は、ピリオドを省略してください)。 このフィールドが意味を持つのは、**plu_alias** フィールドをすべて0 にセットした場合だけです。

戻りパラメーター: verb が正常に実行された場合は、Communications Serverは次のパラメーターを戻します。

primary_rc

AP_OK

conv_id

会話識別子。 この値によって、2 つのトランザクション・プログラム間に確立される会話が識別されます。

conv_group_id
会話に割り振られるセッションの会話グループ識別子。

verb が非ブロッキングで完了していない場合は、Communications Server は次のパラメーターを戻します。

primary_rc
AP_OPERATION_INCOMPLETE

opext AP_OPERATION_INCOMPLETE_FLAG

rtn_ctl パラメーターが AP_IMMEDIATE にセットされているときに、すぐに使用できるセッションがない場合は、Communications Server は次のパラメーターを戻します。

primary_rc
AP_UNSUCCESSFUL

パラメーター・エラーが原因で verb が実行されなかった場合は、Communications Serverは次のパラメーターを戻します。

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_BAD_CONV_TYPE 

AP_BAD_DUPLEX_TYPE
AP_BAD_RETURN_CONTROL
AP_BAD_SECURITY
AP_BAD_SYNC_LEVEL
AP_CONFIRM_INVALID_FOR_FDX
AP_NO_USE_OF_SNASVCMG_CPSVCMG 

AP_BAD_TP_ID
AP_PIP_LEN_INCORRECT
AP_UNKNOWN_PARTNER_MODE

sense_data
[MC_]ALLOCATE が失敗した理由に関する追加情報を提供します。

次に示す 1 次戻りコード (**primary_rc**) およびそれに付随する 2 次戻りコード (**secondary_rc**) が生成される条件については、付録 A. APPC 共通戻りコードで説明します。

AP_ALLOCATION_ERROR
AP_ALLOCATION_FAILURE_NO_RETRY

AP_ALLOCATION_FAILURE_RETRY

AP_FDX_NOT_SUPPORTED_BY_LU

AP_SEC_REQUESTED_NOT_SUPPORTED

AP_TP_BUSY
AP_UNSUCCESSFUL
AP_UNEXPECTED_SYSTEM_ERROR
AP_CANCELLED

primary_rc が AP_ALLOCATION_ERROR の場合は、**sense_data** フィールドには障害に関するより詳細な情報が含まれています。

[MC_]CONFIRM

CONFIRM verb は、ローカル LU 送信バッファーのデータ、および確認要求を、パートナー・トランザクション・プログラムに送ります。 **CONFIRM** verb に対する応答として、パートナー・トランザクション・プログラムは、通常、エラーなしでデータを受け取ったことを確認するために **CONFIRMED** verb を発行します （パートナー・トランザクション・プログラムがエラーを発見した場合は、**SEND_ERROR** verb を発行するか、または異常処置として会話の割振りを解除します)。

トランザクション・プログラムが **CONFIRM** verb を発行できるのは、会話の同期レベル（**ALLOCATE** verb により確立されたもの）が AP_CONFIRM_SYNC_LEVEL である場合だけです。

VCB 構造:

```
typedef struct confirm
{
 unsigned short    opcode;             /* verb operation code      */
 unsigned char     opext;              /* verb extension code      */
 unsigned char     format;             /* format                   */
 unsigned short    primary_rc;         /* primary return code      */
 unsigned long     secondary_rc;       /* secondary return code    */
 unsigned char     tp_id[8];           /* TP identifier            */
 unsigned long     conv_id;            /* conversation identifier  */
 unsigned char     rts_rcvd;           /* request to send received */
#ifdef WINAPPC_FORMAT_1
 unsigned char     expd_data_rcvd;     /* expedited data received  */
#endif
} CONFIRM;

typedef struct mc_confirm
{
 unsigned short    opcode;             /* verb operation code      */
 unsigned char     opext;              /* verb extension code      */
 unsigned char     format;             /* format                   */
 unsigned short    primary_rc;         /* primary return code      */
 unsigned long     secondary_rc;       /* secondary return code    */
 unsigned char     tp_id[8];           /* TP identifier            */
 unsigned long     conv_id;            /* conversation identifier  */
 unsigned char     rts_rcvd;           /* request to send received */
#ifdef WINAPPC_FORMAT_1
 unsigned char     expd_data_rcvd;     /* expedited data received  */
#endif
} MC_CONFIRM;
```

指定パラメーター: トランザクション・プログラムは、次のパラメーターを Communications Server に提供します。

opcode

AP_B_CONFIRM 

AP_M_CONFIRM

opext AP_BASIC_CONVERSATION または AP_MAPPED_CONVERSATION。非ブロッキング操作の場合は、このフラグと AP_NON_BLOCKING を OR で結ぶことができます。

format
VCB のフォーマットを識別します。 上記に示した VCB のバージョンを指定するには、このフィールドに 1 をセットしてください。

tp_id ローカル・トランザクション・プログラムの識別子。 このパラメーターの値は、ローカルで始動したトランザクション・プログラムでは **TP_STARTED** verb から戻された値であり、リモートから始動されたトランザクション・プログラムでは **RECEIVE_ALLOCATE** verb から戻された値です。

conv_id
会話識別子。 このパラメーターの値は、ローカルで始動したトランザクション・プログラムでは **ALLOCATE** verb から戻された値であり、リモートから始動されたトランザクション・プログラムでは **RECEIVE_ALLOCATE** verb から戻された値です。

戻りパラメーター: verb が正常に実行された場合は、Communications Serverは次のパラメーターを戻します。

primary_rc
AP_OK

rts_rcvd
送信要求受信の標識。

AP_YESAP_NO

expd_data_rcvd
急送データ受信の標識。 この標識は、RECEIVE_EXPEDITED_DATA が発行されるまで、AP_YES にセットされたままです。

AP_YESAP_NO

このフィールドには、フォーマット 1 バージョンの VCB が必要です。 フォーマット 1 VCB のアクセスの詳細は、45ページの『全二重 VCB』を参照してください。

verb が非ブロッキングで完了していない場合は、Communications Serverは次のパラメーターを戻します。

primary_rc
AP_OPERATION_INCOMPLETE

opext verb が非ブロッキングで完了していない場合は、Communications Server は次のパラメーターを戻します。

AP_OPERATION_INCOMPLETE_FLAG

パラメーター・エラーが原因で verb が実行されなかった場合は、Communications Serverは次のパラメーターを戻します。

primary_rc

AP_PARAMETER_CHECK

secondary_rc

AP_BAD_CONV_ID

AP_BAD_TP_ID
AP_CONFIRM_INVALID_FOR_FDX
AP_CONFIRM_ON_SYNC_LEVEL_NONE

トランザクション・プログラムがこの verb を発行したときに会話が不適切な状態にあった場合は、Communications Serverは次のパラメーターを戻します。

primary_rc

AP_STATE_CHECK

secondary_rc

AP_CONFIRM_BAD_STATE

AP_CONFIRM_NOT_LL_BDY 

次に示す 1 次戻りコード (**primary_rc**) およびそれに付随する 2 次戻りコード (**secondary_rc**) が生成される条件については、付録 A. APPC 共通戻りコードで説明します。

AP_ALLOCATION_ERROR

AP_SECURITY_NOT_VALID
AP_TRANS_PGM_NOT_AVAIL_RETRY
AP_TRANS_PGM_NOT_AVAIL_NO_RTRY
AP_TP_NAME_NOT_RECOGNIZED
AP_PIP_NOT_ALLOWED
AP_PIP_NOT_SPECIFIED_CORRECTLY
AP_CONVERSATION_TYPE_MISMATCH
AP_SYNC_LEVEL_NOT_SUPPORTED
AP_CONV_FAILURE_NO_RETRY
AP_CONV_FAILURE_RETRY
AP_DEALLOC_ABEND

AP_DEALLOC_ABEND_PROG 

AP_DEALLOC_ABEND_TIMER 

AP_PROG_ERROR_PURGING

AP_SVC_ERROR_PURGING 

AP_CONVERSATION_TYPE_MIXED 

AP_UNEXPECTED_SYSTEM_ERROR 

AP_TP_BUSY

AP_CANCELLED

[MC_]CONFIRMED

CONFIRMED verb は、パートナー・トランザクション・プログラムからの確認要求に応答します。 この verb は、ローカル・トランザクション・プログラムが受信したデータにエラーを検出しなかったことを、パートナー・トランザクション・プログラムに通知します。

確認要求を発行したトランザクション・プログラムは確認応答を待つため、**CONFIRMED** verb により 2 つのトランザクション・プログラムの処理を同期化することができます。

VCB 構造:

```
typedef struct confirmed
{
 unsigned short    opcode;             /* verb operation code     */
 unsigned char     opext;              /* verb extension code     */
 unsigned char     format;             /* format                  */
 unsigned short    primary_rc;         /* primary return code     */
 unsigned long     secondary_rc;       /* secondary return code   */
 unsigned char     tp_id[8];           /* TP identifier           */
 unsigned long     conv_id;            /* conversation identifier */
} CONFIRMED;

typedef struct mc_confirmed
{
 unsigned short    opcode;             /* verb operation code     */
 unsigned char     opext;              /* verb extension code     */
 unsigned char     format;             /* format                  */
 unsigned short    primary_rc;         /* primary return code     */
 unsigned long     secondary_rc;       /* secondary return code   */
 unsigned char     tp_id[8];           /* TP identifier           */
 unsigned long     conv_id;            /* conversation identifier */
} MC_CONFIRMED;
```

指定パラメーター: トランザクション・プログラムは、次のパラメーターを Communications Server に提供します。

opcode

AP_B_CONFIRMED 

AP_M_CONFIRMED 

opext AP_BASIC_CONVERSATION または AP_MAPPED_CONVERSATION。非ブロッキング操作の場合は、このフラグと AP_NON_BLOCKING を OR で結ぶことができます。

format

VCB のフォーマットを識別します。 上記に示した VCB のバージョンを指定するには、このフィールドに 0 をセットしてください。

tp_id ローカル・トランザクション・プログラムの識別子。 このパラメーターの値は、ローカルで始動したトランザクション・プログラムでは **TP_STARTED** verb から戻された値であり、リモートから始動されたトランザクション・プログラムでは **RECEIVE_ALLOCATE** verb から戻された値です。

conv_id
会話識別子。 このパラメーターの値は、ローカルで始動したトランザクション・プログラムでは **ALLOCATE** verb から戻された値であり、リモートから始動されたトランザクション・プログラムでは **RECEIVE_ALLOCATE** verb から戻された値です。

戻りパラメーター: verb が正常に実行された場合は、Communications Server は次のパラメーターを戻します。

primary_rc
AP_OK

verb が非ブロッキングで完了していない場合は、Communications Server は次のパラメーターを戻します。

primary_rc
AP_OPERATION_INCOMPLETE

opext AP_OPERATION_INCOMPLETE_FLAG

パラメーター・エラーが原因で verb が実行されなかった場合は、Communications Serverは次のパラメーターを戻します。

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_BAD_CONV_ID

AP_BAD_TP_IDAP_CONFIRMED_INVALID_FOR_FDX

トランザクション・プロセッサーがこの verb を発行したときに会話が不適切な状態にあった場合は、Communications Serverは次のパラメーターを戻します。

primary_rc
AP_STATE_CHECK

secondary_rc
AP_CONFIRMED_BAD_STATE

次に示す 1 次戻りコード (**primary_rc**) が生成される条件については、付録 A. APPC 共通戻りコードに説明してあります。

AP_TP_BUSY

AP_UNEXPECTED_SYSTEM_ERROR

AP_CONVERSATION_TYPE_MIXED

[MC_]DEALLOCATE

DEALLOCATE verbは、2 つのトランザクション・プログラム間の会話の割振りを解除します。会話の割振り解除の前に、この verb は次のいずれかの verb と同じ処理を行います。

- **FLUSH** verb。これは、ローカル LU の送信バッファーのデータを、パートナーLU（およびトランザクション・プログラム）に送ります。
- **CONFIRM** verb。これは、ローカル LU の送信バッファー・データおよび確認要求を、パートナー・トランザクション・プログラムに送ります。

この verb が正常に実行された後は、指定した会話 ID (conv_id) は無効になります。

半二重会話の場合

- 指定された会話をトランザクション・プログラムから割振り解除します。それは **FLUSH** または **CONFIRM** verb の機能を含むことができます。

全二重会話の場合

- **TYPE(FLUSH)** 付きの **DEALLOCATE** はローカル・プログラムの送信待ち行列をクローズします。 ローカル・プログラムおよびリモート・プログラムの双方で、それらの送信待ち行列行を独立してクローズする必要があります。したがって、会話を終了させるために 2 つの **DEALLOCATE TYPE(FLUSH)** verb が必要です。パートナーがその送信待ち行列をクローズしたことの通知は、**DEALLOCATE_NORMAL** 戻りコードの形で受信待ち行列に与えられます。
- **TYPE(ABEND)** 付きの **DEALLOCATE** は即時の終了であり、会話の両側を同時にクローズします。この通知は、**ERROR_INDICATION** 戻りコードとしてリモート・プログラムの送信待ち行列へ戻され、**DEALLOCATE_ABEND** 戻りコードとしてリモート・プログラムの受信待ち行列へ戻されます。

VCB 構造:

```
typedef struct deallocate
{
 unsigned short    opcode;             /* verb operation code     */
 unsigned char     opext;              /* verb extension code     */
 unsigned char     format;             /* format                  */
 unsigned short    primary_rc;         /* primary return code     */
 unsigned long     secondary_rc;       /* secondary return code   */
 unsigned char     tp_id[8];           /* TP identifier           */
 unsigned long     conv_id;            /* conversation identifier */
#ifdef WINAPPC_FORMAT_1
 unsigned char     expd_data_rcvd;     /* expedited data received */
 unsigned char     reserv3;            /* reserved                */
#endif
 unsigned char     dealloc_type;       /* deallocate type         */
 unsigned short    log_dlen;           /* log data length         */
 unsigned char    *log_dptr;           /* pointer to log data     */
} DEALLOCATE;

typedef struct mc_deallocate
{
```

```
 unsigned short    opcode;             /* verb operation code     */
 unsigned char     opext;              /* verb extension code     */
 unsigned char     format;             /* format                  */
 unsigned short    primary_rc;         /* primary return code     */
 unsigned long     secondary_rc;       /* secondary return code   */
 unsigned char     tp_id[8];           /* TP identifier           */
 unsigned long     conv_id;            /* conversation identifier */
#ifdef WINAPPC_FORMAT_1
 unsigned char     expd_data_rcvd;     /* expedited data received */
 unsigned char     reserv3;            /* reserved                */
#endif
 unsigned char     dealloc_type;       /* deallocate type         */
 unsigned char     reserv4[2];         /* reserved                */
 unsigned char     reserv5[4];         /* reserved                */
} MC_DEALLOCATE;
```

指定パラメーター: トランザクション・プログラムは、次のパラメーターを Communications Server に提供します。

opcode

AP_B_DEALLOCATE 

AP_M_DEALLOCATE 

opext AP_BASIC_CONVERSATION または AP_MAPPED_CONVERSATION。非ブロッキング操作の場合は、このフラグと AP_NON_BLOCKING を OR で結ぶことができます。

全二重会話では、このフラグと AP_FULL_DUPLEX_CONVERSATION をOR で結ぶ必要があります。 

format

VCB のフォーマットを識別します。 上記に示した VCB のバージョンを指定するには、このフィールドに 1 をセットしてください。

tp_id ローカル・トランザクション・プログラムの識別子。 このパラメーターの値は、ローカルで始動したトランザクション・プログラムでは **TP_STARTED** verb から戻された値であり、リモートから始動されたトランザクション・プログラムでは **RECEIVE_ALLOCATE** verb から戻された値です。

conv_id

会話識別子。 このパラメーターの値は、ローカルで始動したトランザクション・プログラムでは **ALLOCATE** verb から戻された値であり、リモートから始動されたトランザクション・プログラムでは **RECEIVE_ALLOCATE** verb から戻された値です。

dealloc_type

割振り解除をどのように行うかを指定します。

AP_ABEND 

AP_ABEND_PROG 

AP_ABEND_SVC

AP_ABEND_TIMER

AP_FLUSH

AP_SYNC_LEVEL

次のパラメーターは、基本だけに適用されます。 

AP_TP_NOT_AVAIL_NO_RETRY

AP_TP_NOT_AVAIL_RETRY

AP_TPN_NOT_RECOGNIZED

AP_PIP_DATA_NOT_ALLOWED

AP_PIP_DATA_INCORRECT

AP_RESOURCE_FAILURE_NO_RETRY

AP_CONV_TYPE_MISMATCH

AP_SYNC_LVL_NOT_SUPPORTED

AP_SECURITY_PARAMS_INVALID

log_dlen 

エラー・ログ・ファイルに送られるデータのバイト数。

有効範囲は 0 から 32767 です。

アプリケーションは VCB の末尾にデータを付加できますが、その場合はこのフィールドは 0 より大きくなり、**log_dptr** を NULL にセットする必要があります （長さが 0 の場合、エラー・ログ・データがないことを意味します)。

log_dptr 

エラー情報が入っているデータ・バッファーのアドレス。アプリケーションは VCB の末尾にデータを付加できますが、その場合は **log_dptr** を NULL にセットする必要があります。

このデータは、ローカル・エラー・ログおよびパートナー LU に送られます。トランザクション・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。 詳細は、*IBM Systems Network Architecture: LU 6.2 Reference: Peer Protocols* を参照してください。

戻りパラメーター: verb が正常に実行された場合は、Communications Server は次のパラメーターを戻します。

primary_rc
AP_OK

expd_data_rcvd
急送データ受信の標識。 この標識は、RECEIVE_EXPEDITED_DATA が発行されるまで、AP_YES にセットされたままです。

このフィールドには、フォーマット 1 バージョンの VCB が必要です。 フォーマット 1 VCB のアクセスの詳細は、45ページの『全二重 VCB』を参照してください。

AP_YESAP_NO

パラメーター・エラーが原因で verb が実行されなかった場合は、Communications Serverは次のパラメーターを戻します。

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_BAD_CONV_ID

AP_BAD_TP_ID

AP_DEALLOC_BAD_TYPE

AP_DEALLOC_LOG_LL_WRONG 

パラメーター・エラーが原因で verb が実行されなかった場合は、Communications Server は次のパラメーターを戻します（マップ式の場合のみ）。 

primary_rc
AP_OPERATION_INCOMPLETE

opext AP_OPERATION_INCOMPLETE_FLAG

トランザクション・プロセッサーがこの verb を発行したときに会話が不適切な状態にあった場合は、Communications Serverは次のパラメーターを戻します。

primary_rc
AP_STATE_CHECK

secondary_rc
AP_DEALLOC_CONFIRM_BAD_STATE

AP_DEALLOC_FLUSH_BAD_STATE

AP_DEALLOC_NOT_LL_BDY 

次に示す 1 次戻りコード (**primary_rc**) が生成される条件については、付録 A. APPC 共通戻りコードに説明してあります。

AP_ALLOCATION_ERROR

- AP_SECURITY_NOT_VALID
- AP_TRANS_PGM_NOT_AVAIL_RETRY
- AP_TRANS_PGM_NOT_AVAIL_NO_RTRY
- AP_TP_NAME_NOT_RECOGNIZED
- AP_PIP_NOT_ALLOWED
- AP_PIP_NOT_SPECIFIED_CORRECTLY
- AP_CONVERSATION_TYPE_MISMATCH
- AP_SYNC_LEVEL_NOT_SUPPORTED
- AP_CONV_FAILURE_NO_RETRY
- AP_CONV_FAILURE_RETRY
- AP_DEALLOC_ABEND 
- AP_DEALLOC_ABEND_PROG 
- AP_DEALLOC_ABEND_SVC 
- AP_DEALLOC_ABEND_TIMER 
- AP_PROG_ERROR_PURGING
- AP_SVC_ERROR_PURGING 

AP_TP_BUSY
AP_CONVERSATION_TYPE_MIXED
AP_DUPLEX_TYPE_MIXED
AP_UNEXPECTED_SYSTEM_ERROR
AP_CANCELLEDAP_ERROR_INDICATION

- AP_ALLOCATION_ERROR_PENDING
- AP_DEALLOC_ABEND_PROG_PENDING
- AP_DEALLOC_ABEND_SVC_PENDING
- AP_DEALLOC_ABEND_TIMER_PENDING
- AP_UNKNOWN_ERROR_TYPE_PENDING

[MC_]FLUSH

FLUSH verb は、ローカル LU の送信バッファーのデータを、パートナー LU（およびトランザクション・プログラム）に送ります。 送信バッファーが空の場合は、この verb は何も行いません。

VCB 構造:

```
typedef struct flush
{
 unsigned short    opcode;            /* verb operation code     */
 unsigned char     opext;             /* verb extension code     */
 unsigned char     format;            /* format                  */
 unsigned short    primary_rc;        /* primary return code     */
 unsigned long     secondary_rc;      /* secondary return code   */
 unsigned char     tp_id[8];          /* TP identifier           */
 unsigned long     conv_id;           /* conversation identifier */
} FLUSH;

typedef struct mc_flush
{
 unsigned short    opcode;            /* verb operation code     */
 unsigned char     opext;             /* verb extension code     */
 unsigned char     format;            /* format                  */
 unsigned short    primary_rc;        /* primary return code     */
 unsigned long     secondary_rc;      /* secondary return code   */
 unsigned char     tp_id[8];          /* TP identifier           */
 unsigned long     conv_id;           /* conversation identifier */
} MC_FLUSH;
```

指定パラメーター: トランザクション・プログラムは、次のパラメーターを Communications Server に提供します。

opcode

AP_B_FLUSH 

AP_M_FLUSH 

opext AP_BASIC_CONVERSATION または AP_MAPPED_CONVERSATION。非ブロッキング操作の場合は、このフラグと AP_NON_BLOCKING を OR で結ぶことができます。

全二重会話では、このフラグとAP_FULL_DUPLEX_CONVERSATION をOR で結ぶ必要があります。

format

VCB のフォーマットを識別します。 上記に示した VCB のバージョンを指定するには、このフィールドに 0 をセットしてください。

tp_id ローカル・トランザクション・プログラムの識別子。 このパラメーターの値は、ローカルで始動したトランザクション・プログラムでは **TP_STARTED**

verb から戻された値であり、リモートから始動されたトランザクション・プログラムでは **RECEIVE_ALLOCATE** verb から戻された値です。

conv_id
会話識別子。 このパラメーターの値は、ローカルで始動したトランザクション・プログラムでは **ALLOCATE** verb から戻された値であり、リモートから始動されたトランザクション・プログラムでは **RECEIVE_ALLOCATE** verb から戻された値です。

戻りパラメーター: verb が正常に実行された場合は、Communications Server は次のパラメーターを戻します。

primary_rc
AP_OK

verb が非ブロッキングで完了していない場合は、Communications Server は次のパラメーターを戻します。

primary_rc
AP_OPERATION_INCOMPLETE

opext AP_OPERATION_INCOMPLETE_FLAG

パラメーター・エラーが原因で verb が実行されなかった場合は、Communications Serverは次のパラメーターを戻します。

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_BAD_CONV_ID

AP_BAD_TP_ID

トランザクション・プログラムがこの verb を発行したときに会話が不適切な状態にあった場合は、Communications Serverは次のパラメーターを戻します。

primary_rc
AP_STATE_CHECK

secondary_rc
AP_FLUSH_NOT_SEND_STATE

次に示す 1 次戻りコード (**primary_rc**) およびそれに付随する 2 次戻りコード (**secondary_rc**) が生成される条件については、付録 A. APPC 共通戻りコードで説明します。

AP_TP_BUSY
AP_CONVERSATION_TYPE_MIXED

AP_DUPLEX_TYPE_MIXED
AP_UNEXPECTED_SYSTEM_ERROR
AP_ERROR_INDICATION

- AP_ALLOCATION_ERROR_PENDING
- AP_DEALLOC_ABEND_PROG_PENDING
- AP_DEALLOC_ABEND_SVC_PENDING
- AP_DEALLOC_ABEND_TIMER_PENDING
- AP_UNKNOWN_ERROR_TYPE_PENDING

[MC_]GET_ATTRIBUTES

GET_ATTRIBUTES verb は、会話の属性を戻します。

VCB 構造:

```
typedef struct get_attributes
{
 unsigned short    opcode;            /* verb operation code     */
 unsigned char     opext;             /* verb extension code     */
 unsigned char     format;            /* verb format             */
 unsigned short    primary_rc;        /* primary return code     */
 unsigned long     secondary_rc;      /* secondary return code   */
 unsigned char     tp_id[8];          /* TP identifier           */
 unsigned long     conv_id;           /* conversation identifier */
 unsigned char     reserv3;           /* reserved                */
 unsigned char     sync_level;        /* sync_level              */
 unsigned char     mode_name[8];      /* mode name               */
 unsigned char     net_name[8];       /* network name of local LU*/
 unsigned char     lu_name[8];        /* local LU name           */
 unsigned char     lu_alias[8];       /* local LU alias          */
 unsigned char     plu_alias[8];      /* partner LU alias        */
 unsigned char     plu_un_name[8];    /* partner LU uninterpreted name */
 unsigned char     reserv4[2];        /* reserved                */
 unsigned char     fqplu_name[17];    /* fully qualified partner LU    */
 unsigned char     reserv5;           /* reserved                */
 unsigned char     user_id[10];       /* user identifier         */
 unsigned long     conv_group_id;     /* conversation group identifier */
 unsigned char     conv_corr_len;     /* conversation correlator */
                                      /* length                  */
 unsigned char     conv_corr[8];      /* conversation correlator */
 unsigned char     reserv6[13];       /* reserved                */
} GET_ATTRIBUTES;

typedef struct mc_get_attributes
{
 unsigned short    opcode;            /* verb operation code          */
 unsigned char     opext;             /* verb extension code          */
 unsigned char     format;            /* verb format                  */
 unsigned short    primary_rc;        /* primary return code          */
 unsigned long     secondary_rc;      /* secondary return code        */
 unsigned char     tp_id[8];          /* TP identifier                */
 unsigned long     conv_id;           /* conversation identifier      */
 unsigned char     reserv3;           /* reserved                     */
 unsigned char     sync_level;        /* sync_level                   */
 unsigned char     mode_name[8];      /* mode name                    */
 unsigned char     net_name[8];       /* network name of local LU     */
 unsigned char     lu_name[8];        /* local LU name                */
 unsigned char     lu_alias[8];       /* local LU alias               */
 unsigned char     plu_alias[8];      /* partner LU alias             */
 unsigned char     plu_un_name[8];    /* partner LU uninterpreted name */
 unsigned char     reserv4[2];        /* reserved                     */
 unsigned char     fqplu_name[17];    /* fully qualified partner LU   */
                                      /* name                         */
 unsigned char     reserv5;           /* reserved                     */
```

```
 unsigned char     user_id[10];        /* user identifier               */
 unsigned long     conv_group_id;      /* conversation group identifier */
 unsigned char     conv_corr_len;      /* conversation correlator       */
                                       /* length                        */
 unsigned char     conv_corr[8];       /* conversation correlator       */
 unsigned char     reserv6[13];        /* reserved                      */
} MC_GET_ATTRIBUTES;
```

指定パラメーター: トランザクション・プログラムは、次のパラメーターを Communications Server に提供します。

opcode

AP_B_GET_ATTRIBUTES 

AP_M_GET_ATTRIBUTES 

opext AP_BASIC_CONVERSATION または AP_MAPPED_CONVERSATION。

全二重会話では、このフラグと AP_FULL_DUPLEX_CONVERSATION をORで結ぶ必要があります。

format

VCB のフォーマットを識別します。 上記に示した VCB のバージョンを指定するには、このフィールドに 0 をセットしてください。

tp_id ローカル・トランザクション・プログラムの識別子。

このパラメーターの値は、ローカルで始動したトランザクション・プログラムでは **TP_STARTED** verb から戻された値であり、リモートから始動されたトランザクション・プログラムでは **RECEIVE_ALLOCATE** verb から戻された値です。

conv_id

会話識別子。

このパラメーターの値は、ローカルで始動したトランザクション・プログラムでは **ALLOCATE** verb から戻された値であり、リモートから始動されたトランザクション・プログラムでは **RECEIVE_ALLOCATE** verb から戻された値です。

戻りパラメーター: verb が正常に実行された場合は、Communications Serverは次のパラメーターを戻します。

primary_rc

AP_OK

sync_level

会話の同期レベル。

AP_CONFIRM_SYNC_LEVEL
AP_NONE

mode_name

会話に割り振られているセッションのネットワーキング特性の名前。 これは、8 バイトの英数字のタイプ A EBCDIC ストリング（英字で始まるもの）で、8 バイトに満たない場合は右側に EBCDIC のスペースが埋め込まれます。

net_name

ローカル LU を含むネットワークの名前。 これは、8 バイトの英数字のタイプ A EBCDIC ストリング（英字で始まるもの）で、8 バイトに満たない場合は右側に EBCDIC のスペースが埋め込まれます。

lu_name

ローカル LU の名前。 これは、8 バイトの英数字のタイプ A EBCDIC ストリング（英字で始まるもの）で、8 バイトに満たない場合は右側に EBCDIC のスペースが埋め込まれます。

lu_alias

ローカル・トランザクション・プログラムにローカル LU を認識させるための別名。これは、 8 バイトの ASCII 文字ストリングです。 8 バイトのすべてが意味を持つため、すべてのバイトをセットする必要があります。

plu_alias

ローカル・トランザクション・プログラムにパートナー LU を認識させるための別名。これは、 8 バイトの ASCII 文字ストリングです。 8 バイトのすべてが意味を持つため、すべてのバイトをセットする必要があります。

plu_un_name

システム・サービス制御点 (SSCP) で定義されているパートナー LU の名前。これは、 8 バイトのタイプA EBCDIC 文字ストリングです。

fqplu_name

パートナー LU の完全修飾名。 この名前は長さが 17 バイトで、17 バイトに満たない場合は右側に EBCDIC のブランクが埋め込まれます。 この名前は、EBCDIC のピリオドで連結された 2 つのタイプA EBCDIC 文字ストリングです（1 つの名前の長さは最大 8 バイトで、ブランクを含んでいてはなりません。ネットワーク ID がない場合は、ピリオドを省略してください)。

user_id

ローカルのトランザクション・プログラムが、リモートのトランザクション・プログラムにアクセスするために、**ALLOCATE** verb を使用して送るユーザー ID。これは 10 バイトのタイプ AE EBCDIC 文字ストリングで、10 バイトに満たない場合は右側に EBCDIC のスペースが埋め込まれます。

conv_group_id

会話に割り振られるセッションの会話グループ識別子。

conv_corr_len

常に 0 にセットされます。

有効範囲は 0 から 8 です。

conv_corr

常に 0 にセットされます。

パラメーター・エラーが原因で verb が実行されなかった場合は、Communications Serverは次のパラメーターを戻します。

primary_rc

AP_PARAMETER_CHECK

secondary_rc

AP_BAD_CONV_ID

AP_BAD_TP_ID

次に示す 1 次戻りコード (**primary_rc**) およびそれに付随する 2 次戻りコード (**secondary_rc**) が生成される条件については、付録 A. APPC 共通戻りコードで説明します。

AP_TP_BUSY

AP_CONVERSATION_TYPE_MIXED

AP_DUPLEX_TYPE_MIXED

AP_UNEXPECTED_SYSTEM_ERROR

[MC_]PREPARE_TO_RECEIVE

PREPARE_TO_RECEIVE verb は、ローカル・トランザクション・プログラムの会話の状態を、SEND または SEND_PENDING から RECEIVE に変更します。

会話状態を送信から受信に変更する前に、この verb は次のいずれかの verb と同じ処理を行います。

- **FLUSH** verb。これは、ローカル LU の送信バッファーのデータを、パートナーLU（およびトランザクション・プログラム）に送ります。
- **CONFIRM** verb。これは、ローカル LU の送信バッファーのデータ、および確認要求を、パートナー・トランザクション・プログラムに送ります。

この verb が正常に実行された後は、ローカル・トランザクション・プログラムはデータを受信できるようになります。

VCB 構造:

```
typedef struct prepare_to_receive
{
 unsigned short    opcode;             /* verb operation code     */
 unsigned char     opext;              /* verb extension code     */
 unsigned char     format;             /* format                  */
 unsigned short    primary_rc;         /* primary return code     */
 unsigned long     secondary_rc;       /* secondary return code   */
 unsigned char     tp_id[8];           /* TP identifier           */
 unsigned long     conv_id;            /* conversation identifier  */
 unsigned char     ptr_type;           /* prepare to receive type  */
 unsigned char     locks;              /* prepare to receive locks */
} PREPARE_TO_RECEIVE;

typedef struct mc_prepare_to_receive
{
 unsigned short    opcode;             /* verb operation code     */
 unsigned char     opext;              /* verb extension code     */
 unsigned char     format;             /* format                  */
 unsigned short    primary_rc;         /* primary return code     */
 unsigned long     secondary_rc;       /* secondary return code   */
 unsigned char     tp_id[8];           /* TP identifier           */
 unsigned long     conv_id;            /* conversation identifier */
 unsigned char     ptr_type;           /* prepare to receive type */
 unsigned char     locks;              /* prepare to receive locks*/
} MC_PREPARE_TO_RECEIVE;
```

指定パラメーター: トランザクション・プログラムは、次のパラメーターを Communications Server に提供します。

opcode

AP_B_PREPARE_TO_RECEIVE 

AP_M_PREPARE_TO_RECEIVE 

opext AP_BASIC_CONVERSATION または AP_MAPPED_CONVERSATION。非ブロッキング操作の場合は、このフラグと AP_NON_BLOCKING を OR で結ぶことができます。

format
VCB のフォーマットを識別します。 上記に示した VCB のバージョンを指定するには、このフィールドに 0 をセットしてください。

tp_id ローカル・トランザクション・プログラムの識別子。 このパラメーターの値は、ローカルで始動したトランザクション・プログラムでは **TP_STARTED** verb から戻された値であり、リモートから始動されたトランザクション・プログラムでは **RECEIVE_ALLOCATE** verb から戻された値です。

conv_id
会話識別子。

このパラメーターの値は、ローカルで始動したトランザクション・プログラムでは **ALLOCATE** verb から戻された値であり、リモートから始動されたトランザクション・プログラムでは **RECEIVE_ALLOCATE** verb から戻された値です。

ptr_type
会話を受信状態に変更するときに、APPC がとるアクションのタイプ。

AP_FLUSHAP_SYNC_LEVEL
AP_P_TO_R_CONFIRM

locks Communications Serverがローカル・トランザクション・プロセッサーへ、いつ制御を戻すかを指定します。

AP_LONG

AP_SHORT

戻りパラメーター: verb が正常に実行された場合は、Communications Server は次のパラメーターを戻します。

primary_rc
AP_OK

verb が非ブロッキングで完了していない場合は、Communications Serverは次のパラメーターを戻します。

primary_rc
AP_OPERATION_INCOMPLETE

opext AP_OPERATION_INCOMPLETE_FLAG

パラメーター・エラーが原因で verb が実行されなかった場合は、Communications Serverは次のパラメーターを戻します。

primary_rc
AP_PARAMETER_CHECK

secondary_rc

AP_BAD_CONV_ID

AP_BAD_TP_ID
AP_P_TO_R_INVALID_FOR_FDX
AP_P_TO_R_INVALID_TYPE

トランザクション・プロセッサーがこの verb を発行したときに会話が不適切な状態にあった場合は、Communications Serverは次のパラメーターを戻します。

primary_rc

AP_STATE_CHECK

secondary_rc

AP_TO_R_NOT_LL_BDY 

AP_P_TO_R_NOT_SEND_STATE

次に示す 1 次戻りコード (**primary_rc**) およびそれに付随する 2 次戻りコード (**secondary_rc**) が生成される条件については、付録 A. APPC 共通戻りコードで説明します。

AP_ALLOCATION_ERROR

AP_SECURITY_NOT_VALID
AP_TRANS_PGM_NOT_AVAIL_RETRY
AP_TRANS_PGM_NOT_AVAIL_NO_RTRY
AP_TP_NAME_NOT_RECOGNIZED
AP_PIP_NOT_ALLOWED
AP_PIP_NOT_SPECIFIED_CORRECTLY
AP_CONVERSATION_TYPE_MISMATCH
AP_SYNC_LEVEL_NOT_SUPPORTED
AP_CONV_FAILURE_NO_RETRY
AP_CONV_FAILURE_RETRY
AP_DEALLOC_ABEND 

AP_DEALLOC_ABEND_PROG 

AP_DEALLOC_ABEND_SVC 

AP_DEALLOC_ABEND_TIMER

MC_PREPARE_TO_RECEIVE

AP_PROG_ERROR_PURGING 

AP_SVC_ERROR_PURGING 

AP_TP_BUSY

AP_CONVERSATION_TYPE_MIXED

AP_UNEXPECTED_SYSTEM_ERROR

AP_CANCELLED

[MC_]RECEIVE_AND_POST

RECEIVE_AND_POST verb は、アプリケーション・データと会話状況情報を非同期に受信します。 この verb を用いると、データがローカル LU に到着しているときでも、トランザクション・プログラムが処理を続けることができます。 この verb は、APPC エントリー・ポイントを介してのみ発行できます。

Win 3.1 SNA API クライアントでは使用できません。

VCB 構造:

```
typedef struct receive_and_post
{
 unsigned short    opcode;             /* verb operation code        */
 unsigned char     opext;              /* verb extension code        */
 unsigned char     format;             /* format                     */
 unsigned short    primary_rc;         /* primary return code        */
 unsigned long     secondary_rc;       /* secondary return code      */
 unsigned char     tp_id[8];           /* TP identifier              */
 unsigned long     conv_id;            /* conversation identifier    */
 unsigned short    what_rcvd;          /* what received              */
 unsigned char     rtn_status;         /* return status with data    */
 unsigned char     fill;               /* data fill                  */
 unsigned char     rts_rcvd;           /* request to send received   */
 unsigned char     expd_data_rcvd;     /* expedited data received    */
 unsigned short    max_len;            /* maximum length of received */
                                       /* data                       */
 unsigned short    dlen;               /* actual length of received  */
                                       /* data                       */
 unsigned char    *dptr;               /* pointer to data buffer     */
 unsigned long    *sema;               /* post handle for verb       */
 unsigned char     reserv5;            /* reserved                   */
} RECEIVE_AND_POST;

typedef struct mc_receive_and_post
{
 unsigned short    opcode;             /* verb operation code        */
 unsigned char     opext;              /* verb extension code        */
 unsigned char     format;             /* format                     */
 unsigned short    primary_rc;         /* primary return code        */
 unsigned long     secondary_rc;       /* secondary return code      */
 unsigned char     tp_id[8];           /* TP identifier              */
 unsigned long     conv_id;            /* conversation identifier    */
 unsigned short    what_rcvd;          /* what received              */
 unsigned char     rtn_status;         /* return status with data    */
 unsigned char     reserv4;            /* reserved                   */
 unsigned char     rts_rcvd;           /* request to send received   */
 unsigned char     expd_data_rcvd;     /* expedited data received    */
 unsigned short    max_len;            /* maximum length of received */
                                       /* data                       */
 unsigned short    dlen;               /* actual length of received  */
                                       /* data                       */
```

```
 unsigned char    *dptr;              /* pointer to data buffer     */
 unsigned long    *sema;              /* post handle for verb       */
 unsigned char     reserv6;           /* reserved                   */
} MC_RECEIVE_AND_POST;
```

指定パラメーター: トランザクション・プログラムは、次のパラメーターを Communications Server に提供します。

opcode

AP_B_RECEIVE_AND_POST 

AP_M_RECEIVE_AND_POST 

opext AP_BASIC_CONVERSATIONまたは AP_MAPPED_CONVERSATION。

format

VCB のフォーマットを識別します。 上記に示した VCB のバージョンを指定するには、このフィールドに 0 をセットしてください。

tp_id ローカル・トランザクション・プログラムの識別子。 このパラメーターの値は、ローカルで始動したトランザクション・プログラムでは **TP_STARTED** verb から戻された値であり、リモートから始動されたトランザクション・プログラムでは **RECEIVE_ALLOCATE** verb から戻された値です。

conv_id

会話識別子。

このパラメーターの値は、ローカルで始動したトランザクション・プログラムでは **ALLOCATE** verb から戻された値であり、リモートから始動されたトランザクション・プログラムでは **RECEIVE_ALLOCATE** verb から戻された値です。

rtn_status

同じ verb で会話状況情報とデータを戻せるかどうかを指定します。

AP_YES

AP_NO

fill 

ローカル・トランザクション・プログラムがデータを受信するときの形式を指定します。

AP_BUFFER

AP_LL

max_len

ローカル・トランザクション・プログラムが受信できるデータの最大バイト数。

有効範囲は 0 から 65535 です。

この値は、受信データが入るバッファーの長さを超えるものであってはなりません。

dptr ローカル LU が受信するデータを入れるバッファーのアドレス。アプリケーションは VCB の末尾にデータを付加できますが、その場合は **dptr** を NULL にセットする必要があります。

sema アプリケーションが待機するイベントのハンドル。 この verb は、Win32 API では WaitForMultipleObjects とともに、また OS/2 では DosWaitEventSem とともに使用するためのものです。

戻りパラメーター: verb が正常に実行された場合は、Communications Serverは次のパラメーターを戻します。

primary_rc

AP_OK

AP_DEALLOC_NORMAL

what_rcvd

着信データといっしょに受信した状況情報。 **rtn_status** が AP_NO にセットされている場合は、このフィールドは、常に次のリストの最初の部分にある値を含みます。 **rtn_status** が AP_YES にセットされている場合は、このフィールドはリストの任意の値を含むことができます。

AP_NONE

AP_CONFIRM_DEALLOCATE

AP_CONFIRM_SEND

AP_CONFIRM_WHAT_RECEIVED

AP_DATA 

AP_DATA_COMPLETE

AP_DATA_INCOMPLETE

AP_SEND

AP_USER_CONTROL_DATA_COMPLETE 

AP_USER_CONTROL_DATA_INCMP 

AP_PS_HEADER_COMPLETE 

AP_PS_HEADER_INCOMPLETE 

AP_DATA_CONFIRM 

AP_DATA_COMPLETE_CONFIRM

AP_DATA_CONFIRM_DEALLOCATE

AP_DATA_COMPLETE_CONFIRM_DEALL

AP_DATA_CONFIRM_SEND

AP_DATA_COMPLETE_CONFIRM_SEND

AP_DATA_SEND

AP_DATA_COMPLETE_SEND

次のパラメーターは、マップ式にだけ適用されます。 

AP_UC_DATA_COMPLETE_CONFIRM

AP_UC_DATA_COMPLETE_CNFM_DEALL

AP_UC_DATA_COMPLETE_CNFM_SEND

AP_UC_DATA_COMPLETE_SEND

AP_PS_HDR_COMPLETE_CONFIRM

AP_PS_HDR_COMPLETE_CNFM_DEALL

AP_PS_HDR_COMPLETE_CNFM_SEND

AP_PS_HDR_COMPLETE_SEND

rts_rcvd

送信要求受信の標識。

AP_YES

AP_NO

expd_data_rcvd

急送データ受信の標識。 この標識は、RECEIVE_EXPEDITED_DATA が発行されるまで、AP_YES にセットされたままです。

AP_YES

AP_NO

このフィールドには、VCB のフォーマット 1 のバージョンが必要です。フォーマット 1 VCB のアクセスの詳細は、45ページの『全二重 VCB』を参照してください。

dlen 受信したデータのバイト数（このデータは、**dptr** パラメーターで指定するバッファーに格納されます)。 長さが 0 の場合、データを受信しなかったことを示します。 このパラメーターが使用されるのは、**what_rcvd** パラメーターが、データを受信したことを示している場合だけです。

パラメーター・エラーが原因で verb が実行されなかった場合は、Communications Serverは次のパラメーターを戻します。

primary_rc

AP_PARAMETER_CHECK

secondary_rc

AP_BAD_CONV_ID

AP_BAD_RETURN_STATUS_WITH_DATA

AP_BAD_TP_ID

AP_RCV_AND_POST_BAD_FILL 

トランザクション・プログラムがこの verb を発行したときに会話が不適切な状態にあった場合は、Communications Serverは次のパラメーターを戻します。

primary_rc

AP_STATE_CHECK

secondary_rc

AP_RCV_AND_POST_BAD_STATE

AP_RCV_AND_POST_NOT_LL_BDY 

トランザクション・プログラムが発行した他の verb により、この verb が取り消された場合は、Communications Serverは次のパラメーターを戻します。

primary_rc

AP_CANCELLED

次に示す 1 次戻りコード (**primary_rc**) およびそれに付随する 2 次戻りコード (**secondary_rc**) が生成される条件については、付録 A. APPC 共通戻りコードで説明します。

AP_ALLOCATION_ERROR

AP_SECURITY_NOT_VALID

AP_TRANS_PGM_NOT_AVAIL_RETRY

AP_TRANS_PGM_NOT_AVAIL_NO_RTRY

AP_TP_NAME_NOT_RECOGNIZED

AP_PIP_NOT_ALLOWED

AP_PIP_NOT_SPECIFIED_CORRECTLY

AP_CONVERSATION_TYPE_MISMATCH

AP_SYNC_LEVEL_NOT_SUPPORTED

AP_CONV_FAILURE_NO_RETRY

AP_CONV_FAILURE_RETRY

AP_DEALLOC_ABEND 

AP_DEALLOC_ABEND_PROG 

AP_DEALLOC_ABEND_SVC 

AP_DEALLOC_ABEND_TIMER

MC_RECEIVE_AND_POST

AP_DEALLOC_NORMAL

AP_PROG_ERROR_NO_TRUNC

AP_PROG_ERROR_PURGING 

AP_PROG_ERROR_TRUNC 

AP_SVC_ERROR_NO_TRUNC 

AP_SVC_ERROR_PURGING 

AP_SVC_ERROR_TRUNC

AP_TP_BUSY

AP_CONVERSATION_TYPE_MIXED

AP_UNEXPECTED_SYSTEM_ERROR

AP_CANCELLED

[MC_]RECEIVE_AND_WAIT

RECEIVE_AND_WAIT verb は、パートナー・トランザクション・プログラムから現在使用可能なデータを受信します。 現在使用可能なデータがない場合は、ローカル・トランザクション・プログラムはデータが到着するまで待ちます。

半二重会話の場合

- プログラムは、会話が送信状態にあるとき、この verb を発行できます。この場合、LU はその送信バッファーをフラッシュして、バッファーにあるすべての情報と SEND 標識をリモート・プログラムへ送ります。 そして会話を受信状態へ変更します。 次に、LU は情報の到着を待機します。 リモート・プログラムは、それが SEND 標識を受け取った後で、データをローカル・プログラムへ送ることができます。

全二重会話の場合

- 送信バッファーが会話割振り要求を含んでいる場合、送信バッファーはフラッシュされます。そうでなれば、この verb によって LU がその送信バッファーをフラッシュすることはありません。送信バッファーにデータを受け取る前に送る必要があるデータが残っている場合は、ローカル・プログラムは、FLUSH を発行してからこの verb を発行する必要があります。

VCB 構造:

```
typedef struct receive_and_wait
{
 unsigned short    opcode;             /* verb operation code        */
 unsigned char     opext;              /* verb extension code        */
 unsigned char     format;             /* format                     */
 unsigned short    primary_rc;         /* primary return code        */
 unsigned long     secondary_rc;       /* secondary return code      */
 unsigned char     tp_id[8];           /* TP identifier              */
 unsigned long     conv_id;            /* conversation identifier    */
 unsigned short    what_rcvd;          /* what received              */
 unsigned char     rtn_status;         /* return status with data    */
 unsigned char     fill;               /* data fill                  */
 unsigned char     rts_rcvd;           /* request to send received   */
 unsigned char     expd_data_rcvd;     /* expedited data received    */
 unsigned short    max_len;            /* maximum length of received */
                                       /* data                       */
 unsigned short    dlen;               /* actual length of received  */
                                       /* data                       */
 unsigned char    *dptr;               /* pointer to data buffer     */
 unsigned char     reserv5[5];         /* reserved                   */
} RECEIVE_AND_WAIT;

typedef struct mc_receive_and_wait
{
 unsigned short    opcode;             /* verb operation code        */
 unsigned char     opext;              /* verb extension code        */
 unsigned char     format;             /* format                     */
 unsigned short    primary_rc;         /* primary return code        */
 unsigned long     secondary_rc;       /* secondary return code      */
```

```
 unsigned char     tp_id[8];           /* TP identifier              */
 unsigned long     conv_id;            /* conversation identifier    */
 unsigned short    what_rcvd;          /* what received              */
 unsigned char     rtn_status;         /* return status with data    */
 unsigned char     reserv4;            /* reserved                   */
 unsigned char     rts_rcvd;           /* request to send received   */
 unsigned char     expd_data_rcvd;     /* expedited data received    */
 unsigned short    max_len;            /* maximum length of received */
                                       /* data                       */
 unsigned short    dlen;               /* actual length of received  */
                                       /* data                       */
 unsigned char    *dptr;               /* pointer to data buffer     */
 unsigned char     reserv6[5];         /* reserved                   */
} MC_RECEIVE_AND_WAIT;
```

指定パラメーター: トランザクション・プログラムは、次のパラメーターを Communications Server に提供します。

opcode

AP_B_RECEIVE_AND_WAIT 

AP_M_RECEIVE_AND_WAIT 

opext AP_BASIC_CONVERSATION または AP_MAPPED_CONVERSATION。非ブロッキング操作の場合は、このフラグと AP_NON_BLOCKING を OR で結ぶことができます。

全二重会話では、このフラグと AP_FULL_DUPLEX_CONVERSATION をOR で結ぶ必要があります。

format

VCB のフォーマットを識別します。 上記に示した VCB のバージョンを指定するには、このフィールドに 0 をセットしてください。

tp_id ローカル・トランザクション・プログラムの識別子。

このパラメーターの値は、ローカルで始動したトランザクション・プログラムでは **TP_STARTED** verb から戻された値であり、リモートから始動されたトランザクション・プログラムでは **RECEIVE_ALLOCATE** verb から戻された値です。

conv_id

会話識別子。

このパラメーターの値は、ローカルで始動したトランザクション・プログラムでは **ALLOCATE** verb から戻された値であり、リモートから始動されたトランザクション・プログラムでは **RECEIVE_ALLOCATE** verb から戻された値です。

rtn_status

同じ verb で会話状況情報とデータを戻せるかどうかを指定します。

AP_YES

AP_NO

fill 
ローカル・トランザクション・プログラムがデータを受信するときの形式を指定します。

AP_BUFFER

AP_LL

max_len
ローカル・トランザクション・プログラムが受信できるデータの最大バイト数。

有効範囲は 0 から 65535 です。

この値は、受信データが入るバッファーの長さを超えるものであってはなりません。

dptr ローカル LU が受信するデータを入れるバッファーのアドレス。アプリケーションは VCB の末尾にデータを付加できますが、その場合は **dptr** を NULL にセットする必要があります。

戻りパラメーター: verb が正常に実行された場合は、Communications Serverは次のパラメーターを戻します。

primary_rc
AP_OK

AP_DEALLOC_NORMAL

what_rcvd
着信データといっしょに受信した状況情報。 **rtn_status** が AP_NO にセットされている場合は、このフィールドは、常に次のリストの最初の部分にある値を含みます。 **rtn_status** が AP_YES にセットされている場合は、このフィールドはリストの任意の値を含むことができます。

AP_NONE

AP_CONFIRM_DEALLOCATE

AP_CONFIRM_SEND

AP_CONFIRM_WHAT_RECEIVED

AP_DATA 

AP_DATA_COMPLETE

AP_DATA_INCOMPLETE

AP_SEND

AP_USER_CONTROL_DATA_COMPLETE 

AP_USER_CONTROL_DATA_INCMP

AP_PS_HEADER_COMPLETE 

AP_PS_HEADER_INCOMPLETE 

AP_DATA_CONFIRM 

AP_DATA_COMPLETE_CONFIRM

AP_DATA_CONFIRM_DEALLOCATE 

AP_DATA_COMPLETE_CONFIRM_DEALL

AP_DATA_CONFIRM_SEND 

AP_DATA_COMPLETE_CONFIRM_SEND

AP_DATA_SEND 

AP_DATA_COMPLETE_SEND

次のパラメーターは、マップ式だけに適用されます。 

AP_UC_DATA_COMPLETE_CONFIRM

AP_UC_DATA_COMPLETE_CNFM_DEALL

AP_UC_DATA_COMPLETE_CNFM_SEND

AP_UC_DATA_COMPLETE_SEND

AP_PS_HDR_COMPLETE_CONFIRM

AP_PS_HDR_COMPLETE_CNFM_DEALL

AP_PS_HDR_COMPLETE_CNFM_SEND

AP_PS_HDR_COMPLETE_SEND

rts_rcvd

送信要求受信の標識。

AP_YES

AP_NO

次の verb のこのフォーマットは VCB のフォー マット 1 のバージョンです。フォーマット 1 VCB のアクセスの詳細は、45ページの『全二重 VCB』を参照してください。

expd_data_rcvd

急送データ受信の標識。 この標識は、RECEIVE_EXPEDITED_DATA が発行されるまで、AP_YES にセットされたままです。

AP_YES

AP_NO

dlen このパラメーターが使用されるのは、**what_rcvd** パラメーターがデータの到着を示している場合だけです。 受信したデータのバイト数（このデータは、**dptr** パラメーターで指定するバッファーに格納されます)。 長さが 0 の場合、データを受信しなかったことを示します。

verb が非ブロッキングで完了していない場合は、Communications Serverは次のパラメーターを戻します。

primary_rc

AP_OPERATION_INCOMPLETE

opext AP_OPERATION_INCOMPLETE_FLAG

パラメーター・エラーが原因で verb が実行されなかった場合は、Communications Serverは次のパラメーターを戻します。

primary_rc

AP_PARAMETER_CHECK

secondary_rc

AP_BAD_CONV_ID

AP_BAD_RETURN_STATUS_WITH_DATA

AP_BAD_TP_ID

AP_RCV_AND_WAIT_BAD_FILL 

トランザクション・プログラムがこの verb を発行したときに会話が不適切な状態にあった場合は、Communications Serverは次のパラメーターを戻します。

primary_rc

AP_STATE_CHECK

secondary_rc

AP_RCV_AND_WAIT_BAD_STATE

AP_RCV_AND_WAIT_NOT_LL_BDY 

次に示す 1 次戻りコード (**primary_rc**) およびそれに付随する 2 次戻りコード (**secondary_rc**) が生成される条件については、付録 A. APPC 共通戻りコードで説明します。

AP_ALLOCATION_ERROR

AP_SECURITY_NOT_VALID

AP_TRANS_PGM_NOT_AVAIL_RETRY

AP_TRANS_PGM_NOT_AVAIL_NO_RTRY

AP_TP_NAME_NOT_RECOGNIZED

AP_PIP_NOT_ALLOWED

AP_PIP_NOT_SPECIFIED_CORRECTLY

AP_CONVERSATION_TYPE_MISMATCH

AP_SYNC_LEVEL_NOT_SUPPORTED

AP_CONV_FAILURE_NO_RETRY

AP_CONV_FAILURE_RETRY

AP_DEALLOC_ABEND 

AP_DEALLOC_ABEND_PROG 

AP_DEALLOC_ABEND_SVC 

AP_DEALLOC_ABEND_TIMER 

AP_DEALLOC_NORMAL

AP_PROG_ERROR_NO_TRUNC

AP_PROG_ERROR_PURGING

AP_PROG_ERROR_TRUNC 

AP_SVC_ERROR_NO_TRUNC 

AP_SVC_ERROR_PURGING 

AP_SVC_ERROR_TRUNC 

AP_TP_BUSY

AP_CONVERSATION_TYPE_MIXED

AP_DUPLEX_TYPE_MIXED

AP_UNEXPECTED_SYSTEM_ERROR

AP_CANCELLED

[MC_]RECEIVE_EXPEDITED_DATA

これは、サーバーにのっているアプリケーションに対してだけサポートされます。

[MC_]RECEIVE_EXPEDITED_DATA verb は、現在利用できる急送データをパートナー TP から受け取ります。 現在、急送データが利用可能であれば、ローカル・トランザクション・プログラムは待機することなくそれを受け取ります。利用可能でなければ、ローカル・トランザクション・プログラムの動作は **rtn_ctl** フィールドによって決まります。

VCB 構造:

```
typedef struct receive_expedited_data
{
 unsigned short    opcode;             /* verb operation code        */
 unsigned char     opext;              /* verb extension code        */
 unsigned char     format;             /* format                     */
 unsigned short    primary_rc;         /* primary return code        */
 unsigned long     secondary_rc;       /* secondary return code      */
 unsigned char     tp_id[8];           /* TP identifier              */
 unsigned long     conv_id;            /* conversation identifier    */
 unsigned char     return_control;     /* when to return control     */
 unsigned char     reserv1[3];         /* reserved                   */
 unsigned char     rts_rcvd;           /* request to send received   */
 unsigned char     expd_data_rcvd;     /* expedited data received    */
 unsigned short    max_len;            /* maximum length of received */
                                       /* data                       */
 unsigned short    dlen;               /* actual length of received  */
                                       /* data                       */
 unsigned char    *dptr;               /* pointer to data buffer     */
} RECEIVE_EXPEDITED_DATA

typedef struct mc_receive_expedited_data
{
 unsigned short    opcode;             /* verb operation code        */
 unsigned char     opext;              /* verb extension code        */
 unsigned char     format;             /* format                     */
 unsigned short    primary_rc;         /* primary return code        */
 unsigned long     secondary_rc;       /* secondary return code      */
 unsigned char     tp_id[8];           /* TP identifier              */
 unsigned long     conv_id;            /* conversation identifier    */
 unsigned char     return_control;     /* when to return control     */
 unsigned char     reserv1[3];         /* reserved                   */
 unsigned char     rts_rcvd;           /* request to send received   */
 unsigned char     expd_data_rcvd;     /* expedited data received    */
 unsigned short    max_len;            /* maximum length of received */
                                       /* data                       */
 unsigned short    dlen;               /* actual length of received  */
                                       /* data                       */
 unsigned char    *dptr;               /* pointer to data buffer     */
} MC_RECEIVE_EXPEDITED_DATA
```

指定パラメーター: トランザクション・プログラムは、次のパラメーターを Communications Server に提供します。

opcode

AP_B_RECEIVE_EXPEDITED_DATA 

AP_M_RECEIVE_EXPEDITED_DATA 

opext AP_BASIC_CONVERSATION または AP_MAPPED_CONVERSATION。非ブロッキング操作の場合は、このフラグと AP_NON_BLOCKING を OR で結ぶことができます。

全二重会話では、このフラグと AP_FULL_DUPLEX_CONVERSATION をOR で結ぶ必要があります。

format

VCB のフォーマットを識別します。 上記に示した VCB のバージョンを指定するには、このフィールドに 0 をセットしてください。

tp_id ローカル・トランザクション・プログラムの識別子。

このパラメーターの値は、ローカルで始動したトランザクション・プログラムでは **TP_STARTED** verb から戻された値であり、リモートから始動されたトランザクション・プログラムでは **RECEIVE_ALLOCATE** verb から戻された値です。

conv_id

会話識別子。

このパラメーターの値は、ローカルで始動したトランザクション・プログラムでは **ALLOCATE** verb から戻された値であり、リモートから始動されたトランザクション・プログラムでは **RECEIVE_ALLOCATE** verb から戻された値です。

return_control

トランザクション・プログラムへいつ制御を戻すかを指定します。

AP_WHEN_EXPD_RECEIVED

AP_IMMEDIATE

max_len

ローカル・トランザクション・プログラムが受信できるデータの最大バイト数。

有効範囲は 0 から 86 です。

この値は、受信データが入るバッファーの長さを超えるものであってはなりません。

dptr ローカル LU が受信するデータを入れるバッファーのアドレス。アプリケーションは VCB の末尾にデータを付加できますが、その場合は **dptr** を NULL にセットする必要があります。

戻りパラメーター: verb が正常に実行された場合は、Communications Serverは次のパラメーターを戻します。

primary_rc
AP_OK

rts_rcvd
送信要求受信の標識。

AP_YES

AP_NO

expd_data_rcvd
急送データ受信の標識。 この標識は、RECEIVE_EXPEDITED_DATA が発行されるまで、AP_YES にセットされたままです。

AP_YESAP_NO

dlen 受信したデータのバイト数（このデータは、**dptr** パラメーターで指定するバッファーに格納されます)。 長さが 0 の場合、データを受信しなかったことを示します。 受信したデータはフォーマットされていないことに注意してください。 2 バイトの長さフィールド (LL) はありません。

verb が非ブロッキングで完了していない場合は、Communications Server は次のパラメーターを戻します。

primary_rc
AP_OPERATION_INCOMPLETE 

opext AP_OPERATION_INCOMPLETE_FLAG

リモート LU が急送データをサポートしていないために verb が実行されない場合は、Communications Server は次のパラメーターを戻します。

primary_rc
AP_EXPD_NOT_SUPPORTED_BY_LU

データがすぐにパートナー・トランザクション・プログラムから利用できず、また **rtn_ctl** フラグが AP_IMMEDIATE である場合は、Communications Serverは次のパラメーターを戻します。

primary_rc
AP_UNSUCCESSFUL

トランザクション・プログラムによって提供されたデータ・バッファーが LU からの利用可能な急送データのすべてを入れることができるほど大きくない場合は、その急送データは戻されず、Communications Server は次のパラメーターを戻します。

primary_rc
AP_BUFFER_TOO_SMALL

dlen LU が受け取ることのできる急送データのバイト数。

パラメーター・エラーが原因で verb が実行されなかった場合は、Communications Serverは次のパラメーターを戻します。

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_BAD_CONV_ID

AP_BAD_TP_ID
AP_EXPD_BAD_RETURN_CONTROL
AP_RCV_EXPD_INVALID_LENGTH

トランザクション・プログラムがこの verb を発行したときに会話が不適切な状態にあった場合は、Communications Serverは次のパラメーターを戻します。

primary_rc
AP_STATE_CHECK

secondary_rc
AP_EXPD_DATA_BAD_CONV_STATE

次に示す 1 次戻りコード (**primary_rc**) およびそれに付随する 2 次戻りコード (**secondary_rc**) が生成される条件については、付録 A. APPC 共通戻りコードで説明します。

AP_ALLOCATION_ERROR
AP_SECURITY_NOT_VALID
AP_TRANS_PGM_NOT_AVAIL_RETRY
AP_TRANS_PGM_NOT_AVAIL_NO_RTRY
AP_TP_NAME_NOT_RECOGNIZED
AP_PIP_NOT_ALLOWED
AP_PIP_NOT_SPECIFIED_CORRECTLY
AP_CONVERSATION_TYPE_MISMATCH
AP_SYNC_LEVEL_NOT_SUPPORTED
AP_CONV_FAILURE_NO_RETRY
AP_CONV_FAILURE_RETRY
AP_DEALLOC_ABEND_PROG
AP_DEALLOC_ABEND_SVC
AP_DEALLOC_ABEND_TIMER
AP_DEALLOC_NORMAL
AP_TP_BUSY
AP_CONVERSATION_TYPE_MIXED

AP_DUPLEX_TYPE_MIXED

AP_UNEXPECTED_SYSTEM_ERROR

AP_CANCELLED

AP_ERROR_INDICATION 

[MC_]RECEIVE_IMMEDIATE

[MC_]RECEIVE_IMMEDIATE verb は、パートナー・トランザクション・プログラムから現在使用可能なすべてのデータまたは状況情報を受信します。現在使用可能なデータがない場合は、ローカル・トランザクション・プログラムはデータが到着するのを待たず、すぐに制御が戻ります。

VCB 構造:

```
typedef struct receive_immediate
{
 unsigned shor     opcode;             /* verb operation code        */
 unsigned char     opext;              /* verb extension code        */
 unsigned char     format;             /* format                     */
 unsigned short    primary_rc;         /* primary return code        */
 unsigned long     secondary_rc;       /* secondary return code      */
 unsigned char     tp_id[8];           /* TP identifier              */
 unsigned long     conv_id;            /* conversation identifier    */
 unsigned short    what_rcvd;          /* what received              */
 unsigned char     rtn_status;         /* return status with data    */
 unsigned char     fill;               /* data fill                  */
 unsigned char     rts_rcvd;           /* request to send received   */
 unsigned char     expd_data_rcvd;     /* expedited data received    */
 unsigned short    max_len;            /* maximum length of received */
                                       /* data                       */
 unsigned short    dlen;               /* actual length of received  */
                                       /* data                       */
 unsigned char    *dptr;               /* pointer to data buffer     */
 unsigned char     reserv5[5];         /* reserved                   */
} RECEIVE_IMMEDIATE;

typedef struct mc_receive_immediate
{
 unsigned short    opcode;             /* verb operation code        */
 unsigned char     opext;              /* verb extension code        */
 unsigned char     format;             /* format                     */
 unsigned short    primary_rc;         /* primary return code        */
 unsigned long     secondary_rc;       /* secondary return code      */
 unsigned char     tp_id[8];           /* TP identifier              */
 unsigned long     conv_id;            /* conversation identifier    */
 unsigned short    what_rcvd;          /* what received              */
 unsigned char     rtn_status;         /* return status with data    */
 unsigned char     reserv4;            /* reserved                   */
 unsigned char     rts_rcvd;           /* request to send received   */
 unsigned char     expd_data_rcvd;     /* expedited data received    */
 unsigned short    max_len;            /* maximum length of received */
                                       /* data                       */
 unsigned short    dlen;               /* actual length of received  */
                                       /* data                       */
 unsigned char    *dptr;               /* pointer to data buffer     */
 unsigned char     reserv6[5];         /* reserved                   */
} MC_RECEIVE_IMMEDIATE;
```

指定パラメーター: トランザクション・プログラムは、次のパラメーターを Communications Server に提供します。

opcode

AP_B_RECEIVE_IMMEDIATE 

AP_M_RECEIVE_IMMEDIATE 

opext AP_BASIC_CONVERSATION または AP_MAPPED_CONVERSATION。非ブロッキング操作の場合は、このフラグと AP_NON_BLOCKING を OR で結ぶことができます。

全二重会話では、このフラグと AP_FULL_DUPLEX_CONVERSATION をOR で結ぶ必要があります。

format

VCB のフォーマットを識別します。 上記に示した VCB のバージョンを指定するには、このフィールドに 0 をセットしてください。

tp_id ローカル・トランザクション・プログラムの識別子。

このパラメーターの値は、ローカルで始動したトランザクション・プログラムでは **TP_STARTED** verb から戻された値であり、リモートから始動されたトランザクション・プログラムでは **RECEIVE_ALLOCATE** verb から戻された値です。

conv_id

会話識別子。

このパラメーターの値は、ローカルで始動したトランザクション・プログラムでは **[MC_]ALLOCATE** verb から戻された値であり、リモートから始動されたトランザクション・プログラムでは **RECEIVE_ALLOCATE** verb から戻された値です。

rtn_status

同じ verb で会話状況情報とデータを戻せるかどうかを指定します。

AP_YES

AP_NO

fill 

ローカル・トランザクション・プログラムがデータを受信するときの形式を指定します。

AP_BUFFER

AP_LL

max_len

ローカル・トランザクション・プログラムが受信できるデータの最大バイト数。

有効範囲は 0 から 65535 です。

この値は、受信データが入るバッファーの長さを超えるものであってはなりません。

dptr ローカル LU が受信するデータを入れるバッファーのアドレス。アプリケーションは VCB の末尾にデータを付加できますが、その場合は **dptr** を NULL にセットする必要があります。

戻りパラメーター: verb が正常に実行された場合は、Communications Serverは次のパラメーターを戻します。

primary_rc

AP_OK

AP_DEALLOC_NORMAL

what_rcvd

着信データといっしょに受信した状況情報。 **rtn_status** が AP_NO にセットされている場合は、このフィールドは、常に次のリストの最初の部分にある値を含みます。 **rtn_status** が AP_YES にセットされている場合は、このフィールドにリストからのすべての値を含みます。

AP_NONE

AP_CONFIRM_DEALLOCATE

AP_CONFIRM_SEND

AP_CONFIRM_WHAT_RECEIVED

AP_DATA

AP_DATA_COMPLETE

AP_DATA_INCOMPLETE

AP_SEND 

AP_USER_CONTROL_DATA_COMPLETE 

AP_USER_CONTROL_DATA_INCMP 

AP_PS_HEADER_COMPLETE 

AP_PS_HEADER_INCOMPLETE 

AP_DATA_CONFIRM 

AP_DATA_COMPLETE_CONFIRM

AP_DATA_CONFIRM_DEALLOCATE

AP_DATA_COMPLETE_CONFIRM_DEALL

AP_DATA_CONFIRM_SEND

AP_DATA_COMPLETE_CONFIRM_SEND

AP_DATA_SEND 

AP_DATA_COMPLETE_SEND

AP_UC_DATA_COMPLETE_CONFIRM 

AP_UC_DATA_COMPLETE_CNFM_DEALL 

AP_UC_DATA_COMPLETE_CNFM_SEND 

AP_UC_DATA_COMPLETE_SEND 

AP_PS_HDR_COMPLETE_CONFIRM 

AP_PS_HDR_COMPLETE_CNFM_DEALL 

AP_PS_HDR_COMPLETE_CNFM_SEND 

AP_PS_HDR_COMPLETE_SEND 

expd_data_rcvd

急送データ受信の標識。

AP_YES

AP_NO

rts_rcvd

送信要求受信の標識。

AP_YES

AP_NO

dlen このパラメーターが使用されるのは、**what_rcvd** パラメーターが、データを受信したことを示している場合だけです。受信したデータのバイト数 (このデータは、**dptr** パラメーターで指定するバッファーに格納されます)。長さが 0 の場合、データを受信しなかったことを示します。

verb が非ブロッキングで完了していない場合は、Communications Serverは次のパラメーターを戻します。

primary_rc

AP_OPERATION_INCOMPLETE

opext AP_BASIC_CONVERSION または AP_MAPPED_CONVERSATION と

AP_NON_BLOCKING (次のものと OR 結合される)
AP_OPERATION_INCOMPLETE_FLAG

パートナー・トランザクション・プログラムにすぐに使用可能なデータがない場合は、Communications Server は次のパラメーターを戻します。

primary_rc
AP_UNSUCCESSFUL

パラメーター・エラーが原因で verb が実行されなかった場合は、Communications Serverは次のパラメーターを戻します。

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_BAD_CONV_ID
AP_BAD_RETURN_STATUS_WITH_DATA
AP_BAD_TP_ID
AP_RCV_IMMD_BAD_FILL 

トランザクション・プログラムがこの verb を発行したときに会話が不適切な状態にあった場合は、Communications Serverは次のパラメーターを戻します。

primary_rc
AP_STATE_CHECK

secondary_rc
AP_RCV_IMMD_BAD_STATE

次に示す 1 次戻りコード (**primary_rc**) およびそれに付随する 2 次戻りコード (**secondary_rc**) が生成される条件については、付録 A. APPC 共通戻りコードで説明します。

AP_ALLOCATION_ERROR
AP_SECURITY_NOT_VALID
AP_TRANS_PGM_NOT_AVAIL_RETRY
AP_TRANS_PGM_NOT_AVAIL_NO_RTRY
AP_TP_NAME_NOT_RECOGNIZED
AP_PIP_NOT_ALLOWED
AP_PIP_NOT_SPECIFIED_CORRECTLY
AP_CONVERSATION_TYPE_MISMATCH
AP_SYNC_LEVEL_NOT_SUPPORTED

AP_CONV_FAILURE_NO_RETRY

AP_CONV_FAILURE_RETRY

AP_DEALLOC_ABEND 

AP_DEALLOC_ABEND_PROG

AP_DEALLOC_ABEND_SVC

AP_DEALLOC_ABEND_TIMER

AP_DEALLOC_NORMAL

AP_PROG_ERROR_NO_TRUNC

AP_PROG_ERROR_PURGING

AP_PROG_ERROR_TRUNC 

AP_SVC_ERROR_NO_TRUNC 

AP_SVC_ERROR_PURGING 

AP_SVC_ERROR_TRUNC 

AP_TP_BUSY

AP_CONVERSATION_TYPE_MIXED

AP_UNEXPECTED_SYSTEM_ERROR

AP_DUPLEX_TYPE_MIXED

AP_CANCELLED

[MC_]REQUEST_TO_SEND

[MC_]REQUEST_TO_SEND verb は、ローカル・トランザクション・プログラムがデータを送信したいことを、パートナー・トランザクション・プログラムに知らせます。

VCB 構造:

```
typedef struct request_to_send
{
 unsigned short    opcode;             /* verb operation code     */
 unsigned char     opext;              /* verb extension code     */
 unsigned char     format;;            /* format                  */
 unsigned short    primary_rc;         /* primary return code     */
 unsigned long     secondary_rc;       /* secondary return code   */
 unsigned char     tp_id[8];           /* TP identifier           */
 unsigned long     conv_id;            /* conversation identifier */
} REQUEST_TO_SEND;

typedef struct mc_request_to_send
{
 unsigned short    opcode;             /* verb operation code     */
 unsigned char     opext;              /* verb extension code     */
 unsigned char     format;             /* format                  */
 unsigned short    primary_rc;         /* primary return code     */
 unsigned long     secondary_rc;       /* secondary return code   */
 unsigned char     tp_id[8];           /* TP identifier           */
 unsigned long     conv_id;            /* conversation identifier */
} MC_REQUEST_TO_SEND;
```

指定パラメーター: トランザクション・プログラムは、次のパラメーターを Communications Server に提供します。

opcode

AP_B_REQUEST_TO_SEND 

AP_M_REQUEST_TO_SEND 

opext AP_BASIC_CONVERSATION または AP_MAPPED_CONVERSATION。非ブロッキング操作の場合は、このフラグと AP_NON_BLOCKING を OR で結ぶことができます。

format

VCB のフォーマットを識別します。 上記に示した VCB のバージョンを指定するには、このフィールドに 0 をセットしてください。

tp_id ローカル・トランザクション・プログラムの識別子。

このパラメーターの値は、ローカルで始動したトランザクション・プログラムでは **TP_STARTED** verb から戻された値であり、リモートから始動されたトランザクション・プログラムでは **RECEIVE_ALLOCATE** verb から戻された値です。

conv_id
会話識別子。

このパラメーターの値は、ローカルで始動したトランザクション・プログラムでは **[MC_]ALLOCATE** verb から戻された値であり、リモートから始動されたトランザクション・プログラムでは **RECEIVE_ALLOCATE** verb から戻された値です。

戻りパラメーター: verb が正常に実行された場合は、Communications Server は次のパラメーターを戻します。

primary_rc
AP_OK

verb が非ブロッキングで完了していない場合は、Communications Server は次のパラメーターを戻します。

primary_rc
AP_OPERATION_INCOMPLETE

opext AP_OPERATION_INCOMPLETE_FLAG

[MC_]REQUEST_TO_SEND を非ブロッキング・モード（45ページの『待ち行列レベルの非ブロッキング』を参照）で発行し、送受信待ち行列上の verb の処理中に会話が終了した場合は、Communications Server は次のパラメーターを戻します。

primary_rc
AP_CONVERSATION_ENDED

この会話では、アプリケーションがこれ以上 verb を発行しないようにする必要があります。

パラメーター・エラーが原因で verb が実行されなかった場合は、Communications Serverは次のパラメーターを戻します。

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_BAD_CONV_ID

AP_BAD_TP_ID
AP_R_T_S_INVALID_FOR_FDX

トランザクション・プログラムがこの verb を発行したときに会話が不適切な状態にあった場合は、Communications Serverは次のパラメーターを戻します。

primary_rc
AP_STATE_CHECK

secondary_rc
AP_R_T_S_BAD_STATE

MC_REQUEST_TO_SEND

次に示す 1 次戻りコード (**primary_rc**) が生成される条件については、付録 A. APPC 共通戻りコードに説明してあります。

AP_TP_BUSY
AP_CONVERSATION_TYPE_MIXED
AP_UNEXPECTED_SYSTEM_ERROR
AP_CANCELLED

[MC_]SEND_CONVERSATION

[MC_]SEND_CONVERSATION verb は、ローカル LU とパートナー LU との間のセッションに会話を割り振り（その結果パートナー LU のトランザクション・プログラムが開始される）、この会話で単一のデータ・レコードを送信し、確認を待たずに会話の割振りを解除します。 これは、**[MC_]ALLOCATE**、**[MC_]SEND_DATA**、および **[MC_]DEALLOCATE** (**FLUSH**) の一連の verb を順に発行するのと同じ働きをします（一般に『片方向ブラケット』と呼ばれます）。

VCB 構造:

```
typedef struct send_conversation
{
 unsigned short    opcode;                /* verb operation code           */
 unsigned char     opext;                 /* verb extension code           */
 unsigned char     format;                /* format                        */
 unsigned short    primary_rc             /* primary return code           */
 unsigned long     secondary_rc;          /* secondary return code         */
 unsigned char     tp_id[8];              /* TP identifier                 */
 unsigned char     reserv3[8];            /* reserved                      */
 unsigned char     rtn_ctl;               /* return control                */
 unsigned char     reserv4;               /* reserved                      */
 unsigned long     conv_group_id;         /* conversation group identifier */
 unsigned long     sense_data;            /* sense data                    */
 unsigned char     plu_alias[8];          /* partner LU alias              */
 unsigned char     mode_name[8];          /* mode name                     */
 unsigned char     tp_name[64];           /* TP name                       */
 unsigned char     security;              /* security                      */
 unsigned char     reserv5[11];           /* reserved                      */
 unsigned char     pwd[10];               /* security password             */
 unsigned char     user_id[10];           /* security user_id              */
 unsigned short    pip_dlen;              /* PIP data length               */
 unsigned char    *pip_dptr;              /* pointer to PIP data           */
 unsigned char     reserv5a;              /* reserved                      */
 unsigned char     fqplu_name[17];        /* fully qualified partner LU    */
                                          /* name                          */
 unsigned char     reserv6[8];            /* reserved                      */
 unsigned short    dlen;                  /* data length                   */
 unsigned char    *dptr;                  /* pointer to data buffer        */
} SEND_CONVERSATION;

typedef struct mc_send_conversation
{
 unsigned short    opcode;                /* verb operation code           */
 unsigned char     opext;                 /* verb extension code           */
 unsigned char     format;                /* format                        */
 unsigned short    primary_rc;            /* primary return code           */
 unsigned long     secondary_rc;          /* secondary return code         */
 unsigned char     tp_id[8];              /* TP identifier                 */
 unsigned char     reserv3[8];            /* reserved                      */
 unsigned char     rtn_ctl;               /* return control                */
 unsigned char     reserv4;               /* reserved                      */
 unsigned long     conv_group_id;         /* conversation group identifier */
 unsigned long     sense_data;            /* sense data                    */
```

```
 unsigned char     plu_alias[8];          /* partner LU alias             */
 unsigned char     mode_name[8];          /* mode name                    */
 unsigned char     tp_name[64];           /* TP name                      */
 unsigned char     security;              /* security                     */
 unsigned char     reserv6[11];           /* reserved                     */
 unsigned char     pwd[10];               /* security password            */
 unsigned char     user_id[10];           /* security user_id             */
 unsigned short    pip_dlen;              /* PIP data length              */
 unsigned char    *pip_dptr;              /* pointer to PIP data          */
 unsigned char     reserv6a;              /* reserved                     */
 unsigned char     fqplu_name[17];        /* fully qualified partner LU   */
                                          /* name                         */
 unsigned char     reserv7[8];            /* reserved                     */
 unsigned short    dlen;                  /* data length                  */
 unsigned char    *dptr;                  /* pointer to data buffer       */
} MC_SEND_CONVERSATION;
```

指定パラメーター: トランザクション・プログラムは、次のパラメーターを Communications Server に提供します。

opcode

AP_B_SEND_CONVERSATION 

AP_M_SEND_CONVERSATION 

opext AP_BASIC_CONVERSATION または AP_MAPPED_CONVERSATION。非ブロッキング操作の場合は、このフラグと AP_NON_BLOCKING を OR で結ぶことができます。

format

VCB のフォーマットを識別します。 上記に示した VCB のバージョンを指定するには、このフィールドに 0 をセットしてください。

tp_id ローカル・トランザクション・プログラムの識別子。

このパラメーターの値は、ローカルで始動したトランザクション・プログラムでは **TP_STARTED** verb から戻された値であり、リモートから始動されたトランザクション・プログラムでは **RECEIVE_ALLOCATE** verb から戻された値です。

rtn_ctl

ローカル・トランザクション・プログラムからのセッション要求を処理するローカル LU が、ローカル・トランザクション・プログラムにいつ制御を戻すかを指定します。

AP_IMMEDIATE
AP_WHEN_SESSION_ALLOCATED
AP_WHEN_SESSION_FREE

AP_WHEN_CONV_GROUP_ALLOC
AP_WHEN_CONWINNER_ALLOC
AP_WHEN_CONLOSER_ALLOC

conv_group_id
割り振るセッションの会話グループ識別子。 このパラメーターが提供されるのは、**rtn_ctl** を AP_WHEN_CONV_GROUP_ALLOC にセットした場合だけです。

plu_alias
ローカル・トランザクション・プログラムにパートナー LU を認識させるための別名。これは、8 バイトの ASCII 文字ストリングです。 8 バイトのすべてが意味を持つため、すべてのバイトをセットする必要があります。 この名前は、構成時に確立されたパートナー LU の名前に一致している必要があります。

このフィールドをすべて 0 にセットした場合は、Communications Server は **fqplu_name** フィールドを使用して、必要なパートナー LU を指定します。

mode_name
構成時に定義したネットワーキング特性のセットの名前。これは、8 バイトの英数字のタイプ A EBCDIC ストリング（英字で始まるもの）で、8 バイトに満たない場合は右側に EBCDIC のスペースが埋め込まれます。

tp_name
呼び出されたトランザクション・プログラムの名前。Communications Serverはこのフィールドの文字セットをチェックしません。 ローカルのトランザクション・プログラムで **ALLOCATE** verb に指定されている **tp_name** の値は、パートナーのトランザクション・プログラムで **RECEIVE_ALLOCATE** verb に指定されている **tp_name** の値と一致している必要があります。

security
パートナー・トランザクション・プログラムへのアクセスの妥当性をチェックするために、パートナー LU が必要とする情報を指定します。

AP_NONE
AP_PGM
AP_SAME
AP_PGM_STRONG

pwd **user_id** に関連付けられているパスワード。 これは 10 バイトのタイプ AE EBCDIC 文字ストリングで、10 バイトに満たない場合は右側に EBCDIC のスペースが埋め込まれます。 これは Security = Program (AP_PGM または AP_PGM_STRONG) の場合は必須で、その他の場合は任意選択です。

user_id
パートナー・トランザクション・プログラムにアクセスするために必要なユーザー ID。 これは 10 バイトのタイプ AE EBCDIC 文字ストリングで、10 バ

イトに満たない場合は右側に EBCDIC のスペースが埋め込まれます。 これは Security = Program (AP_PGM または AP_PGM_STRONG) の場合は必須で、その他の場合は任意選択です。

pip_dlen
パートナー・トランザクション・プログラムに渡すプログラム初期設定パラメーター (PIP) の長さ。

有効範囲は 0 から 32767 です。

pip_dptr
PIP データが入っているバッファーのアドレス。 このパラメーターは、**pip_dlen** が 0 より大きい場合にのみ使用します。

fqplu_name
パートナー LU の完全修飾 LU 名。 この名前は長さが 17 バイトで、17 バイトに満たない場合は右側に EBCDIC のブランクが埋め込まれます。 この名前は、EBCDIC のピリオドで連結された 2 つのタイプ A EBCDIC 文字ストリングです （1 つの名前の長さは最大 8 バイトで、ブランクを含んでいてはなりません。 ネットワーク ID がない場合は、ピリオドを省略してください)。 このフィールドが意味を持つのは、**plu_alias** フィールドをすべてゼロにセットした場合だけです。

dlen 送信するデータのバイト数。

有効範囲は 0 から 65535 です。

dptr 送信するデータが入っているバッファーのアドレス。 アプリケーションは VCB の末尾にデータを付加できますが、その場合は **dptr** を NULL にセットする必要があります。

戻りパラメーター: verb が正常に実行された場合は、Communications Serverは次のパラメーターを戻します。

primary_rc
AP_OK

conv_group_id
会話に割り振られるセッションの会話グループ識別子。

verb が非ブロッキングで完了していない場合は、Communications Server は次のパラメーターを戻します。

primary_rc
AP_OPERATION_INCOMPLETE

opext AP_OPERATION_INCOMPLETE_FLAG

rtn_ctl パラメーターが AP_IMMEDIATE にセットしてあるときに、すぐに使用できるセッションがない場合は、Communications Server は次のパラメーターを戻します。

primary_rc
AP_UNSUCCESSFUL

パラメーター・エラーが原因で verb が実行されなかった場合は、Communications Serverは次のパラメーターを戻します。

primary_rc

AP_PARAMETER_CHECK

secondary_rc

AP_BAD_TP_ID

AP_BAD_LL 

AP_BAD_RETURN_CONTROL

AP_BAD_SECURITY

AP_PIP_LEN_INCORRECT

AP_NO_USE_OF_SNASVCMG 

AP_UNKNOWN_PARTNER_MODE

次に示す 1 次戻りコード (**primary_rc**) およびそれに付随する 2 次戻りコード (**secondary_rc**) が生成される条件については、付録 A. APPC 共通戻りコードで説明します。

AP_UNSUCCESSFUL 

AP_ALLOCATION_ERROR

AP_ALLOCATION_FAILURE_NO_RETRY

AP_ALLOCATION_FAILURE_RETRY

AP_SEC_REQUESTED_NOT_SUPPORTED 

AP_TP_BUSY

AP_CONVERSATION_TYPE_MIXED 

AP_UNEXPECTED_SYSTEM_ERROR

AP_CANCELLED

primary_rc が AP_ALLOCATION_ERROR の場合は、**sense_data** フィールドには障害に関するより詳細な情報が含まれています。

[MC_]SEND_DATA

[MC_]SEND_DATA verb は、パートナー・トランザクション・プログラムに送信するデータを、ローカル LU の送信バッファーに蓄えます。

VCB 構造:

```
typedef struct send_data
{
 unsigned short    opcode;               /* verb operation code      */
 unsigned char     opext;                /* verb extension code      */
 unsigned char     format;               /* format                   */
 unsigned short    primary_rc;           /* primary return code      */
 unsigned long     secondary_rc;         /* secondary return code    */
 unsigned char     tp_id[8];             /* TP identifier            */
 unsigned long     conv_id;              /* conversation identifier  */
 unsigned char     rts_rcvd;             /* request to send received */
 unsigned char     expd_data_rcvd;       /* expedited data received  */
 unsigned short    dlen;                 /* data length              */
 unsigned char    *dptr;                 /* pointer to data          */
 unsigned char     type;                 /* send data type           */
 unsigned char     reserv4;              /* reserved                 */
} SEND_DATA;

typedef struct mc_send_data
{
 unsigned short    opcode;             /* verb operation code      */
 unsigned char     opext;              /* verb extension code      */
 unsigned char     format;             /* format                   */
 unsigned short    primary_rc;         /* primary return code      */
 unsigned long     secondary_rc;       /* secondary return code    */
 unsigned char     tp_id[8];           /* TP identifier            */
 unsigned long     conv_id;            /* conversation identifier  */
 unsigned char     rts_rcvd;           /* request to send received */
#ifdef WINAPPC_FORMAT_1
 unsigned char     expd_data_rcvd;     /* expedited data received  */
#else
 unsigned char     data_type;          /* data type received       */
#endif
 unsigned short    dlen;               /* data length              */
 unsigned char    *dptr;               /* pointer to data          */
 unsigned char     type;               /* send data type           */
#ifdef WINAPPC_FORMAT_1
 unsigned char     data_type;          /* data type received       */
#else
 unsigned char     reserv4;            /* reserved                 */
#endif
} MC_SEND_DATA;
```

指定パラメーター: トランザクション・プログラムは、次のパラメーターを Communications Server に提供します。

opcode

AP_B_SEND_DATA

AP_M_SEND_DATA 

opext AP_BASIC_CONVERSATION または AP_MAPPED_CONVERSATION。非ブロッキング操作の場合は、このフラグと AP_NON_BLOCKING を OR で結ぶことができます。

全二重会話では、このフラグと AP_FULL_DUPLEX_CONVERSATION をOR で結ぶ必要があります。

format

VCB のフォーマット。 前述したフォーマットを得るためには、これを 1 に設定してください。

tp_id ローカル・トランザクション・プログラムの識別子。

このパラメーターの値は、ローカルで始動したトランザクション・プログラムでは **TP_STARTED** verb から戻された値であり、リモートから始動されたトランザクション・プログラムでは **RECEIVE_ALLOCATE** verb から戻された値です。

conv_id

会話識別子。

このパラメーターの値は、ローカルで始動したトランザクション・プログラムでは **[MC_]ALLOCATE** verb から戻された値であり、リモートから始動されたトランザクション・プログラムでは **RECEIVE_ALLOCATE** verb から戻された値です。

dlen 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。

有効範囲は 0 から 65535 です。

dptr 送信するデータが入っているバッファーのアドレス。 アプリケーションは VCB の末尾にデータを付加できますが、その場合は **dptr** を NULL にセットする必要があります。

type **SEND_DATA** に加えて他の verb の機能も実行するかどうかを指定します。

AP_NONE
AP_SEND_DATA_CONFIRM
AP_SEND_DATA_FLUSH
AP_SEND_DATA_P_TO_R_FLUSH
AP_SEND_DATA_P_TO_R_SYNC_LEVEL
AP_SEND_DATA_P_TO_R_CONFIRM
AP_SEND_DATA_DEALLOC_FLUSH
AP_SEND_DATA_DEALLOC_SYNC_LEVEL
AP_SEND_DATA_DEALLOC_CONFIRM
AP_SEND_DATA_DEALLOC_ABEND

戻りパラメーター: verb が正常に実行された場合は、Communications Serverは次のパラメーターを戻します。

primary_rc
AP_OK

rts_rcvd
送信要求受信の標識。

AP_YES

AP_NO

expd_data_rcvd
急送データ受信の標識。 この標識は、RECEIVE_EXPEDITED_DATA が発行されるまで、AP_YES にセットされたままです。

AP_YES

AP_NO

パラメーター・エラーのために verb が実行されない場合は、Communications Serverは次のパラメーターを戻します。

primary_rc
AP_PARAMETER_CHECK

opext AP_OPERATION_INCOMPLETE_FLAG

パラメーター・エラーが原因で verb が実行されなかった場合は、Communications Serverは次のパラメーターを戻します。

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_BAD_CONV_ID

AP_BAD_TP_ID

AP_BAD_LL 

AP_SEND_DATA_INVALID_TYPE

AP_SEND_DATA_CONFIRM_SYNC_NONE

AP_SEND_TYPE_INVALID_FOR_FDX

トランザクション・プログラムがこの verb を発行したときに会話が不適切な状態にあった場合は、Communications Serverは次のパラメーターを戻します。

primary_rc
AP_STATE_CHECK

secondary_rc
AP_SEND_DATA_NOT_SEND_STATE

AP_SEND_DATA_NOT_LL_BDY 

次に示す 1 次戻りコード (**primary_rc**) およびそれに付随する 2 次戻りコード (**secondary_rc**) が生成される条件については、付録 A. APPC 共通戻りコードで説明します。

AP_ALLOCATION_ERROR

- AP_SECURITY_NOT_VALID
- AP_TRANS_PGM_NOT_AVAIL_RETRY
- AP_TRANS_PGM_NOT_AVAIL_NO_RTRY
- AP_TP_NAME_NOT_RECOGNIZED
- AP_PIP_NOT_ALLOWED
- AP_PIP_NOT_SPECIFIED_CORRECTLY
- AP_CONVERSATION_TYPE_MISMATCH
- AP_SYNC_LEVEL_NOT_SUPPORTED
- AP_CONV_FAILURE_NO_RETRY
- AP_CONV_FAILURE_RETRY
- AP_DEALLOC_ABEND 
- AP_DEALLOC_ABEND_PROG 
- AP_DEALLOC_ABEND_SVC 
- AP_DEALLOC_ABEND_TIMER 
- AP_PROG_ERROR_PURGING
- AP_SVC_ERROR_PURGING 
- AP_TP_BUSY
- AP_CONVERSATION_TYPE_MIXED
- AP_DUPLEX_TYPE_MIXED 
- AP_UNEXPECTED_SYSTEM_ERROR
- AP_CANCELLED
- AP_ERROR_INDICATION
- AP_ALLOCATION_ERROR_PENDING
- AP_DEALLOC_ABEND_PROG_PENDING
- AP_DEALLOC_ABEND_SVC_PENDING
- AP_DEALLOC_ABEND_TIMER_PENDING
- AP_UNKNOWN_ERROR_TYPE_PENDING

[MC_]SEND_ERROR

[MC_]SEND_ERROR verb は、ローカル・トランザクション・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・トランザクション・プログラムに通知します。

VCB 構造:

```
typedef struct send_error
{
 unsigned short    opcode;              /* verb operation code      */
 unsigned char     opext;               /* verb extension code      */
 unsigned char     format;              /* format                   */
 unsigned short    primary_rc;          /* primary return code      */
 unsigned long     secondary_rc;        /* secondary return code    */
 unsigned char     tp_id[8];            /* TP identifier            */
 unsigned long     conv_id;             /* conversation identifier  */
 unsigned char     rts_rcvd;            /* request to send received */
 unsigned char     err_type;            /* error type               */
 unsigned char     err_dir;             /* error direction          */
 unsigned char     expd_data_rcvd;      /* expedited data received  */
 unsigned short    log_dlen;            /* log data length          */
 unsigned char    *log_dptr;            /* pointer to log data      */
} SEND_ERROR;

typedef struct mc_send_error
{
 unsigned short    opcode;              /* verb operation code      */
 unsigned char     opext;               /* verb extension code      */
 unsigned char     format;              /* format                   */
 unsigned short    primary_rc;          /* primary return code      */
 unsigned long     secondary_rc;        /* secondary return code    */
 unsigned char     tp_id[8];            /* TP identifier            */
 unsigned long     conv_id;             /* conversation identifier  */
 unsigned char     rts_rcvd;            /* request to send received */
 unsigned char     err_type;            /* error type               */
 unsigned char     err_dir;             /* error direction          */
 unsigned char     expd_data_rcvd;      /* expedited data received  */
 unsigned char     reserv5[2];          /* reserved                 */
 unsigned char     reserv6[4];          /* reserved                 */
} MC_SEND_ERROR;
```

指定パラメーター: トランザクション・プログラムは、次のパラメーターを Communications Server に提供します。

opcode

AP_B_SEND_ERROR 

AP_M_SEND_ERROR 

opext AP_BASIC_CONVERSATION または AP_MAPPED_CONVERSATION。非ブロッキング操作の場合は、このフラグと AP_NON_BLOCKING を OR で結ぶことができます。

全二重会話では、このフラグと AP_FULL_DUPLEX_CONVERSATION をOR で結ぶ必要があります。

format

VCB のフォーマットを識別します。 上記に示した VCB のバージョンを指定するには、このフィールドに 0 をセットしてください。

tp_id ローカル・トランザクション・プログラムの識別子。

このパラメーターの値は、ローカルで始動したトランザクション・プログラムでは **TP_STARTED** verb から戻された値であり、リモートから始動されたトランザクション・プログラムでは **RECEIVE_ALLOCATE** verb から戻された値です。

conv_id

会話識別子。

このパラメーターの値は、ローカルで始動したトランザクション・プログラムでは **[MC_]ALLOCATE** verb から戻された値であり、リモートから始動されたトランザクション・プログラムでは **RECEIVE_ALLOCATE** verb から戻された値です。

err_type 

報告するエラーのタイプ（アプリケーション・プログラムまたはサービス・プログラム）を示します。

AP_PROGAP_SVC

err_dir

報告するエラーが、パートナー・トランザクション・プログラムから受信したデータに関するものなのか、ローカル・トランザクション・プログラムが送信しようとしたデータに関するものなのかを示します。

このパラメーターが使用されるのは、**SEND_ERROR** verb が SEND_PENDING 状態で発行される場合だけです。

AP_RCV_DIR_ERROR

AP_SEND_DIR_ERROR

log_dlen 

エラー・ログ・ファイルに送られるデータのバイト数。

有効範囲は 0 から 32767 です。

アプリケーションは VCB の末尾にデータを付加できますが、その場合はこのフィールドは 0 より大きくなり、**log_dptr** を NULL にセットする必要があります （長さが 0 の場合、エラー・ログ・データがないことを意味します)。

log_dptr 

エラー情報が入っているデータ・バッファーのアドレス。アプリケーションは VCB の末尾にデータを付加できますが、その場合は **log_dptr** を NULL にセットする必要があります。

このデータは、ローカル・エラー・ログおよびパートナー LU に送られます。**SEND_ERROR** verb でこのパラメーターが使用されるのは、**log_dlen** がゼロより大きい場合です。

トランザクション・プログラムは、エラー・データを汎用データ・ストリーム (GDS) として形式設定する必要があります。 詳細は、*IBM Systems Network Architecture: LU 6.2 Reference: Peer Protocols* を参照してください。

戻りパラメーター: verb が正常に実行された場合は、Communications Serverは次のパラメーターを戻します。

primary_rc
AP_OK

rts_rcvd
送信要求受信の標識。

AP_YES

AP_NO

expd_data_rcvd
急送データ受信の標識。 この標識は、RECEIVE_EXPEDITED_DATA が発行されるまで、AP_YES にセットされたままです。

AP_YES

AP_NO

verb が非ブロッキングで完了していない場合は、Communications Server は次のパラメーターを戻します。

primary_rc
AP_OPERATION_INCOMPLETE

opext AP_OPERATION_INCOMPLETE_FLAG

パラメーター・エラーが原因で verb が実行されなかった場合は、Communications Serverは次のパラメーターを戻します。

primary_rc
AP_PARAMETER_CHECK

secondary_rc

AP_BAD_CONV_ID

AP_BAD_ERROR_DIRECTION

AP_BAD_TP_ID

AP_SEND_ERROR_BAD_TYPE 

AP_SEND_ERROR_LOG_LL_WRONG 

トランザクション・プログラムがこの verb を発行したときに会話が不適切な状態にあった場合は、Communications Serverは次のパラメーターを戻します。

primary_rc

AP_STATE_CHECK

secondary_rc

AP_SEND_ERROR_BAD_STATE

次に示す 1 次戻りコード (**primary_rc**) およびそれに付随する 2 次戻りコード (**secondary_rc**) が生成される条件については、付録 A. APPC 共通戻りコードで説明します。

verb 発行が許可されている状態で verb が発行された場合: verb 発行が許可されている状態で **[MC_]SEND_ERROR** verb が発行された場合は、次の戻りコードが生成されます。

AP_CONV_FAILURE_NO_RETRY
AP_CONV_FAILURE_RETRY
AP_TP_BUSY
AP_CONVERSATION_TYPE_MIXED
AP_DUPLEX_TYPE_MIXED
AP_UNEXPECTED_SYSTEM_ERROR
AP_CANCELLED
AP_ERROR_INDICATION

AP_ALLOCATION_ERROR_PENDING

AP_DEALLOC_ABEND_PROG_PENDING

AP_DEALLOC_ABEND_SVC_PENDING

AP_DEALLOC_ABEND_TIMER_PENDING

AP_UNKNOWN_ERROR_TYPE_PENDING

SEND 状態で verb が発行された場合:
次の戻りコードが生成されるのは、**[MC_]SEND_ERROR** verb が SEND 状態で発行された場合だけです。

AP_ALLOCATION_ERROR

- AP_SECURITY_NOT_VALID
- AP_TRANS_PGM_NOT_AVAIL_RETRY
- AP_TRANS_PGM_NOT_AVAIL_NO_RTRY
- AP_TP_NAME_NOT_RECOGNIZED
- AP_PIP_NOT_ALLOWED
- AP_PIP_NOT_SPECIFIED_CORRECTLY
- AP_CONVERSATION_TYPE_MISMATCH
- AP_SYNC_LEVEL_NOT_SUPPORTED
- AP_DEALLOC_ABEND 
- AP_DEALLOC_ABEND_PROG 
- AP_DEALLOC_ABEND_SVC 
- AP_DEALLOC_ABEND_TIMER 
- AP_PROG_ERROR_PURGING
- AP_SVC_ERROR_PURGING 

RECEIVE 状態で verb が発行された場合: 次の戻りコードが生成されるのは、verb が RECEIVE 状態で発行された場合だけです。

AP_DEALLOC_NORMAL

[MC_]SEND_EXPEDITED_DATA

これは、サーバーにのっているアプリケーションに対してだけサポートされます。

[MC_]SEND_EXPEDITED_DATA verb は、パートナー・トランザクション・プログラムに送信するために、データをローカル LU の急送送信バッファーに蓄えます。このデータは、早期に送られた非急送データの前に、パートナー・トランザクション・プログラムに到着する可能性があります。

VCB 構造:

```
typedef struct send_expedited_data
{
 unsigned short    opcode;             /* verb operation code     */
 unsigned char     opext;              /* verb extension code     */
 unsigned char     format;             /* format                  */
 unsigned short    primary_rc;         /* primary return code     */
 unsigned long     secondary_rc;       /* secondary return code   */
 unsigned char     tp_id[8];           /* TP identifier           */
 unsigned long     conv_id;            /* conversation identifier  */
 unsigned char     rts_rcvd;           /* request to send received */
 unsigned char     expd_data_rcvd;     /* expedited data received  */
 unsigned short    dlen;               /* data length              */
 unsigned char    *dptr;               /* pointer to data          */
 unsigned char     reserve4[2];        /* TP identifier           */
} SEND_EXPEDITED_DATA;

typedef struct mc_send_expedited_data
{
 unsigned short    opcode;               /* verb operation code       */
 unsigned char     opext;                /* verb extension code       */
 unsigned char     format;               /* format                    */
 unsigned short    primary_rc;           /* primary return code       */
 unsigned long     secondary_rc;         /* secondary return code     */
 unsigned char     tp_id[8];             /* TP identifier             */
 unsigned long     conv_id;              /* conversation identifier   */
 unsigned char     rts_rcvd;             /* request to send received  */
 unsigned char     expd_data_rcvd;       /* expedited data received   */
                                         /* transaction plan          */
                                         /* data                      */
 unsigned short    dlen;                 /* actual length of received */
                                         /* data                      */
 unsigned char    *dptr;                 /* pointer to data buffer    */
 unsigned char     reserv4[2];           /* reserved                  */
} MC_SEND_EXPEDITED_DATA
```

指定パラメーター: トランザクション・プログラムは、次のパラメーターを Communications Server に提供します。

opcode

AP_B_SEND_EXPEDITED_DATA 

AP_M_SEND_EXPEDITED_DATA 

opext AP_BASIC_CONVERSATION または AP_MAPPED_CONVERSATION。非ブロッキング操作の場合は、このフラグと AP_NON_BLOCKING を OR で結ぶことができます。

全二重会話では、このフラグと AP_FULL_DUPLEX_CONVERSATION をORで結ぶ必要があります。

format

VCB のフォーマットを識別します。 上記に示した VCB のバージョンを指定するには、このフィールドに 0 をセットしてください。

tp_id ローカル・トランザクション・プログラムの識別子。

このパラメーターの値は、ローカルで始動したトランザクション・プログラムでは **TP_STARTED** verb から戻された値であり、リモートから始動されたトランザクション・プログラムでは **RECEIVE_ALLOCATE** verb から戻された値です。

conv_id

会話識別子。

このパラメーターの値は、ローカルで始動したトランザクション・プログラムでは **[MC_]ALLOCATE** verb から戻された値であり、リモートから始動されたトランザクション・プログラムでは **RECEIVE_ALLOCATE** verb から戻された値です。

dlen 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。

有効範囲 は 1 から 86 です。

dptr エラー情報が入っているデータ・バッファーのアドレス。 アプリケーションは VCB の末尾にデータを付加できますが、その場合は **dptr** を NULL にセットする必要があります。

データはフォーマットされていないことに注意してください。 2 バイト長フィールド (LL) は存在しません。

戻りパラメーター: verb が正常に実行された場合は、Communications Serverは次のパラメーターを戻します。

primary_rc

AP_OK

rts_rcvd

送信要求受信の標識。

AP_YES

AP_NO

expd_data_rcvd

急送データ受信の標識。 この標識は、RECEIVE_EXPEDITED_DATA が発行されるまで、AP_YES にセットされたままです。

AP_YES

AP_NO

verb が非ブロッキングで完了していない場合は、Communications Server は次のパラメーターを戻します。

primary_rc

AP_OPERATION_INCOMPLETE

opext AP_OPERATION_INCOMPLETE_FLAG

リモート LU が急送データをサポートしていないために verb が実行されない場合は、Communications Server は次のパラメーターを戻します。

primary_rc

AP_EXPD_NOT_SUPPORTED_BY_LU

パラメーター・エラーが原因で verb が実行されなかった場合は、Communications Serverは次のパラメーターを戻します。

primary_rc

AP_PARAMETER_CHECK

secondary_rc

AP_BAD_CONV_ID

AP_BAD_TP_ID

AP_SEND_EXPD_INVALID_LENGTH

AP_RCV_EXPD_INVALID_LENGTH 

トランザクション・プログラムがこの verb を発行したときに会話が不適切な状態にあった場合は、Communications Serverは次のパラメーターを戻します。

primary_rc

AP_STATE_CHECK

secondary_rc

AP_EXPD_DATA_BAD_CONV_STATE

次に示す 1 次戻りコード (**primary_rc**) およびそれに付随する 2 次戻りコード (**secondary_rc**) が生成される条件については、付録 A. APPC 共通戻りコードで説明します。

AP_ALLOCATION_ERROR

AP_SECURITY_NOT_VALID

AP_TRANS_PGM_NOT_AVAIL_RETRY

AP_TRANS_PGM_NOT_AVAIL_NO_RTRY

AP_TP_NAME_NOT_RECOGNIZED

AP_PIP_NOT_ALLOWED

AP_PIP_NOT_SPECIFIED_CORRECTLY
AP_CONVERSATION_TYPE_MISMATCH
AP_SYNC_LEVEL_NOT_SUPPORTED

AP_CONV_FAILURE_NO_RETRY
AP_CONV_FAILURE_RETRY
AP_DEALLOC_ABEND_PROG
AP_DEALLOC_ABEND_SVC
AP_DEALLOC_ABEND_TIMER
AP_TP_BUSY
AP_CONVERSATION_TYPE_MIXED
AP_DUPLEX_TYPE_MIXED
AP_UNEXPECTED_SYSTEM_ERROR
AP_CANCELLED

[MC_]TEST_RTS

[MC_]TEST_RTS verb は、ローカル・トランザクション・プログラムがパートナー・トランザクション・プログラムから送信要求通知 (REQUEST_TO_SEND) を受信したかどうかを判別します。

VCB 構造:

```
typedef struct test_rts
{
 unsigned short    opcode;             /* verb operation code     */
 unsigned char     opext;              /* verb extension code     */
 unsigned char     format;             /* format                  */
 unsigned short    primary_rc;         /* primary return code     */
 unsigned long     secondary_rc;       /* secondary return code   */
 unsigned char     tp_id[8];           /* TP identifier           */
 unsigned long     conv_id;            /* conversation identifier */
 unsigned char     reserv3;            /* reserved                */
} TEST_RTS;

typedef struct mc_test_rts
{
 unsigned short    opcode;               /* verb operation code     */
 unsigned char     opext;                /* verb extension code     */
 unsigned char     format;               /* format                  */
 unsigned short    primary_rc;           /* primary return code     */
 unsigned long     secondary_rc;         /* secondary return code   */
 unsigned char     tp_id[8];             /* TP identifier           */
 unsigned long     conv_id;              /* conversation identifier */
 unsigned char     reserv3;              /* reserved                */
} MC_TEST_RTS;
```

指定パラメーター: トランザクション・プログラムは、次のパラメーターを Communications Server に提供します。

opcode

AP_B_TEST_RTS 

AP_M_TEST_RTS 

opext AP_BASIC_CONVERSATION または AP_MAPPED_CONVERSATION。

format

VCB のフォーマットを識別します。 上記に示した VCB のバージョンを指定するには、このフィールドに 0 をセットしてください。

tp_id ローカル・トランザクション・プログラムの識別子。 このパラメーターの値は、ローカルで始動したトランザクション・プログラムでは **TP_STARTED** verb から戻された値であり、リモートから始動されたトランザクション・プログラムでは **RECEIVE_ALLOCATE** verb から戻された値です。

conv_id

会話識別子。

このパラメーターの値は、ローカルで始動したトランザクション・プログラムでは **[MC_]ALLOCATE** verb から戻された値であり、リモートから始動されたトランザクション・プログラムでは **RECEIVE_ALLOCATE** verb から戻された値です。

戻りパラメーター: verb が正常に実行された場合は、Communications Server は次のパラメーターを戻します。

primary_rc

パートナー・トランザクション・プログラムから送信要求の通知を受信したかどうかを示します。

AP_OK

AP_UNSUCCESSFUL

パラメーター・エラーが原因で verb が実行されなかった場合は、Communications Serverは次のパラメーターを戻します。

primary_rc

AP_PARAMETER_CHECK

secondary_rc

AP_BAD_CONV_ID

AP_BAD_TP_ID
AP_TEST_INVALID_FOR_FDX

次に示す 1 次戻りコード (**primary_rc**) が生成される条件については、付録 A. APPC 共通戻りコードに説明してあります。

AP_TP_BUSY
AP_CONVERSATION_TYPE_MIXED
AP_UNEXPECTED_SYSTEM_ERROR

[MC_]TEST_RTS_AND_POST

[MC_]TEST_RTS_AND_POST verb は、ローカル・トランザクション・プログラムがパートナー・トランザクション・プログラムから送信要求通知 (REQUEST_TO_SEND) を受信したかどうかを、非同期に判別します。 トランザクション・プログラムは、会話上で他の verb が未完了状態にある場合も含めて、いつでも **[MC_]TEST_RTS_AND_POST** を発行することができます。
[MC_]TEST_RTS_AND_POST が戻るのは、送信要求通知を受信したとき、または会話が終了したとき、あるいは会話の障害が検出されたときです。

この verb は、APPC エントリー・ポイントを介してのみ発行できます。

VCB 構造:

```
typedef struct test_rts_and_post
{
 unsigned short    opcode;             /* verb operation code     */
 unsigned char     opext;              /* verb extension code     */
 unsigned char     format;             /* format                  */
 unsigned short    primary_rc;         /* primary return code     */
 unsigned long     secondary_rc;       /* secondary return code   */
 unsigned char     tp_id[8];           /* TP identifier           */
 unsigned long     conv_id;            /* conversation identifier */
 unsigned char     reserv3;            /* reserved                */
 unsigned long     sema;               /* post handle for verb    */
} TEST_RTS_AND_POST;

typedef struct mc_test_rts_and_post
{
 unsigned short    opcode;             /* verb operation code     */
 unsigned char     opext;              /* verb extension code     */
 unsigned char     format;             /* format                  */
 unsigned short    primary_rc;         /* primary return code     */
 unsigned long     secondary_rc;       /* secondary return code   */
 unsigned char     tp_id[8];           /* TP identifier           */
 unsigned long     conv_id;            /* conversation identifier */
 unsigned char     reserv3;            /* reserved                */
 unsigned long     sema;               /* post handle for verb    */
} MC_TEST_RTS_AND_POST;
```

指定パラメーター: トランザクション・プログラムは、次のパラメーターを Communications Server に提供します。

opcode

AP_B_TEST_RTS_AND_POST 

AP_M_TEST_RTS_AND_POST 

opext AP_BASIC_CONVERSATION または AP_MAPPED_CONVERSATION。

format
VCB のフォーマットを識別します。 上記に示した VCB のバージョンを指定するには、このフィールドに 0 をセットしてください。

tp_id ローカル・トランザクション・プログラムの識別子。

このパラメーターの値は、ローカルで始動したトランザクション・プログラムでは **TP_STARTED** verb から戻された値であり、リモートから始動されたトランザクション・プログラムでは **RECEIVE_ALLOCATE** verb から戻された値です。

conv_id
会話識別子。

このパラメーターの値は、ローカルで始動したトランザクション・プログラムでは **[MC_]ALLOCATE** verb から戻された値であり、リモートから始動されたトランザクション・プログラムでは **RECEIVE_ALLOCATE** verb から戻された値です。

sema アプリケーションが待機するイベントのハンドル。 この verb は、Win32 API でWaitForMultipleObjects とともに使用するためのものです。 この機能の詳細については、Win32 API のプログラミング解説書を参照してください。

戻りパラメーター: verb が正常に実行された (すなわち、送信要求の通知が受け取られた)場合は、Communications Serverは次のパラメーターを戻します。

primary_rc
AP_OK

会話が終了したか、または会話障害が検出されたためにこの verb が戻った場合は、Communications Serverは次のパラメーターを戻します。

primary_rc
AP_UNSUCCESSFUL

パラメーター・エラーが原因で verb が実行されなかった場合は、Communications Serverは次のパラメーターを戻します。

primary_rc
AP_PARAMETER_CHECK

secondary_rc
AP_BAD_CONV_ID

AP_BAD_TP_ID
AP_TEST_INVALID_FOR_FDX

次に示す 1 次戻りコード (**primary_rc**) が生成される条件については、付録 A. APPC 共通戻りコードに説明してあります。

AP_CONVERSATION_TYPE_MIXED
AP_UNEXPECTED_SYSTEM_ERROR
AP_CANCELLED

第2部 LUA API

第9章 IBM 従来型 LU アプリケーションの基本概念

この章では、IBM 従来型論理装置アプリケーション（LUA）アクセス方式を紹介し、システム・ネットワーク体系 (SNA) との関係について説明します。

LUA と SNA の概略

IBM LUA アクセス方式は、2 次従属論理装置 (LU) のためのアプリケーション・プログラミング・インターフェース (API) を Communications Server に提供します。LUA はシステム・ソフトウェアとインターフェースから構成され、これらのシステム・ソフトウェアとインターフェースは、LU タイプ 0、1、2、および 3 の SNA プロトコルを使用する通信をサポートするために、入出力 (I/O) サービス・ルーチンを提供します。Communications Server は LUA の RUI インターフェースのみをサポートします。

Communications Server が提供する RUI は、Microsoft** NT SNA Server とのバイナリー互換性を備えたものとして設計されており、OS/2 コミュニケーション・マネージャー/2 バージョン 1.0 LUA の RUI インターフェースに似ています。

LUA がアプリケーション・プログラムに提供するサービスに含まれるのは、データ通信をサポートするサービスだけです。つまり、LUA は装置エミュレーション機能は備えていません。 ただし、LUA は、プレゼンテーション・サービス層の機能に独自のサブセットを提供しています。

ワークステーションで LUA アプリケーション・プログラムを実行できるようにするには、Communications Server を導入し構成しておくことが必要です。 Communications Server の導入と構成については、*Communications Server: 概説およびインストール*参照してください。

接続機能

どのような通信システムの場合も、その主要な目的は他のシステムと接続することにあります。 SNA の最終目標は、広範囲にわたる汎用的な接続を可能にする共通プロトコルを提供することです。 LUA 通信および接続のための必要条件の 1 つとして、System/370* (S/370*) 接続が含まれています。

LUA アプリケーション・プログラム

本書では、*LUA アプリケーション・プログラム* という用語は、LUA 通信機能を使用するアプリケーション・プログラム、またはその一部を意味します。 アプリケーション・プログラムはこの通信機能を使用して、LU タイプ 0 、 1 、 2 、または 3 をサポートする他のシステム上のアプリケーション・プログラムと通信します。

ローカル LUA アプリケーション・プログラムが実行されると、リモート・ホスト・アプリケーション・プログラムの間でデータを交換します。 ローカル・アプリケーション・プログラムとリモート・アプリケーション・プログラムを、*パートナー・アプリケーション・プログラム*と呼びます。

LUA verb

verb は、LUA によって処理される決まった形式を持つ要求です。 アプリケーション・プログラムは、LUA に何らかのアクションを要求するために verb を発行します。 LUA verb は制御ブロックとしてコーディングされます。 各 verb 制御ブロックには、それぞれ厳密に定義されたフォーマットがあります。 LUA の機能を使用するには、アプリケーション・プログラムは、verb 制御ブロックを LUA API に渡します。

LUA verb は、常に即時に呼出し元に戻ります。 戻りコードが IN_PROGRESS である場合は、アプリケーションは、verb 要求の中で指定されている通知方式を使用して、verb が完了するまで待つ必要があります。 LUA verb の通知に関する説明については、第12章 LUA エントリー・ポイントを参照してください。

verb 制御ブロックのレイアウトは、Communications Serverの **INCLUDE** ディレクトリーに収めてあります。 verb 制御ブロックのレイアウトおよびサンプル・プログラムは、LUA アプリケーション・プログラムを書くときの参考として使用できます。

LU、ローカル LU、およびパートナー LU

論理装置 (LU) は、アプリケーション・プログラム間のデータの交換を管理します。 LUA アプリケーション・プログラムは、どれも、LU を介して SNA ネットワークにアクセスします。LU は、LUA アプリケーション・プログラムと SNA ネットワークとの間の仲介役を果たします。

LUA では、LUA アプリケーション・プログラム・プロセスと LU との間には 1 対多の関係があります。 つまり、 1 つの LUA アプリケーション・プログラム・プロセスが同時に複数の LU を所有できますが、特定の 1 つの LU を所有できるのは一度に 1 つの LUA アプリケーション・プログラム・プロセスだけです。 第 2 のアプリケーション・プログラム・プロセスが LU を使用するには、第 1 のアプリケーション・プログラムがその LU を解放していなければなりません。

LUA アプリケーション・プログラムは、ローカル LU に対して LUA verb を発行します。 これらの verb に従って、コマンドおよびデータがネットワークを介してパートナー LU に送られます。

注: ローカル LU は、各マシンについて 1 回だけ定義する必要があります。その方法については、*概説およびインストール*に説明があります。

システム・サービス制御点 (SSCP)

ホスト・システムのシステム・サービス制御点 (SSCP) 構成要素は、ホスト・アプリケーションの開始、ホスト・アプリケーションと従属 LU との関連付け、およびLU 間の接続の作成と終了を担当します。

SNA の層

SNA は、７つの厳密に定義された層からなる階層構造です。 アーキテクチャー内の各層は、それぞれ特定の機能を実行します。 SNA の階層構造を理解することは、LUA が提供する各種の機能を理解する上で役立ちます。 ここでは、LUA と SNA との関係を示す、SNA の５つの上位層について説明します。

データ・リンク制御層

データ・リンク制御 (DLC) 層は、ハードウェアへのインターフェースを提供する要素から成っています。 DLC 要素は、同期データ・リンク制御 (SDLC) および IBM トークンリング・ネットワークなど、各種の DLC プロトコルのためのサポートを提供します。 DLC 層は、パス制御 (PC) 層の要素に対して共通リンクの外観を提供します。 DLC 層は、LUA も含めてすべてのCommunications Server LU 実装に共通です。

経路制御層

周辺ノードでの SNA の経路制御 (PC) 層は、ノード内の複数のハーフセッションとの間での経路指定など、基本的な機能を提供します。SNA では、PC 層は、一度に１つのデータ・リンクとの間でのみ経路指定を行うことができます。PC 層は、LUA も含めてすべてのCommunications Server LU に共通です。

伝送制御層

SNA の伝送制御 (TC) 層は、ローカルでサポートされている各ハーフセッションに対して、結合点マネージャー機能およびセッション制御機能を提供します。結合点マネージャー機能は、順序番号検査、ペーシング、および、ハーフセッションのデータ・フローに関連したその他のサポート機能を制御します。 セッション制御機能は、始動、ペーシング、暗号化、非暗号化、および、セッション関連のデータ・フローに関連したその他のサポート機能について、セッション固有のサポートを提供します。 LUA には、Communications Serverでの LU タイプ０、１、２、および３に対して TC 層を実現する機能が含まれています。

データ・フロー制御層

SNA のデータ・フロー制御 (DFC) 層は、セッション内またはセッション間にある機能管理データ (FMD) のペアの間での、FMD 要求および FMD 応答のフローを制御します。 データ・フロー制御層は、要求/応答形式設定、データ・チェイニング・プロトコル、要求/応答相関、送信および受信モード・プロトコル、ブラケット・プロト

コル、エラー回復プロトコル、ブラケット開始停止プロトコル、および待ち行列応答プロトコルなど、さまざまの機能を提供します。 LUA には、Communications Serverでの LU タイプ 0、1、2、および 3 に対してデータ・フロー制御層を実現する機能が含まれています。

プレゼンテーション・サービス層

SNA のプレゼンテーション・サービス (PS) 層には、通信データ・インターフェースをユーザーに提供する機能があります。 プレゼンテーション・サービス層は、アーキテクチャー内で、LU 0 を除くすべての LU タイプに対して定義されています。LUA には、Communications Server内のプレゼンテーション・サービス層の固有のサブセットが含まれています。プレゼンテーション・サービス層の詳細については、*SNA 概念と諸製品*を参照してください。

LU サービスの機能は、SNA セッションのメッセージ・フロー層の一部となっています。これらの機能は、セッションの確立の前にサポートを提供し、セッション構造を構築し、そしてセッション構造を解体します。 LUA の機能は、LU を定義しSNA セッションを開始および停止するために、Communications Server共通サポートとのインターフェースとして働きます。

SNA セッションの使用

LUA アプリケーション・プログラムがパートナー・ホスト・アプリケーション・プログラムと通信できるようにするには、対応する 2 つの LU が、セッションと呼ばれる相互関係で結ばれていることが必要です。 SNA セッションは、2 つのネットワーク・アドレス可能単位 (NAU) が互いに通信できるようにする論理接続です。LU も NAU の一種です。 このセッションは 2 つの LU を接続するものなので、*LU-LU* セッション と呼ばれます。 LU-LU セッションは、エンド・ユーザー同士がデータを交換するために使用できます。

セッションは、SNA ネットワーク内の一対の LU 間でデータがどのように移動するかを管理します。したがって、セッションは、伝送するデータの量、データ・セキュリティー、ネットワーク経路指定、データ損失、および通信の混雑度などの事項に関係します。 セッション特性は、1 次 LU により発行された SNA **BIND** コマンドを 2 次 LU が 受け入れたときに、その **BIND** コマンドの内容によって決まります。

SNA セッションに関する前提条件

LU-LU セッションは、1 次論理装置 (PLU) と 2 次論理装置 (SLU) との間の通信で構成されます。 SLU は LUA アプリケーション・プログラムにより実現されます。LU-LU セッションにおいて PLU と SLU の間でデータを伝送される前に、次の事象が起こります。

1. Communications Serverがデータ・リンクを活動化する。
2. データ・リンクが作動可能な状態にあるときに、システム・サービス制御点 (SSCP) が、Activate Physical Unit (**ACTPU**) コマンドを送り、Communications

Serverからの肯定応答を読み取ることによって、SSCP と物理装置との間にセッションを確立します（SSCP-PU セッション）。 **ACTPU** コマンドからの PU アドレスが構成情報に対応していれば、Communications Serverは肯定応答を送ります。

3. SSCP が、Activate Logical Unit (**ACTLU**) コマンドを送り、Communications Serverからの肯定応答を読み取ることによって、SSCP と論理装置との間のセッションを確立します（SSCP-LU セッション）。 **ACTLU** コマンドからの LU アドレスが構成情報に対応していれば、Communications Serverは肯定応答を送ります。

セッションの始動

SLU または PLU のどちらからでも LU-LU セッションを始動できます。

SLU からの LU-LU セッションの始動

SSCP-LU セッションが確立されると、SLU プログラムは、SSCP に Initiate Self (**INITSELF**) コマンドを送ることにより、LU-LU セッションを要求することができます。 SSCP は **INITSELF** コマンドを受け取り、指定されたホスト・アプリケーション・プログラムが有効かどうかを検査します。 ホスト・アプリケーション・プログラムが有効とみなされるのは、その名前が認識されていて活動状態にある場合です。 ホスト・アプリケーション・プログラムが有効であれば、SSCP は SLU に肯定応答を送り、PLU はセッションを始動します。 ホスト・アプリケーション・プログラムが有効でない場合は、SSCP は SLU に否定応答を送り、PLU はセッションを始動しません。

SSCP が **INITSELF** コマンドに対して肯定応答を送ったが、セッションの確立ができないという場合は、SSCP は Network Services Procedure Error (**NSPE**) コマンドをSLUに送って、セッション確立の試行を中止するように伝えます。 SLU は、**NSPE** コマンドの後で **INITSELF** コマンドを再発行できます。

PLU からの LU-LU セッションの始動

PLU プログラムは非送信請求 LU-LU セッションを始動できます。 PLU は、**BIND** コマンドを生成することによりセッションを始動します。 その後肯定応答が生じた時点で、通信を行うための合意が成立します。 **BIND** コマンドに関連したデータ・フィールドには、PLU アプリケーション・プログラムの名前と、セッションの **BIND** パラメーターが含まれています。 このデータ・フィールドのフォーマットについては、*Systems Network Architecture: Formats* を参照してください。

交渉不可能 **BIND** の場合は、パラメーターが受入れ可能であれば、SLU は肯定応答を戻します。 パラメーターが受入れ不能である場合は、SLU は否定応答とセンス・データを PLU に戻します。

交渉可能 **BIND** コマンドでは、SLU は、PLU パラメーターとの互換性を示す最低26バイトの更新済みセッション・パラメーターを付けて、肯定応答を戻すことができます。 戻されたパラメーターを受入れ可能と認めた場合は、PLU は Start Data Traffic

(**SDT**) コマンドを送ります。 戻されたパラメーターが受入れ不能である場合は、PLU は、SLU からの交渉可能 **BIND** コマンドのパラメーターが受入れ不能であることを示す **UNBIND** コマンドを送ります。

LU-LU セッションでのデータの転送

LU-LU セッションが確立され、SLU プログラムが **SDT** コマンドに応答すれば、データ転送を開始できます。 データ送信操作では、メッセージは、転送するものがなくなるまで、エンド・ユーザーの記憶域からCommunications Serverの記憶域に移動します。 データ受信操作では、Communications ServerはメッセージをCommunications Server自身の記憶域に入れ、その後でエンド・ユーザーの記憶域にそのメッセージを移動します。

静止プロトコルは、LU-LU セッションでのデータの転送を中断します。 PLU または SLU は次の静止プロトコル・コマンドを送ることができます。

- Quiesce at End of Chain (**QEC**)。 このコマンドは、このコマンドの受信側に、データ・チェイニングの最後の部分の送信後にデータ送信を停止するよう要求します。データ・チェイニングは一連の関連するメッセージです。 データ・チェイニングの詳細については、183ページの『データ・チェイニング・プロトコルの使用』を参照してください。
- Quiesce Complete (**QC**)。 このコマンドは、**QEC** コマンドにデータ転送が中断されたことを通知します。 SLU が **QC** コマンドを送ると、Communications Serverは、Release Quiesce (**RELQ**) コマンドを受け取るまで SLU が通常フロー・メッセージを送信できないようにします。
- Release Quiesce (**RELQ**)。 このコマンドは、受信側にデータが再度転送可能になったことを通知します。

セッションの停止

すべてのデータの転送と検査が終われば、セッションを終了できます。 SLU は、1 つのセッションを終了してからでなければ、同一または他の PLU との新しいセッションを始動することはできません。

SLU による LU-LU セッションの停止

SLU が LU-LU セッションを停止する方法は 2 つあります。

- Terminate Self (**TERMSELF**) コマンドまたは **UNBIND** コマンドを送る。 どちらのコマンドの場合もセッションは即時終了します。
- Request Shutdown (**RSHUTD**) コマンドを送る。このコマンドは PLU からの **UNBIND** を要求します。

セッションを即時に終了したいときは、SLU は **TERMSELF** コマンドを SSCP に送ります。SSCP は、指定されている LUA アプリケーション・プログラムがこのセッションに関与しているものかどうかを検査します。 そうである場合は、SSCP は肯定の非データ応答を送ります。 使用しているホスト SNA バージョンによっては、SSCP は、**CLEAR** コマンドを送って LU-LU セッションからすべてのメッセージを

除去し、次に、**UNBIND** コマンドを送ってセッションを終了することができます。あるいは、SLU が PLU に **UNBIND** コマンドを送ることもできます。

PLU による LU-LU セッションの停止

PLU が LU-LU セッションを停止する方法は 2 つあります。

- **CLEAR** コマンドに続けて **UNBIND** コマンドを送るか、または **UNBIND** コマンドだけを送る。 どちらの方法でもセッションは即時終了します。
- シャットダウン (**SHUTD**) コマンドを送る。 このコマンドでは、セッションは正規の手順に従って終了します。 SLU と PLU は対話をかわし、互いにデータの送信を停止するよう指示し、すでに送信済みのデータを受信したことを確認し合います。

LU-LU セッションを終了しても、SSCP-LU セッションには影響はありません。

SSCP-LU セッションおよび SSCP-PU セッションの停止

ホストが SLU に Deactivate Logical Unit (**DACTLU**) コマンドを送ると、SSCP-LU セッションは終了します。 Communications Serverの最後の SSCP-LU セッションが終了した後は、SSCP は、Deactivate Physical Unit (**DACTPU**) コマンドを送信することにより、SSCP-PU セッションを終了することができます。

ホスト・リンクの切断

DACTPU コマンドに対する応答を受信すると、SDLC プロトコルの使用時には、ホストは Set Disconnect Response Mode (**SDRM**) などのようなコマンドをCommunications Serverに戻します。 また、SSCP は、同じコマンドをCommunications Serverに送ることにより、いつでも即時に切断することができます（この場合はすべてのセッションが終了します)。 このようにしてセッションが終了した場合、それまでに活動状態にあったすべての SLU が loss-of-contact 標識を受け取ります。

メッセージ番号

LU-LU セッションにおいて、SLU と PLU の間で伝送されるすべての通常フロー・メッセージには、順番に番号が付きます。 SLU は、SLU から PLU への通常フロー・メッセージの順序番号と、PLU から SLU への通常フロー・メッセージの順序番号を、別々に維持しています。 各通常フロー・メッセージには、その前の通常フロー・メッセージの番号より 1 つ大きい順序番号が与えられます。 SLU と PLU の間に確立される各セッションごとに、一対ずつの順序番号があります。

LU-LU 急送フロー・メッセージ、およびすべての SSCP-LU や SSCP-PU メッセージの場合は、順序番号の代わりに順序番号がないことを示す識別子が使用されます。

セッションが再確立されるか、または **CLEAR** コマンドが送信されると、PLU および SLU はそれぞれの順序番号を 0 に設定します。PLU は、Set and Test Sequence

Numbers (**STSN**) コマンドを使用して、順序番号を変更できます。 これにより、セッションが回復または再始動されたときに、正しい順序番号にセットすることができます。

順序番号エラーが見つかると、SLU は、応答が要求されている場合は否定応答をPLUに送ります。 SLU は、応答を受け取ると、応答順序番号を使用して、その応答を元の要求に対応付けします。 SLU が応答を送るときには、SLU は元の要求の順序番号を提供する必要があります。

セッションの再始動と再同期

PLU または SLU に対して、回線障害などのような回復不能エラーが起きた場合は、LU-LU セッションの再始動の後でセッションの再同期が必要になることがあります。LU-LU セッションの再同期には、回復可能なメッセージの再処理と、メッセージ順序番号の再設定（任意選択）が含まれます。 アプリケーション・プログラムに、失われたメッセージを再伝送するためのルーチンを組み込んでおくことができます。

セッションが再始動され再同期されると、PLU は、**BIND**、**STSN**、および **SDT** コマンドを送ります。 **STSN** コマンドが送られると、PLU と SLU の両方が受入れ可能な順序番号を確立するためにやりとりします。 このやり取りは、一連の **STSN** メッセージと肯定応答で成り立っています。

再同期が必要と判断した場合は、SLU は、Request Recovery (**RQR**) コマンド、否定応答、または LU-Status コマンド (**LUSTAT**) を、ユーザー・センス・バイトに入れて障害の記述とともに送ることができます。 PLU が障害を発見するかまたは SLU からの **RQR** コマンドを受信した場合は、PLU は、**CLEAR** コマンドを送ってネットワークからすべての LU-LU メッセージを除去し、**STSN** コマンドを送って新しい順序番号を設定し、そして **SDT** コマンドを送ります。

要求および応答を制御するためのプロトコルの使用

各種のプロトコルによって、要求および応答の順序に関する規則を制御することができます。ここでは、SNA ネットワークの管理、データの転送、およびネットワーク構成要素の状態の同期化のために使用するプロトコルのいくつかについて説明します。

ペーシング・プロトコルの使用

Communications Server またはホストにとって速すぎるメッセージ・フローが生じることがないようにするために、**BIND** コマンドでペーシングを指定できます。 ペーシングは LU-LU 通常フローのみに適用されます。 ペーシングが適用されているときは、Communications Serverは、指定された数のメッセージしか流れないように制限し、応答を待ってから後続のメッセージが送られるようにします。 ペーシングは、Communications ServerからホストへのフローとホストからCommunications Serverへの

フロー、およびその両方向のフローに対して指定できます。 LU-LU セッションが始動されると、LUA がすべてのペーシングを管理し、アプリケーション・プログラムはまったく関与する必要がありません。

受信ペーシング・プロトコル

受信ペーシング・プロトコルを使用すると、PLU が、LU-LU セッションで SLU から送られるメッセージの数と頻度を制御することができます。 SLU が **BIND** コマンドに含まれているペーシング値を受け取ると、Communications Serverは自動的に、ホストと通信する各 SLU にペーシングを適用します。

交渉可能 **BIND** コマンドに対する肯定応答では、ペーシング値を 0 以外の任意の数に変更できます。SLU が一連のメッセージのうち最初のメッセージを送ると、Communications Server は、要求/応答ヘッダー (RH) の中で、ペーシング応答が戻されることを示すビットをセットします。 Communications Server が PLU からのペーシング応答を受け取る前にペーシング・カウントがゼロになってしまった場合は、Communications Server はそれ以上データ・メッセージを送信することはできません。アプリケーション・プログラムが書込み操作を発行し、ペーシング応答が受信されなかった場合は、 Communications Server はその書込み操作を延期します。

送信ペーシング・プロトコル

SLU は、送信ペーシング・プロトコルを自動的に制御します。 PLU から SLU へのメッセージの中でペーシング標識がオンにセットされている場合は、SLU は、アプリケーション・プログラムがそのメッセージを読み取るときに、ペーシング応答を発行します。 ペーシング標識はメッセージ応答に含めることができます。あるいは、受信メッセージについて応答が必要ない場合は、分離ペーシング応答 (IPR) としてペーシング応答を送ることができます。 その場合は、PLU は別のメッセージのペーシング・ウィンドウを送信することができます。

半二重競合/フリップフロップ・プロトコルの使用

次のどちらのプロトコルにも方向変換 (CD) 標識が使用されます。

- 半二重競合プロトコル。これは通常フロー送受信モードであり、どちらかのハーフセッションが、セッションの始めか、または連鎖の最後の要求の送信または受信の後に、通常フロー要求を送ることができます。
- 半二重フリップフロップ・プロトコル。これは通常フロー送受信モードの 1 つで、一方のハーフセッションが、連鎖終りの時点で応答ヘッダー (RH) 内で CD 標識をセットして、相手のハーフセッションが送信を開始できるようにします。

CD 標識は、送信を開始できることを受信側に知らせます。

たとえば、SLU がトランザクションを開始する場合、SLU はまず、そのトランザクションを完全に記述したメッセージを送信します。 最後のメッセージで、SLU は、PLU が応答の送信を開始できることを示すように CD 標識をセットします。 PLU は、トランザクションを完了するために追加の情報が必要な場合は、照会を送り CD 標識をセットします。 トランザクションが完了するまで、この半二重モードでダイアロ

グが進められます。 半二重ダイアログでは、SLU は **SIG** コマンドを使用して、データの送信を停止しデータ・フローの方向を変更するよう PLU に指示することができます。

ブラケット・プロトコルの使用

ブラケット・プロトコルを使用すると、SLU およびPLU は、データ伝送のコンテキスト制御を行い、セッションが単一トランザクションに関与するものであることを指示できます。 ブラケット・プロトコルは、現行セッションが並行セッションにより中断されるのを防ぎます。ブラケットは、１つのトランザクションの範囲を包含します。

ブラケット内の最初のメッセージにはブラケット開始 (BB) 標識が含まれ、ブラケット内の最後のメッセージにはブラケット終了 (EB) 標識が含まれています。１つのメッセージに両方の標識が含まれていれば、そのメッセージは単独で１つのブラケットになります。

ブラケット・セッションの場合は、**BIND** コマンドは、一方の LU をファースト・スピーカーとして指定し、もう一方の LU を送信権要求者として指定します。 ファースト・スピーカーは、相手の LU からの許可を必要とせずにブラケットを開始できます。 しかし、送信権要求者がブラケットを開始するには、ファースト・スピーカーに許可を要求し許可を受ける必要があります。

BID コマンドは、送信権要求者がブラケット開始の許可を要求するために発行する通常フロー要求です。 **BID** に対する肯定応答は、ファースト・スピーカーがブラケットを開始せずに、送信権要求者によるブラケットの開始を待つことを示します。 **BID** コマンドに対する否定応答は、ファースト・スピーカーが、送信権要求者によるブラケット開始の許可を拒否したことを意味します。 ブラケット開始の許可を与えるときは、ファースト・スピーカーは Ready-to-Receive (**RTR**) コマンドを送ることができます。

ファースト・スピーカーは、**BID** コマンドに対して否定応答を送るときに、次の２つの応答コードのいずれかを付加します。

Bracket-Bid-Reject-RTR-Forthcoming
この **BID** コマンドに対する **RTR** コマンドを後で送る（ブラケットの開始を許可する）ことを示します。送信権要求者は、**RTR** コマンドを待つか、または再度 **BID** コマンドを送ることができます。

Bracket-Bid-Reject-No-RTR-Forthcoming
この **BID** コマンドに対しては後で **RTR** コマンドを送らないことを示します。送信権要求者は、それでもなおブラケットの開始を望む場合は、再度 **BID** コマンドを送る必要があります。

送信権要求者は、**BID** コマンドに続けて BB 標識を含む連鎖先頭 FMD を送る代わりに、BB 標識を含む連鎖先頭 FMD を送信することによって、ブラケットの開始を試みることもできます。 これに対して、ファースト・スピーカーは、肯定応答によりその試行を許可するか、いずれかの否定応答コードを示す否定応答により許可を拒

否することができます。 ただし、送信権要求者が **CANCEL** コマンドを送信することにより、BB 標識を含む連鎖を停止した場合は、応答に関係なくブラケットは開始されません。 ファースト・スピーカーは、送信権要求者にブラケット開始の許可を与えるため、または送信権要求者がブラケット開始を望んでいるかどうかを確認するために **RTR** コマンドを使用できます。

RTR コマンドに対する肯定応答は、送信権要求者が次のブラケットを開始するつもりであることを示します。 ブラケットの開始を望まない場合は、送信権要求者は、「RTR 不要」センス・コードを伴う否定応答を発行します。

データ・チェイニング・プロトコルの使用

データ・チェイニング・プロトコルは、一連の関連メッセージを伝送するための任意選択プロトコルです。 SLU から連鎖メッセージを送信するには、SLU は、連鎖内の最初のメッセージについて、連鎖開始 (BC) 標識を 1 にセットします。 連鎖内の最初と最後の間にあるすべてのメッセージについては、SLU は、BC 標識および連鎖終了 (EC) 標識をどちらも 0 にセットします。連鎖内の最後のメッセージについては、EC が再び 1 にセットされます。 SLU は、メッセージを受け取ると、連鎖標識を調べて、メッセージがチェーンされているかどうかを判別します。

データ・チェイニング・プロトコルは、次に示す 3 種類の連鎖から成っています。

- 無応答連鎖。 連鎖内の各要求に*無応答* のマークが付けられます。
- 例外応答連鎖。 連鎖内の各要求に*例外応答* のマークが付けられます。
- 確定応答連鎖。 連鎖内の最後の要求に*確定応答* のマークが付けられ、連鎖内の他のすべての要求には*例外応答* のマークが付けられます。

PLU にメッセージ連鎖を送るとき、SLU または PLU がメッセージ・エラーを見つけた場合は、SLU は **CANCEL** コマンドを送ることができます。 SLU が PLU に **CANCEL** コマンドを送ると、PLU は、この連鎖内ですでに受信済みのすべてのメッセージを破棄します。 連鎖内の要素のどれかに対して PLU が否定応答を送った場合は、SLU は、連鎖を正常に終了させるか、または **CANCEL** コマンドを送ります。

データ交換制御方式

SNA セッションは、整然としたデータ交換のための規則にもとづいて管理されます。

フロー・プロトコル

トランスポート・レベルでは、データは、半二重 (HDX) プロトコルまたは全二重 (FDX) プロトコルによって交換されます。

半二重プロトコルを使用すると、一時には片方の LU は送信だけ、もう片方の LU は受信だけを行い、したがってデータは、一時には一方向にだけ流れます。 半二重のフリップフロップ・プロトコルでは、どちらの LU に送信権があり、どちらの LU に

受信権があるかを、両方の側で認識しています。 指定された時がくると、パートナー LU は、受信側が送信を行い、送信側が受信を行えるように、フローの方向を変更することに同意します。

全二重プロトコルを使用すると、データはいつでもどちらの方向にも流れることができます。 どちら側の LU も制約なしに送信と受信を行えます。

応答モード

SNA メッセージは、どれも、要求か応答のいずれかです。 一方の LU からの各要求は、それに対応する応答をパートナー LU から引き出します。 その応答には要求と同じ伝送順序番号が入っているので、応答と要求はその順序番号によって突き合わせることができます。

応答必須と RH に指定されている要求をアプリケーションが受信したときは、アプリケーションは応答メッセージを生成し、送信する必要があります。 応答を送信しなければならない時点は、応答モードの規則によって決まります。

即時応答モードでは、アプリケーションは、要求に対する応答を送信してからでないと、それ自体の要求を送信することはできません。 一方、遅延応答モードでは、応答は、要求の受信後いつでも送信することができます。

LUA 相関表

LUA は、着信要求と発信要求の順序番号を追跡しますが、この追跡は要求がその応答を受信するまで、すなわち、アプリケーションが着信要求に応答を発行するまで、または PLU が発信要求に応答するまで、行われます。 これらの番号は、*相関表*と呼ばれる Communications Server の区域に記録されます。

即時応答モードでは、1 つのセッションでは、未解決要求は数個までしか、通常は多くても 1 つしか生成できません。 遅延応答モードでは、もっと多くの未解決要求を生成することができます。

LUA 相関表は動的に管理されます。 LUA は任意の数の応答を記録することができます。 非常に多くの応答が累積した場合（おそらくはプログラム論理エラーのために）は、メモリーでのサーバーの実行はゆっくりになり、Communications Server は遮断してしまう可能性があります。

例外応答要求 (RQE)

ほとんどの場合、LUA はユーザー・プログラムから援助されなくても、自動的に要求と応答を相関させることができます。 LUA は要求および応答 RU のセッション内の動きを監視します。 LUA は、要求が応答を必要とする時点、および応答が送信された時点を通知することができます。 ただし、応答が送信されてもそれを LU では通知できない場合が 1 つあり、この場合はユーザー・プログラムで通知する必要があります。

応答が必須であるか、不要であるか、あるいは任意選択であるかは、要求の RH のビット・フィールドに指定されます。 応答が必要でない場合は、LUA はその相関表に応答番号を格納する必要がありません。 応答が必須である場合は、応答はそのフローの次のメッセージとして送信される必要があります。 LUA はこのメッセージを相関表に入れますが、次には応答が来る必要があるので、このメッセージはすぐに消去されます。

RH の中のエラー応答標識 (ERI) は、応答が任意選択であり、受信 LU が RU を受け取れないまたは処理できない場合だけ応答が必要であることを指定するものです。この任意選択の応答RUは例外応答要求（省略してRQE）と呼ばれます。LUAは、RQEに関して、常に自動的に相関表を管理できるわけではありません。表 12 には、LUA がその相関表から自動的に受信 RQE を消去できる場合と、受信 RQE を消去する前にアプリケーションからのシグナルを待機する必要がある場合とを要約してあります。

表 12. RQE の消去

即時応答モード	遅延応答モード			
verb	HDX	FDX	HDX	FDX
RUI_READ	自動	自動	アプリケーションの応答	アプリケーションの応答
RUI_WRITE	自動	アプリケーションの応答	アプリケーションの応答	アプリケーションの応答

HDX または FDX セッションのどちらの場合も、即時応答モードでは、アプリケーションが入力を要求する（RUI_READ を使用して）と、LUA はただちに RQE の番号を廃棄することができます。これは、即時応答モードでは、応答を送信してからでないと別の要求を発行することができないからです。 また、HDX 接続の即時応答モードでは、アプリケーションが出力を要求する （RUI_WRITE を使用して）と、LUA はただちに RQE の番号を廃棄することができます。これは、この出力が RQE 応答であるか、または応答は送信されないからです。

その他のすべての場合は、RQE に対する応答が作成されるかどうかは LUA にはわかりません。 そこで、アプリケーションでは、PLU（否定応答だけを必要とする）のためではなく、RQE が受け入れられたので否定応答は作成されないということを LUA に通知するために、RQE に対する肯定応答をフォーマットし、送信する必要があります。

ここで、LUA は、表から RQE を消去することができます。 この応答は肯定応答ですが、PLU は否定応答しか必要としないので、LUA はネットワークでこのアプリケーションの応答を転送しません。

すなわち、アプリケーションは、ただ LUA を援助するためにだけ、受信 RQE RU が確定応答 RU であるかのようにそれらを処理する必要があります。

セッション・プロファイル

特定のセッションで使用できる固有の SNA プロトコルと規則は、いっしょになってセッションの「プロファイル」を構成します。 セッションにバインドできるプロファイルは 2 つあります。すなわち伝送サービス (TS) プロファイルと機能管理 (FM) プロファイルです。 どちらのプロファイルを選択するかは、BIND の発行時に決められます。

TS プロファイル

SNA には 5 つの TS プロファイルが 1、2、3、4、および 7 という番号で定義されています。しかし、TSプロファイルの1はSSCPとPUの間でしか使用できないため、LUA アプリケーションで使用できるのは 2、3、4、および 7 のプロファイルだけです。これらのプロトコルは、表 13 に示すように、そこで発行できるコマンドが異なります。

表 13. TS プロファイルの特性

プロファイル	ペーシングの使用	CLEAR	CRV	RQR	SDT	STSN
2	常時	使用可	使用不可	使用不可	使用不可	使用不可
3	常時	使用可	任意選択	使用不可	使用可	使用不可
4	常時	使用可	任意選択	使用可	使用可	使用可
7	任意選択	使用不可	任意選択	使用不可	使用不可	使用不可

FM プロファイル

SNA には FM プロファイルとして番号 0、2、3、4、6、7、18 および 19 の 8 つが定義されています。 しかし、プロファイルの 0 と 6 は SSCP でしか使用できず、また、プロファイル 19 は LU タイプ 6.2 のみで使用されるため、LUA アプリケーションで使用できるプロファイルは 5 つだけです。 そのそれぞれは、使用できる SNA 機能に違いがあります。

FM プロファイルの要約を186ページの表 14 に示してあります。この表が空欄になっている場合は、そのプロファイルでは SNA 機能は制限されません。すなわち、BIND のパラメーターをどのように指定することもできます。

LUA RUI がサポートする FM プロファイル 2、3、4、7 および 18

表 14. FM プロファイルの特性

SNA 機能	FMP 2	FMP 3	FMP 4	FMP 7	FMP 18
要求モード	SLU は遅延を使用する				
応答モード	SLU は即時を使用する	即時	即時	即時	即時
RU 連鎖	単一 RU 連鎖のみ				
長さ検査付き圧縮				LU 0 のみ	
FMH-1 セッション制御ブロック (SCB) 圧縮	使用不可				

表 14. FM プロファイルの特性 (続き)

SNA 機能	FMP 2	FMP 3	FMP 4	FMP 7	FMP 18
使用できるデータ・フロー制御 RU	なし	• CANCEL • SIGNAL • LUSTAT（SLU のみ） • CHASE • SHUTD • SHUTC • RSHUTD • BID, RTR	• CANCEL • SIGNAL • LUSTAT • QEC • QC • RELQ • CHASE • SHUTD • SHUTC • RSHUTD • BID, RTR	• CANCEL • SIGNAL • LUSTAT • RSHUTD	• CANCEL • SIGNAL • LUSTAT • CHASE • BIS, SBI • BID, RTR
FM ヘッダー	使用不可				
ブラケット	制限付き使用				
フロー・プロトコル	FDX				
回復	PLU のみによる				

LUA verb の使用

アプリケーションは、LUA verb を介して LUA にアクセスします。 各 verb は LUA にパラメーターを提供し、LUA は要求された機能を実行し、アプリケーションにパラメーターを戻します。

verb の概要

次に、アプリケーションが使用することのできる 7 つの LUA verb についての概要を示します （各 verb の詳細説明については、第13章 LUA verbを参照してください)。

RUI_BID

ホストからの情報が読取り可能であることを、アプリケーションに知らせます。

RUI_INIT

LUA アプリケーションが使用する LU-SSCP セッションをセットアップします。

RUI_PURGE

未完了の **RUI_READ** verb を取り消します。

RUI_READ

LU-SSCP セッションまたは LU-LU セッションで、ホストから LUA アプリケーションの LU に送られたデータまたは状況情報を受信します。

RUI_TERM

LUA アプリケーションが使用する LU-SSCP セッションを終了します。 また、LU-LU セッションが活動状態にあるときは、それを使用不能にします。

RUI_WRITE

LU-SSCP セッションまたは LU-LU セッションで、ホストにデータを送信します。

さらに、Communications Serverは LUA アプリケーションへ RUI_INIT_STATUS 標識を戻すことができます。 このため、未完了の **RUI_INIT** verb の処理中に状態に関する情報を伝えることができます。

RUI セッション

RUI セッションとは、アプリケーションが定めた時間だけ LU を所有することであり、これには、SSCP と LU との間のセッション（SSCP-LU セッション）を確立する操作も含まれます。 また、RUI セッションには、オーバーラップしない 1 つまたは複数の LU-LU セッションを確立することも含まれます。接続不良またはその他のリセット条件が原因で SSCP-LU セッションが失敗した場合は、RUI セッションは終了します。 RUI セッションは **RUI_INIT** verb で始まり、通常は **RUI_TERM** verb で終了します。

RUI verb の発行

189ページの表 15は、RUI アプリケーション・プログラムが、特定の LU についての RUI API に対して verb を発行するための、有効な条件を示しています。 左端の欄の項目は着信 verb を示します。 一番上の行の項目は、実行中の verb を表しています。 表の中の項目が「可」であれば、その verb の組合せは有効な条件を表しています。 表の中の項目が「エラー」であれば、その verb の組合せは不適正な条件を表すものであり、LUA アプリケーション・プログラムにエラー・コードが戻されます。

表 15. RUI verb の条件

	実行中のコマンド						
着信コマンド	**現行セッションなし**	**RUI_INIT**	**RUI_TERM**	**RUI_WRITE**	**RUI_READ**	**RUI_PURGE**	**RUI_BID**
RUI_INIT	OK	エラー	エラー	エラー	エラー	エラー	エラー
RUI_TERM	エラー	OK	エラー	OK	OK	OK	OK
RUI_WRITE	エラー	エラー	エラー	OK (注 1 を参照)	OK	OK	OK
RUI_READ	エラー	エラー	エラー	OK	OK (注 2 を参照)	OK	OK
RUI_PURGE	エラー	エラー	エラー	OK	OK	エラー	OK
RUI_BID	エラー	エラー	エラー	OK	OK	OK	エラー

注:

1. RUI では、**RUI_WRITE** verb を 1 セッションにつき同時に 2 つまで活動状態にできます。 ただし、これらの活動状態の **RUI_WRITE** verb は、それぞれ異なるセッション・フローに対するものでなければなりません。可能なセッション・フローには次の 4 つがあります。
   - SSCP-LU 急送
   - SSCP-LU 通常
   - LU-LU 急送
   - LU-LU 通常
2. RUI では、**RUI_READ** verb を 1 セッションにつき同時に 4 つまで活動状態にできます。 ただし、これらの活動状態の **RUI_READ** verb は、それぞれ異なるセッション・フローに対するものでなければなりません。

非同期 verb の完了

LUA verb には、何らかのローカル処理の後で即時に完了するものもいくつかあります（たとえば **RUI_PURGE** verb）。しかし、ほとんどの verb では、ホスト・アプリケーションとの間でのメッセージの送信および受信が必要なので、完了までにある程度時間がかかります。 このため、LUA は非同期インターフェースとして設計されています。つまり、verb がまだ処理中であってもアプリケーションに制御を戻すことができるので、アプリケーションは、別の LUA verb を発行することも含めて、自由に処理を先へ進めることができます。LUA はアプリケーションに制御を戻すための手段として、verb の中のイベント・ハンドルを使用します。

Communications Server の verb の応答シグナルが遅れる場合 (たとえば、リモート・ノードからの情報を待つ必要があるため)、スタブはその verb を非同期で戻す必要があります。 API は、これを行うために、 1 次戻りコードを LUA_IN_PROGRESS に、そして **lua_flag2** を LUA_ASYNC にセットします。 これに対して、アプリケーションは、他の処理を行うか、または、verb の完了を示す API からの通知を待つことができます。 verb が完了すると、それをアプリケーションに通知するために、VCB 内の 1 次戻りコードが最終値にセットされ、**lua_flag2** は LUA_ASYNC にセットされたままになります。

LUA 通信順序のサンプル

次の示すのは LUA 通信順序の例です。 この例は、セッションの開始、データの交換、およびセッションの終了に対して使用される LUA verb、そして送受信されるSNA メッセージを示しています。 矢印は、SNA メッセージ・フローの方向を示します。

次の省略語を使用しています。

SSCP norm
LU-SSCP セッション、通常フロー

LU norm
LU-LU セッション、通常フロー

LU exp
LU-LU セッション、急送フロー

+rsp 示されているメッセージに対する肯定応答
BC 連鎖開始
MC 連鎖中間
EC 連鎖終了
CD 方向変換標識設定
RQD 確定応答要求

LUA アプリケーションが発行する verb	SNA メッセージ	フローの方向 アプリケーション	ホスト
RUI_INIT	(ACTLU)	<-----	
	(ACTLU +rsp)	----->	
RUI_WRITE (SSCP norm)	INITSELF	----->	
RUI_READ (SSCP norm)	INITSELF +rsp	<-----	
RUI_READ (LU exp)	BIND	<-----	
RUI_WRITE (LU exp)	BIND +rsp	----->	
RUI_READ (LU exp)	SDT	<-----	
RUI_WRITE (LU exp)	SDT +rsp	----->	
RUI_WRITE (LU norm)	データ、BC	----->	
RUI_WRITE (LU norm)	データ、MC	----->	
RUI_WRITE (LU norm)	データ、EC、CD、RQD	----->	
RUI_READ (LU norm)	データ +rsp	<-----	

LUA アプリケーションが発行する verb	SNA メッセージ	フローの方向 アプリケーション	ホスト
RUI_READ (LU norm)	データ、BC	<-----	
RUI_READ (LU norm)	データ、MC	<-----	
RUI_READ (LU norm)	データ、EC、RQD	<-----	
RUI_WRITE (LU norm)	データ +rsp	----->	
RUI_READ (LU exp)	UNBIND	<-----	
RUI_WRITE (LU exp)	UNBIND +rsp	----->	
RUI_TERM	(NOTIFY)	----->	
	(NOTIFY +rsp)	<-----	

この例では、アプリケーションは以下の手順を行います。

1. **RUI_INIT** verb を発行して、LU-SSCP セッションを確立します（**RUI_INIT** verb は、Communications Serverがホストから ACTLU メッセージを受信し、肯定応答を送信するまで、完了しません。ただし、これらのメッセージはCommunications Serverによって処理され、LUA アプリケーションには提示されません)。
2. SSCP に **INITSELF** メッセージを送って **BIND** を要求し、そして応答を受信します。
3. ホストからの **BIND** メッセージを受信し、応答を送信します。 これで LU-LU セッションが確立されます。
4. ホストからの **SDT** メッセージを受信します。このメッセージは、初期設定が完了し、データの転送を開始できることを示します。
5. ３つの RU（最後のものは確定応答が要求であることを示す）から成るデータの連鎖を送り、応答を受信します。
6. ３つの RU から成るデータの連鎖を受信し、応答を送信します。
7. ホストからの **UNBIND** メッセージを受信し、応答を送信します。 これで LU-LU セッションが終了します。
8. **RUI_TERM** verb を発行して、LU-SSCP セッションを終了します (Communications Serverは NOTIFY メッセージをホストへ送り、肯定応答を待ちます。ただし、これらのメッセージはCommunications Serverによって処理され、LUA アプリケーションには提示されません)。

BIND 検査

LU-LU セッションの初期設定の際に、ホストは、LU-LU セッションで使用する RU サイズなどの情報が入った **BIND** メッセージを、Communications Server LUA アプリケーションに送ります。 Communications Serverは、このメッセージを **RUI_READ** verb 上の LUA アプリケーションへ戻します。 **BIND** で指定されているパラメーターが適正であるかどうかを確認するのは、LUA アプリケーションで行います。 このアプリケーションには次のオプションがあります。

- **BIND** に対する OK 応答を含む **RUI_WRITE** verb を発行することにより、**BIND** をそのまま受け入れる。 応答でデータを送る必要はありません。
- １つまたは複数の **BIND** パラメーターについて交渉する（これができるのは **BIND** が交渉可能である場合だけです)。 そのために、アプリケーションは、OK 応答を含み、**BIND** によって変更されたパラメーターを **RUI_WRITE** verb を発行します。
- 該当する SNA センス・コードをデータとして使用して、否定応答を含む **RUI_WRITE** verb を発行することにより、**BIND** を拒否します。

RUI_WRITE verb の詳細については、第13章 LUA verbを参照してください。

注: **BIND** のパラメーターの妥当性を検査し、送信されたすべてのメッセージがそれらのパラメーターに矛盾しないことを確認するのは、LUA アプリケーションで行います。 ただし、次の２つの制約条件が適用されます。

- Communications Serverは、**BIND** 上で指定されたサイズよりも大きい RU の長さを指定している **RUI_WRITE** verb を拒絶します。
- Communications Serverは、２次 LU が競合勝者であり、エラー回復が競合敗者の責任であることを指定する **BIND** を必要とします。

否定応答と SNA センス・コード

SNA センス・コードが LUA アプリケーションに戻されることがあるのは、次のような場合です。

- ホストが LUA アプリケーションからの要求に対して否定応答を送るときは、否定応答の理由を示す SNA センス・コードが含まれています。これは、後続の **RUI_READ** verb で次のようにアプリケーションに報告されます。
  - １次戻りコードは LUA_OK です。
  - 要求/応答標識、応答タイプ標識、および Sense Data Included Indicator (SDI)すべて１（センス・データを含む否定応答を示す）にセットされます。
  - **RUI_READ** verb が戻すデータは SNA センス・コードです。
- Communications Server は、ホストから正しくないデータを受信すると、通常はホストに否定応答を送信し、その正しくないデータは LUA アプリケーションには渡されません。 これは、後続する **RUI_READ** または **RUI_BID**verb で、次のようにアプリケーションに報告されます。
  - １次戻りコードは LUA_NEGATIVE_RSP です。
  - ２次戻りコードは、ホストに送られる SNA センス・コードです。

- Communications Server は、ホストから提供されたデータが無効であることを検出しても、送るべきセンス・コードとしてどれが正しいのかを決めることができない場合があります。 このような場合は、Communications Server は **RUI_READ** verb を発行するときに、その無効データを例外要求 (EXR) に入れて、次のような方法で LUA アプリケーションに渡します。
  - 要求/応答標識を、要求を示す 0 にセット
  - Sense Data Included Indicator (SDI)を、センス・データが含まれていることを示す 1 にセット（通常この標識は応答の場合のみ使用）
  - メッセージ・データを提示 SNA センス・コードで置き換える。

  アプリケーションは、このメッセージに対して否定応答を送る必要があります。アプリケーションは、Communications Serverから提示されたセンス・コードを使用することも、それを変更することもできます。
- Communications Server は、アプリケーションから提供されたデータが無効であることを示すために、そのアプリケーションへセンス・コードを送る場合があります。 これは、そのデータを提供した **RUI_WRITE** verb で、次のようにアプリケーションに報告されます。
  - 1 次戻りコードは LUA_UNSUCCESSFUL です。
  - 2 次戻りコードは SNA センス・コードです。

SNA センス・コードと他の 2 次戻りコードとの区別

センス・コードでない 2 次戻りコードの場合は、この値の最初の 2 バイトは常に 0 です。 SNA センス・コードの場合は、最初の 2 バイトは 0 以外の値です。つまり、1 バイト目はセンス・コードのカテゴリーを示し、2 バイト目はそのカテゴリーの中での特定のセンス・コードを識別します （3 バイト目と 4 バイト目には、追加の情報が含まれることもあり、0 のこともあります)。

SNA センス・コードに関する情報

戻されたセンス・コードに関する情報が必要な場合は、*IBM Systems Network Architecture: Formats* を参照してください。センス・コードは、カテゴリー別の番号順にリストされています。

ペーシング

ペーシングは LUA が処理します。LUA アプリケーションはペーシングを制御する必要はなく、したがってペーシング標識フラグをセットしてはなりません。

LUA アプリケーションからホストに送るデータについてペーシングを使用する場合は（これは **BIND** により決まります）、**RUI_WRITE** verb は完了までに少し時間がかかることがあります。 これは、Communications Serverがそれ以上のデータを送るためには、ホストからのペーシング応答を待つ必要があるためです。

LUA アプリケーションを使用して、ホストへ、またはホストから一方向に大量のデータを転送する場合は（たとえばファイル転送アプリケーションの場合）、ホスト

構成で、その方向にペーシングを使用することを指定する必要があります。これは、データを受信するノードでデータがあふれたり、データ記憶域を使いきってしまったりすることがないようにするためです。

セグメンテーション

RU セグメンテーションは LUA が処理します。 LUA は、常に完全な RU（セグメンテーションされていない RU) をアプリケーションに渡し、アプリケーションは完全な RU をLUA に渡します。

形式的な受信確認

Communications Serverは、アプリケーションによって送られた応答と正しい要求とを関連付けるために、ホストから受け取られた要求の記録を保持しています。アプリケーションが応答を送ると、Communications Serverは、その応答を元の要求からのデータと関連付けした上で、関連の記憶域を解放することができます。

ホストが例外応答のみ（否定応答は送信できるが肯定応答は送信できない）を指定している場合でも、Communications Serverは、アプリケーションが後で否定応答を送信する場合に備えて、要求の記録を保持する必要があります。 アプリケーションが応答を送信しなかった場合は、この要求に関連した記憶域を解放することができません。

そのため、Communications Serverは、ホストからの例外応答のみの要求に対しても、LUA アプリケーションは肯定応答を発行することができます (これは形式的な受信確認として知られています)。この応答はホストに送られるものではなく、Communications Sever が要求に関連した記憶域を消去するために使用します。

連鎖の終りまでのデータの除去

ホストがLUAアプリケーションに要求単位の連鎖を送るとき、アプリケーションは、連鎖内の最後の RU を受信するまで待ってから応答を送信することも、連鎖の最後ではない RU に対して否定応答を送ることもあります。 連鎖の途中で否定応答が送られた場合は、Communications Serverは残りのすべての RU をこの連鎖から除去し、アプリケーションには送りません。

Communications Serverは、連鎖内の最後の RU を受け取ると、そのことをアプリケーションへ知らせるために、**RUI_READ** または **RUI_BID** verb の１次戻りコードを LUA_NEGATIVE_RSP にセットし、２次戻りコードを０にセットします。

注: ホストでは、連鎖の途中で CANCEL などのようなメッセージを送信することにより、連鎖を終了することができます。 その場合は、**RUI_READ** verb によりアプリケーションに CANCEL メッセージが戻され、LUA_NEGATIVE_RSP 戻りコードは使用されません。

構成

LUA アプリケーションが使用する各 LU は、Communications Server の NOF verb または SNA ノード構成プログラムを使用して構成する必要があります （詳細は、*Communications Server システム管理プログラミング*を参照してください。）さらに、Communications Serverの構成は LUA LU プールを含む場合があります。プールは類似した特性を持つ LU のグループであり、アプリケーションは、このグループから空いている LU を任意に選んで使用できます。 これは、使用可能な LU よりアプリケーションの方が多いときに先着順で LU を割り振るためや、また、異なるリンクで異なる LU を選択できるようにするために使用できます。

LUA LU プール（任意選択）

必要があれば、アプリケーションで使用するために複数の LUA LU を構成し、それらの LU を 1 つのプールとしてグループ化することができます。 このようにすれば、アプリケーションが、セッションを始動しようとするときに特定の LU ではなくそのプールを指定でき、プール内の最初に使用可能になった LU がアプリケーションに割り当てられます。

LUA アプリケーションは、LU 名を指定した **RUI_INIT** verb を発行し、セッションの始動を望んでいることを Communications Server に知らせます。 この名前は、*Communications Server システム管理プログラミング*であらかじめ定義されているLUA LU または LU プールの名前と一致する必要があります。
Communications Server はこの名前を次のように使用します。

- 指定された名前がプール内に含まれていない LU の名前である場合は、その LU が使用可能であれば（つまりまだ他の LUA アプリケーションで使用中でなければ）、その LU を使用してセッションが割り当てられます。
- 指定された名前が LU プールの名前であるか、または、プール内に含まれていてすでに使用中の特定 LU の名前である場合は、プール内の最初の使用可能な LU（もしあれば）を使用してセッションが割り当てられます。

  注: これは、**RUI_INIT** verb に指定した名前の LU ではないこともあります。

SNA API クライアントの考慮事項

LUA アプリケーションをクライアント・ワークステーションにのせている場合は、LUA セッションもそのローカル・ワークステーションに定義する必要があります。この LUA セッションの名前には複数の通信サーバーと LUA 定義を指定することができるので、SNA クライアント・コードでは、ある接続が利用できなくなると新しいサーバーに切り換えることができます。

第10章 LUA verb の機能

この章では、LUA verb について、以下のような特別な場合とその使用上のヒントを述べます。

- 例外要求、つまりユーザー・プログラムが否定応答を出すための LUA からの要求の処理
- プログラムの設計による LAN 通信量の最小化
- LUA verb が無限中断に入った場合の処理
- セッション障害からの回復

例外要求の処理

RUI は、いくつかのプロトコルについてその状態を監視し、RU のフォーマットが妥当であるかどうかを検査します。 インターフェースは、PLU から不適切な RU が着信したことを検出したら、否定応答を発行する必要があります。 LUA は、着信 RU を例外要求 (EXR) としてフォーマットすることによって、この検出されたエラーについてユーザー・アプリケーションに通知します。 EXR は、送信権要求 verb (RUI_BID) または入力 verb (RUI_READ) に入れてユーザー・プログラムに送達されます。 EXR は、要求ヘッダー (RH) の中の以下の状態で EXR であるとわかります。

- *lua_rh.rri* が 0 にセットされている (RU は要求単位)
- *lua_rh.sdi* が 1 にセットされている (センス・データが含まれる)

これは RH ビットの組合せとして異常です。 センス・データは、通常は要求 RU ではなく、応答 RU に入っています。 LUA は、この異常な組合せを使用して、PLU で明らかにエラーが発生したという異常状態をユーザー・プログラムに警告します。4 バイトのセンス・コードは EXR の一部であり、検出したエラーを示します。センス・データの他に、LUA はオリジナル RU の最高 3 バイトを戻します。

verb レコードの変更

アプリケーションは、EXR を否定応答としてフォーマットし、使用している API にしたがって、いずれかの RUI_WRITE によって PLU に送信する必要があります。 EXR 入力を応答出力に変換するには、verb レコードに以下の変更を行います。

- これが応答であることを示すために、*lua_rh.rri* を 1 にセットします。
- 否定応答であることを示すために、*lua_rh.ri* を 1 にセットします。
- *lua_flag2* の値にもとづいて *lua_flag1* に適切なデータ・フロー・フラグをセットします。
- *lua_message_type* を LUA_MESSAGE_TYPE_0 にセットします。
- 使用中の API にしたがって、*lua_opcode* を LUA_OPCODE _RUI_WRITE にセットします。
- *lua_data_length* を 4（センス・データの長さ）にセットします。

- *lua_data_ptr* をセンス・データのアドレスにセットします。この位置は、EXR を検出した verb によって異なります。verb が RUI_BID であった場合は、センス・データは verb レコードの「ピーク・バッファー」の中に入っており、また、verb が RUI_READ であった場合は、センス・データは入力バッファーの中に入っています。
- *lua_max_length* を 0 にセットします。

これで、ユーザー・プログラムは、EXR のための verb レコードとバッファーを使用して、否定応答を送信するために RUI_WRITE を開始することができます。

ブラケット開始要求拒否の処理

1 つの例外を除きすべての場合において、EXR の中に LUA が提供するセンス・コードは、PLU に戻すのに適した唯一のセンス・コードです。 ただし、ブラケットが使用中で、PLU がスピーカーになることを求めているのであれば、アプリケーションは、次のようにセンス・コードを選択することができます。

- LUA は PLU からの BID コマンドを拒否することができます。 BID を拒否するには、LUA は、センス・コード LUA_BB_REJECT_NO_RTR を含む EXR をフォーマットして、ブラケットの開始を拒否すること、および RTR コマンドはこれ以降発行されないことを宣言します。 このセンス・コードを数値で表すと 0x00001308L (ユーザーが C プログラムでコードする場合は Intel** すなわちバイト・スワップ形式で) になります。
- アプリケーションは、ブラケット操作をサポートし、後に RTR コマンドを出すことができるのであれば、BID コマンドを受け入れることができます。 PLU にその BID が受入れ可能であると通知するには、センス・コードを LUA_BB_REJECT_RTR (値 0x00001408L) に変更します。このセンス・コードは、RTR が後に発行されることを示します。 アプリケーションは後で RTR メッセージをフォーマットして、送信する必要があります。

LAN 通信量の最小化

アプリケーションをクライアント・ワークステーションで実行する必要があるときは、『送信権要求論理』の使用を減らすことによって LAN 通信量のオーバーヘッドを最小にするようにアプリケーションを設計することができます。

RUI_BID 使用の抑制

verb RUI_BID は、データ単位がサーバーで利用できるようになるまで待機し、それから完了します。 RUI_BID が完了すると、ユーザー・プログラムに対して、データが特定のフローで特定の長さで準備されたことが通知されます。 この通知を受けたら、ユーザー・プログラムではバッファーを割り振り、データを求めて RUI_READ verb を発行することができます。

送信権要求 verb に続けて入力 verb を発行すると、次の 4 つの LAN メッセージが生成されます。

- RUI_BID を開始するメッセージ
- BID が完了したことをワークステーションに通知するメッセージ
- RUI_READ を開始するメッセージ
- データをワークステーションに戻すメッセージ

しかし、RUI_READ は同一のジョブを 1 ステップで行うことができます。 単に RUI_READ verb を開始して、それが完了するのを待機する場合は、LAN メッセージは 2 つ少なくなります。

「送信権要求論理」の唯一の利点は、メッセージを受け取る前にメッセージのサイズがわかることです。 それによって、必要になるバッファーの大きさがわかるまでデータ・バッファーの割振りを延ばすことができます。 入力 verb だけしか使用しないときは、送信権要求が完了した後でバッファーを割り振るのではなく、前もって最大のバッファー・サイズを知っておく必要があります。

中断の処理

RUI verb の完了は、PLU アプリケーション、ホスト・システム、ネットワークおよび Communications Server のアクションによって異なります。 これらのいずれか 1 つの応答が遅いか、またはいずれかが応答に失敗すると、verb はいつまでも中断されたままになる可能性があります。 プログラムの設計時に、中断された verb を終了する方法をユーザーまたはプログラムに提供することによって中断に備えることができます。

RUI_INIT の取消し

RUI_INIT verb は、割り当てられた LU をホストが活動化するまで、中断されます。 通常ホストは、アプリケーションの開始前に ACTLU コマンドを送信しますが、そうしなければならないわけではありません。 アプリケーションの開始時に、メインフレームは稼働していないかもしれないし、まだ初期化の最中であるかもしれません。

延期されている RUI_INIT をプログラムで取り消す必要がある場合は、RUI_TERM verb を出すことができます。

RUI_WRITE の取消し

ペーシングを使用しているときは、出力を中断することができます。 ホストが一時的にデータの読取りを停止したり、またはペーシング応答の転送に失敗した場合は、ペーシング・ウィンドウがオープンするのを待って RUI_WRITE が中断されることがあります。

中断されている RUI_WRITE を取り消さなければならない場合は、RUI_TERM でセッションをクローズする必要があります。

RUI_READ の取消し

入力 verb は、通常その verb が指定したフローに入力が到着するまで中断されます。プログラムは、RUI_PURGE を使用して保留中の RUI_READ を取り消すことができます。セッションをクローズした場合も、保留中の入力 verb は取り消されます。

verb 完了の確認

ユーザー・プログラムで verb の完了の処理を誤ると、無限待機を起こすことがあります。 プログラムが verb を開始し、その完了を同期的に注意せず、非同期に完了を待機してしまうと、プログラムは永久に待ち状態になります。

RUI エントリー・ポイントは、その明示的結果として、実行した verb の 1 次戻りコードを戻します。 verb が完了するかどうかを非同期に知る最も簡単な方法は、200ページの図 8 に示すように、この明示的戻りコードを LUA_IN_PROGRESS でテストすることです。

```
unsigned short rc;
rc = RUI(ptrToTheVerb);
if (LUA_IN_PROGRESS == rc)
   // verb will complete later; the callback function will be entered
else
   // verb is finished now; the callback function will never be entered
```

図 8. verb 完了テスト

セッション障害からの回復

LUA コミュニケーション verb が 1 次戻りコードの LUA_SESSION_FAILURE で完了した場合は、LUA セッションはエラーが原因でクローズされています。 具体的なエラーについては、2 次戻りコードで特定されます。

セッションはしばしば構築しなおすことができます。 LUA は、ユーザー・プログラムが要求すれば回復を試みます。

プログラムが操作の 1 次戻りコードとして LUA_SESSION_FAILURE を受け取った場合、回復したいのであれば、以下のようにする必要があります。

- セッションをクローズしないこと。セッションはすでにクローズされています。
- 最初にこのセッションをオープンしたときに使用した verb (RUI_INIT) を使って、セッションを再オープンする。 この verb がゼロ以外の 1 次戻りコードで完了している場合は、この時点でこのセッションは再始動できません。
- 回復が進行中の場合は、いつかは回復する可能性があるので対話式ユーザーに通知する。 ユーザーの作業の状態は、PLU アプリケーションの設計によって異なります。

第11章 LUA プログラムの実行

この章では、LUA プログラムの実行と作成に関するいくつかの局面について説明します。説明事項は次の通りです。

- LUA サービスの呼出しと順序付け
- LUA プログラムの作成
- 非同期完了とコールバック機能の使用
- 異なるプラットフォームにおけるコンパイルとリンク

Communications Server が提供する RUI は、Microsoft** NT SNA Server とのバイナリー互換性を備えたものとして設計されており、OS/2 コミュニケーション・マネージャー/2 バージョン 1.0 LUA の RUI インターフェースに似ています。

LUA プログラムの作成

LUA には、RUI verb のための主要 DLL が 1 つ含まれています。 LUA アプリケーション・プログラムは、verb を発行するためにこの DLL を呼び出します。

LUA アプリケーション・プログラムは、verb 制御ブロックの中に必要なフィールドを選択してセットし、RUI を呼び出して、その verb 制御ブロックを指すポインターを渡します。 verb 制御ブロック内のフィールドは、要求されるアクションを LUA に対して定義します。 LUA は、アプリケーション・プログラムに制御を戻す前に verb 制御ブロック内のフィールドを修正することで、アクションの結果を示します。アプリケーション・プログラムは、verb 制御ブロックから戻されたパラメーターを以後の処理に使用することができます。

下記の表は、RUI プログラムをコンパイルし、リンクするのに必要な、提供されているヘッダー・ファイルとライブラリーのソース・モジュールを示しています。

表 16. オペレーティング・システムのためのヘッダー・ファイルとライブラリー

オペレーティング・システム	ヘッダー・ファイル	ライブラリー	DLL 名
WINNT & WIN95	WINLUA.H	WINRUI32.LIB	WINRUI32.DLL
WIN3.1	WINRUI.H	WINRUI.LIB	WINRUI.DLL
OS/2	LUA_C.H	ACSRUI.LIB	ACSRUI.DLL

LUA サービスの呼出し

ユーザー・プログラムは、指示したエントリー・ポイントを呼び出して *verb レコード*と呼ばれるデータ構造のアドレスをパラメーターとして渡すことによってLUA サービスを呼び出します。 このレコードには、特定の機能のための入力パラメーターが入っています。 LUA は、この操作で得られる出力パラメーターで、このレコードを更新します。

verb レコードの内容

構造はそれぞれ異なっていますが、3 種類の verb レコードには、どれも以下のパラメーターのためのフィールドがあります。

操作　行うべき特定の操作を指定する番号。 操作の記号名は、『_cons.h』 組込みファイルに宣言されます。

verb レコード長
verbレコードのサイズ。これは操作によって異なります。このフィールドは、このレコードを処理するために LUA が必要とします。

セッション識別子
通信およびサービスの verb においてセッションを識別する番号、またはセッションの名前。

1 次戻りコード
一般的な正常または失敗を示すために LUA が戻す番号。

2 次戻りコード
失敗時に具体的な問題点を示すために LUA が戻す番号。

相関係数
verb レコードを他のデータに関連付けるため、または非同期完了時に verb レコードを識別するためにユーザー・アプリケーションが使用できる長整数。

通知ハンドル
verb が非同期に完了したときに通知されるイベントのハンドル。

Windows NT および Windows 95 では、これはイベント・ハンドルである必要があります。 Windows 3.1 では、これはウィンドウ・ハンドルです。 OS/2 では、これはセマフォア・ハンドルである必要があります。

これらのフィールドの多くは、そのぞれの verb レコードにおいて、データ・タイプが同じで、オフセットも同じです。 ただし、操作コードと verb の長さは、それぞれで異なる特性をもちます。

複数プロセス

LUA アプリケーション・プログラムは単一プロセスに制限されます。 異なるプロセスで同じ LUA アプリケーション・プログラムの異なるインスタンスを開始することはできますが、各アプリケーション・プログラムは、それぞれ異なる LUA LU を使用する必要があります。

さらに、1 つのプロセスを複数の LUA アプリケーション・プログラムで構成し、それぞれに専用の LUA LU を持たせることもできます。

複数スレッド

1 つの LUA アプリケーション・プログラムが、複数のスレッドを使用して verb を発行することもできます。これにより、1 つの LUA アプリケーション・プログラムか

ら同時に複数の verb を発行することができます。 異なるスレッドで同じ LUA アプリケーション・プログラムの異なるインスタンスを開始することができますが、各アプリケーション・プログラムがそれぞれ異なる LUA LU を使用する必要があります。

注: LUA アプリケーション・プログラムは、verb を発行してからも、その verb が完了するまでは、verb 制御ブロックのどの部分も変更しないでください。 RUI は、verb 制御ブロックのアプリケーション・コピーのみを使用します。 詳細は、203ページの『LUA verb の通知』を参照してください。

LUA verb の通知

LUA verb は同期または非同期に完了します。 verb の同期完了とは、LUA の呼出しの後で RUI が LUA アプリケーション・プログラムに戻った時点で、その verb に関するすべての処理が完了し、非同期ポスト方式が使用されないことを意味します。 verb はタイミングによっては非同期に完了することもありますが、LUA が LUA アプリケーション・プログラムに戻るときまでには、すべての処理が完了しています。 verb の非同期完了とは、成功か失敗かに関係なく処理が完了した時点で、LUA がポスト方式を使用してアプリケーション・プログラムに通知することを意味します。

verb が非同期に完了したとき、LUA アプリケーション・プログラムには次のいずれかの方法で通知できます。

- LUA アプリケーション・プログラムで **lua_flag2_async** および **lua_prim_rc** パラメーターを使用して、verb の処理状況を判別する。
- アプリケーションで、**lua_post_handle** パラメーターにイベントを指定する。このハンドルは verb の完了時にセットされます。

ASCII から EBCDIC への変換

一般に、ホストへのメッセージはすべて EBCDIC 形式で送られ、PLU もメッセージが EBCDIC であることを想定しています。 たとえば、**BIND** に含まれる PLU 名は EBCDIC ストリングでなければなりません。 ASCII でストリングを保持しているLUA アプリケーション・プログラムは、ストリングを SNA メッセージ内に送る前に、そのストリングを EBCDIC に変換する必要があります。

LUA アプリケーション・プログラムがアプリケーション・データを変換する必要があるかどうかは、パートナー・アプリケーション・プログラム間の個々の合意によって決まります。 LUA アプリケーション・プログラムが、通常 EBCDIC を使用しているノードと通信する場合は、ASCII データを適切なところで EBCDIC データに変換する必要があります。

ASCII から EBCDIC への（またはその逆の）変換には、271ページの『第15章 共通サービス verb (CSV)』で説明した変換 verb が使用できます。

第12章 LUA エントリー・ポイント

この章では、LUA のためのプロシージャー・エントリー・ポイントを説明します。

RUI DLL は以下のプロシージャー・エントリー・ポイントを定義しています。

RUI()

すべての **RUI** verb についてイベント通知を提供します。

```
void WINAPI RUI (LUA_VERB_RECORD* vcb);
```

パラメーター

	説明
vcb	指定パラメーター。verb 制御ブロックのアドレスを指定します。

lua_prim_rc に戻される値は、非同期通知が行われるかどうかを示します。 このフィールドが LUA_IN_PROGRESS にセットされている場合は、イベント通知を介して非同期通知が行われます。 このフラグが LUA_IN_PROGRESS 以外である場合は、要求は同期に完了しています。 1 次戻りコードおよび 2 次戻りコードを調べて、エラーの有無を確認してください。

アプリケーションは、イベントへのハンドルを、verb 制御ブロックの *lua_post_handle* パラメーターに指定する必要があります。 このイベントは未通知状態になっていなければなりません。

非同期操作が完了すると、イベント通知によりアプリケーションに完了が知らされます。 イベントが通知されたら、 1 次戻りコードおよび 2 次戻りコードを調べて、エラー条件の有無を確認してください。 **関連情報:** 207ページの『WinRUI』

これは、OS/2 の下でサポートされる唯一のエントリー・ポイントです。

WinRUI

すべての RUI verb について非同期メッセージ通知を提供します。

```
int WINAPI WinRUI (HWND hWnd,
                   LUA_VERB_RECORD* vcb);
```

パラメーター

説明

hwnd 完了メッセージを受け取るウィンドウ・ハンドル。

vcb verb 制御ブロックへのポインター。

この関数は、RUI によって処理要求が受け入れられたかどうかを示す値を戻します。戻り値 0 は、要求が受け入れられ処理されることを示します。 0 以外の値はエラーを示します。 エラー・コードには次のようなものがあります。

WLUAINVALIDHANDLE

提供されたウィンドウ・ハンドルが無効です。

lua_flag2.async に戻される値は、非同期通知が生じるかどうかを示します。 このフラグが（ゼロ以外に）セットされている場合は、アプリケーションのメッセージ待ち行列に通知されるメッセージにより非同期通知が行われます。 このフラグがセットされていない場合は、要求は同期的に完了します。 1 次戻りコードおよび 2 次戻りコードを調べて、エラーの有無を確認してください。

verb が完了した時点で、アプリケーションのウィンドウ *hWind* は、**WinRUI** を入力ストリングとする **RegisterWindowMessage** から戻されたメッセージを受け取ります。 **lParam** 引数には、完了したと通知される VCB のアドレスが入ります。*wParam* 引数は未定義です。 処理要求が受け入れられる可能性がありますが（関数呼出しが 0 を戻した場合）、後で拒否され、VCB 内に 1 次戻りコードおよび 2 次戻りコードがセットされることもあります。 1 次戻りコードおよび 2 次戻りコードを調べて、エラーの有無を確認してください。

アプリケーションが、最初に **WinRUIStartup** によりセッションを初期設定しないで **WinRUI** を呼び出した場合は、エラーが戻されます。

関連情報: 206ページの『RUI()』

WinRUICleanup()

アプリケーションを終了し RUI API からアプリケーションの登録を取り消します。

```
BOOL WINAPI WinRUICleanup (void);
```

戻り値は、登録取消しが成功したか失敗したかを示します。 値が 0 以外である場合は、アプリケーションの登録は正常に取り消されています。値が 0 の場合は、アプリケーションの登録は解除されていません。

WinRUICleanup は、RUI API の登録を取り消すため、たとえば特定のアプリケーションに割り振られている資源を解放するために使用します。

LU がセッション中にあるとき（**RUI_TERM** が発行されていないとき）に **WinRUICleanup** が呼び出された場合は、すべてのオープン・セッションについて、そのアプリケーションに対する **RUI_TERM**（クローズ・タイプは ABEND）を発行します。**関連情報:** 213ページの『WinRUIStartup()』

WinRUIGetLastInitStatus()

これは、サーバーにのっているアプリケーションに対してだけサポートされます。

この関数は、アプリケーションが **RUI_INIT** の状況を判別して、**RUI_INIT** のタイムアウトが生じていないかどうかを確認するための手段として使用できます。 この呼出しは、状況報告を開始するため、状況報告を終了するため、または現在の状況を判別するために使用します。 詳細については「使用上の注意」の項を参照してください。

```
int WINAPI WinRUIGetLastInitStatus (DWORD dwSid,
                                    HANDLE hStatusHandle,
                                    DWORD dwNotifyType,
                                    BOOL bClearPrevious);
```

パラメーター

説明

dwSid 状況を確認したいセッションのセッション識別子。この値が `0` の場合は、*hStatusHandle* はすべてのセッションの状況を報告します。 **RUI_INIT** を対象とした **RUI**() または **WinRUI**() の呼出しから戻った時点で、ただちに **RUI_INIT** VCB 内の *lua_sid* が有効になります。

hStatusHandle

セッションの状況が変化したことをアプリケーションに通知するために使用されるハンドル。 ウィンドウ・ハンドル、イベント・ハンドル、または NULL のいずれかです。これに応じて *dwNotifyType* を設定する必要があります。

- *hStatusHandle* がウィンドウ・ハンドルである場合は、状況はウィンドウ・メッセージを使用してアプリケーションに送られます。 プログラムは、ストリング **WinRUI** を使用して **RegisterWindowMessage** からメッセージを受け取ります。 セッション状況は、パラメーター wParam に入ります（「戻り値」を参照）。 *dwNotifyType* の値に応じて、**lParam** には、セッションの RUI セッション ID か、または、**RUI_INIT** verbの **lua_correlator** の値が入ります。
- *hStatusHandle* がイベント・ハンドルである場合は、*dwSid* に指定したセッションの状況が変化すると、イベントが通知済みの状態になります。 アプリケーションはさらに **WinRUIGetLastInitStatus()** の呼出しを発行して、新しい状況を確認する必要があります。 このイベントは、**RUI** verb の完了を通知するために使用するイベントと同じであってはなりません。
- *hStatusHandle* が NULL の場合は、*dwSid* に指定したセッションの状況が戻りコードに入れて戻されます。この場合は、*bClearPrevious* が TRUE でな

い限り、*dwSid* は 0 であってはなりません。 *hStatusHandle* が NULL の場合は、*dwNotifyType* は無視されます。

dwNotifyType
要求される指示のタイプ。 これは、ウィンドウ・メッセージの **lParam** の内容、および、**WinRUIGetLastInitStatus()** が*hStatusHandle* をどのように解釈するかを決定します。 使用できる値は次の通りです。

WLUA_NTFY_EVENT
hStatusHandle パラメーターにはイベント・ハンドルが入ります。

WLUA_NTFY_MSG_CORRELATOR
hStatusHandle パラメーターにはウィンドウ・ハンドルが入り、戻されるウィンドウ・メッセージの **lParam** には LUA 相関係数およびRUI が入ります。

WLUA_NTFY_MSG_SID
hStatusHandle パラメーターにはウィンドウ・ハンドルが入り、戻されるウィンドウ・メッセージの **lParam** には LUA セッション識別子が入ります。

bClearPrevious
TRUE の場合は、*dwSid* により識別されるセッションについては状況メッセージは送られません。 *dwSid* が `0` の場合は、どのセッションについても状況メッセージは送られません。 *bClearPrevious* が TRUE の場合は、*hStatusHandle* および *dwNotifyType* は無視されます。

WLUASYSNOTREADY
Communications Server が実行していません。

WLUANTFYINVALID
dwNotifyType パラメーターが無効です。

WLUAINVALIDHANDLE
hStatusHandle パラメーターに有効なハンドルが含まれていません。

WLUALINKINACTIVE
ホストへのリンクがまだ活動状態にありません。

WLUAPUINACTIVE
ホストへのリンクは活動状態ですが、**ACTPU** が受信されていません。

WLUAPUACTIVE
ACTPU が受信されました。

WLUAPUREACTIVATED
PU が再活動化されました。

WLUAUINACTIVE
ホストへのリンクが活動状態にあり、**ACTPU** が受信されましたが、**ACTLU** が受信されていません。

WLUALUACTIVE
LU が活動状態にあります。

WLUAUNKNOWN
セッションの状況が不明です （これは内部エラーです)。

WLUASIDINVALID
指定された SID は、**RUI** が認識しているどれにも一致しません。

WLUASIDZERO
hStatusHandle パラメーターが NULL で、*bClearPrevious* が FALSE ですが、*dwSid* は `0` です。

WLUAGLOBALHANDLER
dwSid パラメーターが `0` で、すべてのセッションからのメッセージが通知されます。 （これは正常戻りコードであり、エラーではありません)。

この関数は、ウィンドウ・ハンドルまたはイベント・ハンドルを使用して、状況の変化を非同期に通知するためのものですが、これだけでセッションの現行状況を知ることもできます。

この関数をウィンドウ・ハンドルで使用するには、以下の 2 つの方法のいずれかを使用してください。

```
WinRUIGetLastInitStatus(Sid,Handle,WLUA_NTFY_MSG_CORRELATOR,FALSE);
```

または

```
WinRUIGetLastInitStatus(Sid,Handle,WLUA_NTFY_MSG_SID,FALSE);
```

ここでは、状況の変化がウィンドウ・メッセージによって報告され、指定したウィンドウ・ハンドルに送られます。 WLUA_NTFY_MSG_CORRELATOR を指定した場合は、ウィンドウ・メッセージの **lParam** フィールドには、セッションの **lua_correlator** フィールドが入ります。 WLUA_NFTY_MSG_SID を指定した場合は、ウィンドウ・メッセージの **lParam** フィールドには、セッションの LUA セッション識別子が入ります。

この関数をウィンドウ・ハンドルとともに使用した場合は、状況の報告を取り消すには次のコマンドを使用します。

```
WinRUIGetLastInitStatus(Sid,NULL,0,TRUE);
```

ここでは、*Sid* が 0 以外の場合は、状況はそのセッションについてのみ報告されるという点に注意してください。 *Sid* が `0` の場合は、すべてのセッションの状況が報告されます。

この関数をイベント・ハンドルとともに使用するためには、次を発行します。

```
WinRUIGetLastInitStatus(Sid,Handle,WLUA_NOTIFY_EVENT,FALSE);
```

WinRUIGetLastInitStatus()

状況変化が生じると、指定したハンドルに該当するイベントが通知されます。 イベントの通知では情報は戻されないので、状況を知るには次の呼出しを発行する必要があります。

```
Statu = WinRUIGetLastInitStatus(Sid,NULL,0,0,FALSE);
```

この場合は、*Sid* を指定する必要があります。

この関数をイベント・ハンドルとともに使用した場合は、状況の報告を取り消すには次のコマンドを使用します。

```
WinRUIGetLastInitStatus(Sid,NULL,0,TRUE);
```

この関数を使用してセッションの現行状況を照会するには、イベント・ハンドルまたはウィンドウ・ハンドルは必要ありません。 代わりに次のコマンドを使用します。

```
Status = WinRUIGetLastInitStatus(Sid,NULL,0,0,FALSE);
```

WinRUIStartup()

アプリケーションで、必要な RUI API のバージョンを指定し、API の詳細情報を検索することができます。

```
int WINAPI WinRUIStartup (WORD wVersionRequired,
                          LPWLUADATA* luadata);
```

パラメーター

説明

wVersionRequired

必要な RUI API サポートのバージョンを指定します。 高位バイトはリリース番号（改訂番号）を示し、低位バイトはバージョン番号を示します。

luadata

RUI のバージョンを戻します。

戻り値は、アプリケーションが正常に登録されたかどうか、および、RUI API が指定のバージョン番号をサポートできるかどうかを示します。 値が `0` の場合は、アプリケーションは正常に登録されていて、指定したバージョンがサポートされています。 その他の場合は、戻り値は次のいずれかです。

WLUAVERNOTSUPPORTED

要求した RUI API サポートのバージョンは、この特定 RUI API では提供されていません。

WLUAINVALID

要求したバージョンを判別できませんでした。

この呼出しは、API の将来のバージョンとの互換性を確保することを目的とするものです。 現在のバージョンは 1.0 です。 **関連情報:** 208ページの『WinRUICleanup()』

GetLuaReturnCode()

VCB 内の 1 次戻りコードおよび 2 次戻りコードを、印刷可能ストリングに変換します。 この関数は、LUA アプリケーションが使用するための標準のエラー・ストリングを提供します。

```
int WINAPI GetLuaReturnCode (lua_common* vcb,
                             UINT buffer_length,
                             unsigned char* buffer_addr);
```

パラメーター

説明

vcb 指定パラメーター。verb 制御ブロックのアドレスを指定します。

buffer_length
指定パラメーター。buffer_addr が指すバッファーの長さを指定します。 この長さの推奨値は 256 です。

buffer_addr
指定/戻りパラメーター。NULL 文字で終了する定型ストリングが入るバッファーのアドレスを指定します。指定したバッファー内のストリングの長さが戻されます。

0x20000001
パラメーターが無効です。この関数は、指定した verb 制御ブロックからの読取り、または指定したバッファーへの書込みができませんでした。

0x20000002
指定したバッファーが小さすぎます。

buffer_addr に戻されたエラー・ストリングは、改行文字 (**/n**) で終わっていません。

呼出しの例

次の例は、**WINRUI32.DLL** を呼び出す方法を示しています。この DLL のヘッダー・ファイルは **WINLUA.H** です。例では、プログラムから **RUI** を発行するために、**RUI DLL** を呼び出します。

```
#include "WINLUA.H"                          /* LUA C include file for
                                                the LUA Application. */
. . .

. . .

example()
{
LUA_VERB_RECORD    VerbRecord;               /* Declare VerbRecord as a verb
                                                control block using the
                                                TYPEDEF in WINLUA.H */

. . .
```

```
WINRUI((LUA_VERB_RECORD *) &VerbRecord);      /* Call the RUI API */
. . .
}
```

GetLuaReturnCode()

第13章 LUA verb

この章には次の情報を収めてあります。

- LUA 共通制御ブロック構造の詳細
- すべての LUA verb および LUA verb レコードの説明

各 LUA verb について次の情報を示します。

- verb の目的。
- LUA の指定パラメーターと戻りパラメーター。 各パラメーターの説明には、パラメーターの有効値に関する情報のほか、その他の必要な追加情報が含まれています。
- 他の verb との相互作用。
- verb の使用に関する追加情報。

注: *予約済み* と示されているパラメーターは、常に 0 にセットされます。

LUA verb 制御ブロックのフォーマット

verb 制御ブロックは次のものから成っています。

- **lua_common**。これはすべての verb に使用されるもので、217ページの『共通verb ヘッダー』の項で説明します。
- **specific**。これは **RUI_BID** verb のみに使用されます（222ページの『RUI_BID データ構造』で説明します）。

構造は次のように定義されます。

```
typedef struct lua_verb_record
{
 LUA_COMMON        common;                 /* The common verb header     */
 union {
   unsigned char   lua_peek_data[12];      /* field specific to RUI_BID  */
 }
} LUA_VERB_RECORD;
```

共通 verb ヘッダー

Communications Server LUA は、汎用の*共通 verb* ヘッダーを使用して、すべての着信データと発信データをトランスポートします。 verb 制御ブロック内のフィールドは次のように定義されます。

```
typedef struct lua_common
{
 unsigned short    lua_verb;           /* LUA_VERB_RUI               */
 unsigned short    lua_verb_length;    /* VCB length                 */
 unsigned short    lua_prim_rc;        /* primary return code        */
 unsigned long     lua_sec_rc;         /* secondary return code      */
```

```
 unsigned short    lua_opcode;          /* verb opcode                */
 unsigned long     lua_correlator;      /* verb correlator            */
 unsigned char     lua_luname[8];       /* local LU name              */
 unsigned short    lua_extension_list_offset;
                                        /* reserved                   */
 unsigned short    lua_cobol_offset;    /* reserved                   */
 unsigned long     lua_sid;             /* session ID                 */
 unsigned short    lua_max_length;      /* max buffer length          */
 unsigned short    lua_data_length;     /* actual data length         */
 unsigned char    *lua_data_ptr;        /* data pointer               */
 unsigned long     lua_post_handle;     /* post handle                */
 LUA_TH            lua_th;              /* TH structure               */
 unsigned char     lua_rh;              /* message RH                 */
 unsigned char     lua_flag1;           /* application message flag   */
 unsigned char     lua_message_type;    /* SNA message type           */
 unsigned char     lua_flag2;           /* LUA message flag           */
 unsigned char     lua_resv56[7];       /* reserved                   */
 unsigned char     lua_encr_decr_option;/* cryptography               */
} LUA_COMMON;

typedef struct lua_th
{
 unsigned char     reserv1;             /* reserved                   */
 unsigned char     daf;                 /* DAF                        */
 unsigned char     oaf;                 /* OAF                        */
 unsigned char     snf[2];              /* SNF                        */
} LUA_TH;
```

lua_verb

LUA verb であることを識別します。これは常に **LUA_VERB_RUI** にセットされます。

lua_verb_length

verb 制御ブロックの長さ。

lua_prim_rc

LUA がセットする 1 次戻りコード。

lua_sec_rc

LUA がセットする 2 次戻りコード。

lua_opcode

verb 操作コード。これは、**LUA** verb を発行しようとしていることを示します。

lua_correlator

４バイトの相関係数。これは、この verb をアプリケーション内の他の処理に関連付けるために使用できます。 LUA はこのパラメーターを使用しません。

lua_luname

LUA セッションで使用するローカル LU 名（ASCII)。 これは LU 名またはLU プール名です。詳細は、229ページの『RUI_INIT』を参照してください。

lua_sid
この verb を発行する LUA セッションのセッション ID。

lua_max_length
データの受信に使用するバッファーの長さ。

lua_data_length
送信するデータの長さ、または受信したデータの実際の長さ。

lua_data_ptr
送信するデータ、またはデータを受信するデータ・バッファーを指すポインター。

lua_th.flags
伝送ヘッダー内でセットするフラグを指定します （詳細は、*Systems Network Architecture Formats* を参照）。これは、次の値のどれか 1 つか、または、いくつかを OR で結んだものです。

LUA_FID
フォーマット識別タイプ 2

LUA_MPF
セグメント化マッピング・フィールド

LUA_BBIU
開始 BIU

LUA_EBIU
終了 BIU

LUA_ODAI
OAF-DAF 割当て標識

LUA_EFI
急送フロー標識

lua_th.daf
DAF（宛先アドレス・フィールド）。

lua_th.oaf
OAF（起点アドレス・フィールド）。

lua_th.snf
順序番号フィールド。

lua_rh 送信または受信されるメッセージの要求応答ヘッダー (RH) を指定します。（詳細は、*Systems Network Architecture Formats* を参照）。これは、次の値のどれか 1 つか、または、いくつかを OR で結んだものです。

LUA_RRI
要求応答標識

LUA_RH_FMD
RU カテゴリー: FMI データ・セグメント

LUA_RH_NC
RU カテゴリー: ネットワーク制御

LUA_RH_DFC
RU カテゴリー: データ・フロー制御

LUA_RH_SC
RU カテゴリー: セッション制御

LUA_FI
フォーマット標識

LUA_SDI
センス・データ組込み標識

LUA_BCI
連鎖開始標識

LUA_ECI
連鎖終了標識

LUA_DR1I
確定応答 1 標識

LUA_DR2I
確定応答 2 標識

LUA_RI
例外応答標識 (要求の場合)、または応答タイプ標識 (応答の場合)

LUA_QRI
待ち行列応答標識

LUA_PI
ペーシング標識

LUA_BBI
ブラケット開始標識

LUA_EBI
ブラケット終了標識

LUA_CDI
方向変換標識

LUA_CSI
コード選択標識

LUA_EDI
暗号化データ標識

LUA_PDI
埋込みデータ標識

lua_flag1
アプリケーションが提供するメッセージに関するフラグを指定します （詳

細は、*Systems Network Architecture Formats* を参照)。フラグは、次の値のどれか 1 つ、またはいくつかを OR で結んだものです。

LUA_BID_ENABLE
送信権要求使用可能標識

LUA_NOWAIT
データ待機なしフラグ

LUA_SSCP_EXP
SSCP 急送フロー

LUA_SSCP_NORM
SSCP 通常フロー

LUA_LU_EXP
LU 急送フロー

LUA_LU_NORM
LU 通常フロー

LUA_CLOSE_ABEND

LUA_RESERVE1

lua_message_type
RUI_READ verb が受信した（または **RUI_BID** verb に対して指示された）SNA メッセージのタイプ。 これは次のいずれかの値です。

LUA_MESSAGE_TYPE_LU_DATA

LUA_MESSAGE_TYPE_SSCP_DATA

LUA_MESSAGE_TYPE_RSP

LUA_MESSAGE_TYPE_BID

LUA_MESSAGE_TYPE_BIND

LUA_MESSAGE_TYPE_BIS

LUA_MESSAGE_TYPE_CANCEL

LUA_MESSAGE_TYPE_CHASE

LUA_MESSAGE_TYPE_CLEAR

LUA_MESSAGE_TYPE_CRV

LUA_MESSAGE_TYPE_LUSTAT_LU

LUA_MESSAGE_TYPE_LUSTAT_SSCP

LUA_MESSAGE_TYPE_QC

LUA_MESSAGE_TYPE_QEC

LUA_MESSAGE_TYPE_RELQ

LUA_MESSAGE_TYPE_RQR

LUA_MESSAGE_TYPE_RTR

LUA_MESSAGE_TYPE_SBI

LUA_MESSAGE_TYPE_SHUTD

LUA_MESSAGE_TYPE_SIGNAL

LUA_MESSAGE_TYPE_SDT

LUA_MESSAGE_TYPE_STSN

LUA_MESSAGE_TYPE_UNBIND

lua_flag2

LUA が戻すメッセージに関するフラグを指定します （詳細は、*Systems Network Architecture Formats* を参照)。 フラグは、次の値のどれか 1 つ、またはいくつかを OR で結んだものです。

LUA_BID_ENABLE

送信権要求使用可能標識

LUA_ASYNC

非同期 verb 完了フラグ

LUA_SSCP_EXP

SSCP 急送フロー

LUA_SSCP_NORM

SSCP 通常フロー

LUA_LU_EXP

LU 急送フロー

LUA_LU_NORM

LU 通常フロー

lua_encr_decr_option

暗号オプション。

RUI_BID データ構造

次のパラメーターは **RUI_BID** verb に固有のものであり、この verb の場合のみ提供されます。

lua_peek_data

読取りを待っている最大 12 バイトのデータ。

RUI_BID

RUI_BID verb は、受信されたメッセージが読取りを待っていることを、RUI アプリケーション・プログラムに知らせるために使用します。 これによって、アプリケーションは、**RUI_READ** verb を発行する前に、どのようなデータが使用可能かを判別することができます。 使用可能なメッセージがある場合は、**RUI_BID** verb は、戻り時に、受信されたメッセージ・フローの詳細、メッセージ・タイプ、メッセージの TH と RH、および最大 12 バイトのメッセージ・データを示します。 **RUI_BID** と **RUI_READ** の主な違いは、**RUI_BID** では、アプリケーションはデータを着信メッセージ待ち行列から除去せずに検査できるので、データをそのまま残しておいて後でまたアクセスできるという点にあります。 **RUI_READ** は、待ち行列からメッセージを除去してしまうので、アプリケーションは、いったん読み取ってしまったらそのデータを処理しなければなりません。

指定パラメーター: アプリケーションは次のパラメーターを提供します。

lua_verb
LUA_VERB_RUI

lua_verb_length
LUA verb レコードの長さ（バイト数）。 これは次のようにセットします。

```
sizeof(struct LUA_COMMON) + 12.
```

lua_opcode
LUA_OPCODE_RUI_BID

lua_correlator
任意選択。 この verb をアプリケーション内の他の処理に関連付けるために使用できる 4 バイトの値です。 LUA はこの情報を使用せず、変更もしません。

lua_luname
セッションで使用するローカル LU の ASCII 名。 これは、活動 LUA セッションの LU 名に一致している必要があります。

このパラメーターが必要なのは、**lua_sid** パラメーターが 0 の場合だけです。 **lua_sid** にセッション ID を指定した場合は、LUA はこのパラメーターを使用しません。

このパラメーターは長さが 8 バイトでなければなりません。名前が 8 文字に満たない場合は、右側をスペース (0x20) で埋めてください。

lua_sid
セッションのセッション ID。 これは、前の **RUI_INIT** verb で戻されたセッション ID に一致していなければなりません。 このパラメーターは任意選択です。ここでセッション ID を指定しない場合は、**lua_luname** パラメーターにセッションの LU 名を指定する必要があります。

lua_post_handle
これは、非同期 verb の完了を通知するために使用する 4 バイトのハンドルです。

戻りパラメーター: 次のパラメーターは常に戻されます。

lua_flag2
これは、verb が非同期に完了した場合に限り LUA_ASYNC にセットされます。

その他の戻りパラメーターは、verb が正常に完了したかどうかによって異なります。次の項を参照してください。

verb が正常に完了した場合は、次のパラメーターが戻されます。

lua_prim_rc
LUA_OK

lua_sid
この verb を発行するときに、アプリケーションでセッション ID ではなく **lua_luname** パラメーターを指定してある場合は、LUA はセッション ID を提供します。

lua_max_length
受信したメッセージ内のデータのバイト数。

lua_data_length
lua_peek_data パラメーターに戻されたデータのバイト数（ 0 〜12）。

lua_th 受信したメッセージの伝送ヘッダー (TH) からの情報。

lua_rh 受信したメッセージの要求/応答ヘッダー (RH) からの情報。

lua_message_type
受信したメッセージのメッセージ・タイプ。これは次の値のいずれかです。

LUA_MESSAGE_TYPE_LU_DATA
LUA_MESSAGE_TYPE_SSCP_DATA
LUA_MESSAGE_TYPE_RSP
LUA_MESSAGE_TYPE_BID
LUA_MESSAGE_TYPE_BIND
LUA_MESSAGE_TYPE_BIS
LUA_MESSAGE_TYPE_CANCEL
LUA_MESSAGE_TYPE_CHASE
LUA_MESSAGE_TYPE_CLEAR
LUA_MESSAGE_TYPE_CRV
LUA_MESSAGE_TYPE_LUSTAT_LU
LUA_MESSAGE_TYPE_LUSTAT_SSCP
LUA_MESSAGE_TYPE_QC
LUA_MESSAGE_TYPE_QEC
LUA_MESSAGE_TYPE_RELQ
LUA_MESSAGE_TYPE_RTR

LUA_MESSAGE_TYPE_SBI

LUA_MESSAGE_TYPE_SHUTD

LUA_MESSAGE_TYPE_SIGNAL

LUA_MESSAGE_TYPE_SDT

LUA_MESSAGE_TYPE_STSN

LUA_MESSAGE_TYPE_UNBIND

lua_flag2

次のいずれかのフラグがセットされて、データがどのメッセージ・フローで受信されたのかを示します。

LUA_SSCP_EXP

SSCP 急送フロー

LUA_LU_EXP

LU 急送フロー

LUA_SSCP_NORM

SSCP 通常フロー

LUA_LU_NORM

LU 通常フロー

lua_peek_data

メッセージ・データの最初の 12 バイト（または、12 バイトより短い場合はメッセージ・データのすべて）。

次の戻りコードは、他の verb によって取り消されたことが原因で、この verb が正常に完了しなかったことを示します。

lua_prim_rc

LUA_CANCELLED

lua_sec_rc

LUA_TERMINATED

この verb が保留状態にあるときに **RUI_TERM** verb が発行されました。

次の戻りコードは、指定パラメーターのどれかにエラーがあったために、この verb が正常に完了しなかったことを示します。

lua_prim_rc

LUA_PARAMETER_CHECK

lua_sec_rc

可能な値は次のとおりです。

LUA_BID_ALREADY_ENABLED

前の **RUI_BID** verb が未完了状態にあるために、この **RUI_BID** verb が拒否されました。 未完了状態にできる **RUI_BID** は一度に 1 つだけです。

LUA_RESERVED_FIELD_NOT_ZERO
verb レコード内の予約フィールド、またはこの verb では使用されないパラメーターが、0 以外の値にセットされています。

LUA_VERB_LENGTH_INVALID
lua_verb_length パラメーターの値が、この verb に必要な verb レコードの長さに達していません。

次の戻りコードは、セッション状態がこの verb にとって無効であるときに、このverbが発行されたことを示します。

lua_prim_rc
LUA_STATE_CHECK

lua_sec_rc
LUA_NO_RUI_SESSION

RUI_INIT verb がこのセッションではまだ正常に完了していないか、またはセッションで障害が発生しました。

次の戻りコードは、指定した verb レコードは有効であったが、verb が正常に完了しなかったことを示します。

lua_prim_rc
LUA_UNSUCCESSFUL

lua_sec_rc
LUA_INVALID_PROCESS

この verb を発行したアプリケーション・インスタンスは、このセッションで **RUI_INIT** verb を発行したインスタンスと同じではありません。

次の戻りコードは、Communications Serverが、ホストから受信したデータにエラーを検出したことを示します。Communications Serverは、受信したメッセージを **RUI_READ** verb 上のアプリケーションに渡す代わりに、そのメッセージ（そのメッセージが連鎖の一部である場合は連鎖の残りの部分）を破棄し、ホストに否定応答を送ります。 LUA は、後続の **RUI_READ** verb または **RUI_BID** verb 上のアプリケーションに、否定応答が送られたことを知らせます。

lua_prim_rc
LUA_NEGATIVE_RSP

lua_sec_rc
２次戻りコードには、否定応答とともにホストに送られたセンス・コードが含まれています。 戻されるセンス・コード値の解釈の仕方は、175ページの『SNA の層』を参照してください。

0 の 2 次戻りコードは、連鎖の途中のメッセージに対する否定応答の **RUI_WRITE** の後で、Communications Server がその連鎖からすべてのメッセージを受け取り、そして破棄したことを示します。

次の１次戻りコードおよび２次戻りコードは、その他の理由で verb が正常に完了しなかったことを示します。

lua_prim_rc

LUA_SESSION_FAILURE

セッションがダウンしました。

lua_sec_rc

可能な値は次のとおりです。

LUA_LU_COMPONENT_DISCONNECTED

通信リンクまたはホスト LU に関する問題が原因で、LUA セッションが失敗しました。

LUA_RUI_LOGIC_ERROR

この戻りコードは次のいずれかを示します。

- ホスト・システムが SNA プロトコルに違反した。
- は LUA で内部エラーが検出されたことを示します。

トレースを活動状態にして問題を再現し、ホストが正しいデータを送っているかどうかを確認してください。

lua_prim_rc

LUA_INVALID_VERB

lua_verb パラメーターまたは **lua_opcode** パラメーターのいずれかが無効でした。 verb は実行されませんでした。

lua_prim_rc

LUA_UNEXPECTED_DOS_ERROR

資源不足などのオペレーティング・システム・エラーが起こりました。

lua_sec_rc

この値はオペレーティング・システムの戻りコードです。 この戻りコードの意味については、オペレーティング・システムの資料を参照してください。

他の verb との相互作用: この verb を発行するには、その前に **RUI_INIT** verb が正常に完了していなければなりません。

未完了状態にできる **RUI_BID** は一度に 1 つだけです。**RUI_BID** verb が正常に完了した後で、再度この verb を発行するには、後続の **RUI_READ** verb の LUA_BID_ENABLE に **lua_flag1** をセットします。 この方法でこの verb を再発行する場合、アプリケーション・プログラムは、**RUI_BID** verb レコードに関連した記憶域を解放または変更してはなりません。

RUI_READ および **RUI_BID** の両方が未完了状態にあるときにホストからメッセージが到着した場合は、**RUI_READ** は完了し、**RUI_BID** は進行中のままとなります。

使用上の注意: 到着する各メッセージは、それぞれ 1 回だけ送信権要求されます。**RUI_BID** verb が、特定のセッション・フロー上でデータが待機状態にあることを示したら、アプリケーションは、**RUI_READ** verb を発行してそのデータを受信する必

要があります。 **RUI_READ** verb を発行することにより、送信権要求されたメッセージが受け入れられるまでは、以後の **RUI_BID** では、そのセッション・フローでのデータの到着は報告されません。

一般に、この verb で戻される **lua_data_length** パラメーターは、**lua_peek_data** の中のデータの長さを示すだけで、待機しているメッセージのデータの合計長を示すものではありません（12 バイトに満たない値が戻された場合を除きます）。
lua_max_length パラメーターは、受信したメッセージのバイト数を戻します。アプリケーションは、データを受け入れる **RUI_READ** verb でのデータ長が、メッセージを収容するのに十分な長さであることを確認する必要があります。

RUI_INIT

RUI_INIT verb は、指定された LUA LU のための SSCP-LU セッションを確立します。

指定パラメーター: アプリケーションは次のパラメーターを提供します。

lua_verb

LUA_VERB_RUI

lua_verb_length

LUA verb レコードの長さ（バイト数）。 これは sizeof(struct LU_COMMON) に設定します。

lua_opcode

LUA_OPCODE_RUI_INIT

lua_correlator

任意選択。 この verb をアプリケーション内の他の処理に関連付けるために使用できる 4 バイトの値です。 LUA はこの情報を使用せず、変更もしません。

lua_luname

セッションを開始する ローカル LU または LU プールの ASCII での名称。 これは、構成されている LUA LU 名または LU プール名と一致している必要があります。 Communications Server のアプリケーションでは、この名前は次のように使用されます。

この名前がプール内にはない名前である場合は、Communications Serverは、この LU を使用してセッションを開始しようとします。

この名前が LU プールの名前であるか、またはプール内の LU の名前である場合は、Communications Server はプール内で最初に使用可能になった LU をセッションを開始しようとします。 このフィールドは 8 バイトの ASCII ストリングで、8 バイトに満たない場合は、後ろにスペース文字 (0x20) が埋め込まれます。

SNA API クライアントのアプリケーションでは、この名前は、構成されている LUA セッション名に一致する必要があります。

lua_post_handle

これは、非同期 verb の完了を通知するために使用する 4 バイトのハンドルです。

lua_flag1

RUI_INIT verb を処理するときに、Communications Server から RUI_INIT_STATUS 指示を受け取るには、アプリケーションはこれを LUA_ASYNC_STATUS にセットする必要があります （RUI_INIT_STATUS メッセージについては、238ページの『RUI_INIT_STATUS』で説明します）。

lua_encr_decr_option

セッション・レベルの暗号化オプション。 Communications Serverは次の2つの値を受け入れます。

0　　セッション・レベルの暗号化を使用しません。
128　　暗号化および暗号解読がアプリケーションにより行われます。

その他の値を指定した場合は、戻りコード LUA_ENCR_DECR_LOAD_ERROR が戻されます （ユーザー定義の暗号化および暗号解読ルーチンを示す 1 〜127 の範囲内の値は、OS/2 コミュニケーション・マネージャー/2 の LUA ではサポートされていますが、CommunicationsServerではサポートされていません）。

戻りパラメーター: 次のパラメーターは常に戻されます。

lua_flag2
これは、verb が非同期に完了した場合に限り LUA_ASYNC にセットされます。

注: RUI_INIT は、LUA_PARAMETER_CHECK などのようなエラーを戻す場合以外は、非同期に完了します。

その他の戻りパラメーターは、verb が正常に完了したかどうかによって異なります。次の項を参照してください。

verb が正常に実行された場合は、LUA は次のパラメーターを戻します。

lua_prim_rc
LUA_OK

lua_sid
新しいセッションのセッション ID。 これは、後続の verb でこのセッションを識別するために使用されます。

lua_luname
セッションで使用するローカル LU の名前。 これが必要なのは、アプリケーションが LU プールを指定していて、プール内のどの LU がすでに使用されているかを知る必要がある場合です。

次の戻りコードは、他の verb によって取り消されたことが原因で、この verb が正常に完了しなかったことを示します。

lua_prim_rc
LUA_CANCELLED

lua_sec_rc
LUA_TERMINATED

RUI_TERM verb が発行されたのは、**RUI_INIT** が完了する前でした。

次の戻りコードは、指定パラメーターのどれかにエラーがあったために、この verb が正常に完了しなかったことを示します。

lua_prim_rc
LUA_PARAMETER_CHECK

lua_sec_rc

可能な値は次のとおりです。

LUA_INVALID_LUNAME

lua_luname パラメーターが見つかりませんでした。 *Communications Server システム管理プログラミング* の API で、LU 名 または LU プール名が定義されていることを確認してください。

LUA_RESERVED_FIELD_NOT_ZERO

verb レコード内の予約フィールド、またはこの verb では使用されないパラメーターが、0 以外の値にセットされています。

LUA_VERB_LENGTH_INVALID

lua_verb_length パラメーターの値が、この verb に必要な verb レコードの長さに達していません。

次の戻りコードは、セッション状態がこの verb にとって無効であるときに、このverb が発行されたことを示します。

lua_prim_rc

LUA_STATE_CHECK

lua_sec_rc

LUA_DUPLICATE_RUI_INIT

このアプリケーションによってすでに使用されている (または、このアプリケーションがすでに **RUI_INIT** verb を進行させている) LU 名または LU プール名が指定された **lua_luname** パラメーター。

次の戻りコードは、指定した verb レコードは有効であったが、verb が正常に完了しなかったことを示します。

lua_prim_rc

LUA_UNSUCCESSFUL

lua_sec_rc

可能な値は次のとおりです。

LUA_COMMAND_COUNT_ERROR

verb で、LU プールの名前またはプール内の LU の名前を指定しましたが、プール内のすべての LU が使用中です。

LUA_ENCR_DECR_LOAD_ERROR

verb で、**lua_encr_decr_option** に 0 または 128 以外の値を指定しました。

LUA_INVALID_PROCESS

lua_luname パラメーターに指定した LU が他のプロセスで使用中です。

LUA_LINK_NOT_STARTED

ホストへのリンクが開始されていません。

次に示す **lua_sec_rc** の値は Communications Server のセンス・コードであり、**lua_prim_rc** が LUA_UNSUCCESSFUL である場合に戻されます（これらの値は LU の状態を反映します）。

X10020000
ACTPU が受信されていません。 **RUI_INIT** は PU を活動化しません。

X10100000
ACTPU が受信されていません。 **RUI_INIT** は PU を活動化します。

X10110000
ACTPU が受信されました。 **ACTLU** は受信されていません。 SSCP は、自己定義従属 LU (SSDLU) をサポートしません。 **RUI_INIT** は LU を活動化します。

X10120000
ACTPU が受信されました。 **ACTLU** は受信されていません。 SSCP は SSDLU をサポートします。 **RUI_INIT** は LU を活動化します。

次の 1 次戻りコードおよび 2 次戻りコードは、その他の理由で verb が正常に完了しなかったことを示します。

lua_prim_rc
LUA_SESSION_FAILURE

セッションがダウンしました。

lua_sec_rc
LUA_LU_COMPONENT_DISCONNECTED

通信リンクまたはホスト LU に関する問題が原因で、LUA セッションが失敗しました。

lua_prim_rc
LUA_INVALID_VERB

lua_verb パラメーターまたは **lua_opcode** パラメーターのいずれかが無効でした。 verb は実行されませんでした。

lua_prim_rc
LUA_UNEXPECTED_DOS_ERROR

資源不足などのオペレーティング・システム・エラーが起こりました。

lua_sec_rc
この値はオペレーティング・システムの戻りコードです。 この戻りコードの意味については、オペレーティング・システムの資料を参照してください。

他の verb との相互作用: この verb は、セッションで発行する最初の LUA verb でなければなりません。 この verb が完了するまでは、このセッションで発行できる他の LUA verb は、**RUI_TERM**（これは保留中の **RUI_INIT** を終了させます）だけです。このセッションで発行するその他のすべての verb は、次に示すこの verb のパラメーターのどちらかを使用して、セッションを識別する必要があります。

- セッション ID は **lua_sid** パラメーターにアプリケーションに戻されます。

- LU 名は、アプリケーションが **lua_luname** パラメーターに指定します。

使用上の注意: **RUI_INIT** verb は、ホストからの **ACTLU** を受信した後に完了します。必要な場合は、この verb は無制限に待機することになります。**RUI_INIT** verb より前に **ACTLU** が受信されている場合は、LUA はホストに **NOTIFY** を送って、LU がすでに使用可能な状態にあることを知らせます。

注: **ACTLU** および **NOTIFY** のどちらも、LUA アプリケーションには見えません。

RUI_INIT verb が正常に完了すると、このセッションは、セッション開始の対象となった LU を使用します。 他の LUA セッション（このアプリケーション、または他のアプリケーションのどちらからのものであっても）は、**RUI_TERM** verb が発行されるまでは、この LU を使用することはできません。

RUI_PURGE

RUI_PURGE verb は前の **RUI_READ** を取り消します。**lua_flag1** を LUA_NO WAIT（即時戻りオプション）にセットしないで **RUI_READ** を送った場合に、指定したフロー上に使用可能なデータがないと、その RUI_READ は無制限に待機状態になることがあります。 **RUI_PURGE** は、このような待機中の verb の制御を強制的に戻らせます（1 次戻りコードは CANCELLED）。

指定パラメーター: アプリケーションは次のパラメーターを提供します。

lua_verb
LUA_VERB_RUI

lua_verb_length
LUA verb レコードの長さ（バイト数）。 これは sizeof(struct LUA_COMMON) にセットします。

lua_opcode
LUA_OPCODE_RUI_PURGE

lua_correlator
任意選択。 この verb をアプリケーション内の他の処理に関連付けるために使用できる 4 バイトの値です。 LUA はこの情報を使用せず、変更もしません。

lua_luname
セッションで使用するローカル LU の ASCII 名。これは、活動 LUA セッションの LU 名に一致している必要があります。

このパラメーターが必要なのは、**lua_sid** パラメーターが 0 の場合だけです。 **lua_sid** にセッション ID を指定した場合は、LUA はこのパラメーターを使用しません。

このパラメーターは長さが 8 バイトでなければなりません。名前が 8 文字に満たない場合は、右側をスペース (0x20) で埋めてください。

lua_sid
セッションのセッション ID。 これは、前の **RUI_INIT** verb で戻されたセッション ID に一致していなければなりません。

このパラメーターは任意選択です。ここでセッション ID を指定しない場合は、**lua_luname** パラメーターにセッションの LU 名を指定する必要があります。

lua_data_ptr
除去する **RUI_READ** LUA_VERB_RECORD を指すポインター。

lua_post_handle
これは、非同期 verb の完了を通知するために使用する 4 バイトのハンドルです。

戻りパラメーター: 次のパラメーターは常に戻されます。

lua_flag2
これは、verb が非同期に完了した場合に限り LUA_ASYNC にセットされます。

その他の戻りパラメーターは、verb が正常に完了したかどうかによって異なります。次の項を参照してください。

verb が正常に完了した場合は、次のパラメーターが戻されます。

lua_prim_rc
LUA_OK

lua_sid
この verb を発行するときに、アプリケーションでセッション ID ではなく **lua_luname** パラメーターを指定してある場合は、LUA はセッション ID を提供します。

次の戻りコードは、他の verb によって取り消されたことが原因で、この verb が正常に完了しなかったことを示します。

lua_prim_rc
LUA_CANCELLED

lua_sec_rc
LUA_TERMINATED

この verb が保留状態にあるときに **RUI_TERM** verb が発行されました。

次の戻りコードは、指定パラメーターのどれかにエラーがあったために、この verb が正常に完了しなかったことを示します。

lua_prim_rc
LUA_PARAMETER_CHECK

lua_sec_rc
可能な値は次のとおりです。

LUA_BAD_DATA_PTR
lua_data_ptr パラメーターが 0 にセットされています。

LUA_RESERVED_FIELD_NOT_ZERO
verb レコード内の予約フィールド、またはこの verb では使用されないパラメーターが、0 以外の値にセットされています。

LUA_VERB_LENGTH_INVALID
lua_verb_length パラメーターの値が、この verb に必要な verb レコードの長さに達していません。

次の戻りコードは、セッション状態がこの verb にとって無効であるときに、このverb が発行されたことを示します。

lua_prim_rc
LUA_STATE_CHECK

lua_sec_rc
可能な値は次のとおりです。

LUA_SEC_RC_OK
前の **RUI_PURGE** verb がまだこのセッションで進行中です。

LUA_NO_RUI_SESSION
このセッションで、まだ **RUI_INIT** verb が正常に完了していないか、またはセッション障害が発生しました。

次の戻りコードは、指定した verb レコードは有効であったが、verb が正常に完了しなかったことを示します。

lua_prim_rc
LUA_UNSUCCESSFUL

lua_sec_rc
可能な値は次のとおりです。

LUA_INVALID_PROCESS
この verb を発行したアプリケーション・インスタンスは、このセッションで **RUI_INIT** verb を発行したインスタンスと同じではありません。

LUA_NO_READ_TO_PURGE
lua_data_ptr パラメーターに、**RUI_READ** LUA_VERB_RECORD を指すポインターが含まれていなかったか、または **RUI_PURGE** verb が発行される前に **RUI_READ** verb が完了しました。

次の 1 次戻りコードおよび 2 次戻りコードは、その他の理由で verb が正常に完了しなかったことを示します。

lua_prim_rc
LUA_SESSION_FAILURE

セッションがダウンしました。

lua_sec_rc
可能な値は次のとおりです。

LUA_LU_COMPONENT_DISCONNECTED
通信リンクまたはホスト LU に関する問題が原因で、LUA セッションが失敗しました。

LUA_RUI_LOGIC_ERROR
この戻りコードは次のいずれかを示します。

- ホスト・システムが SNA プロトコルに違反した。
- は LUA で内部エラーが検出されたことを示します。

トレースを活動状態にして問題を再現し、ホストが正しいデータを送っているかどうかを確認してください。

lua_prim_rc
LUA_INVALID_VERB

lua_verb パラメーターまたは **lua_opcode** パラメーターのいずれかが無効でした。 verb は実行されませんでした。

lua_prim_rc

LUA_UNEXPECTED_DOS_ERROR

資源不足などのオペレーティング・システム・エラーが起こりました。

lua_sec_rc

この値はオペレーティング・システムの戻りコードです。 この戻りコードの意味については、オペレーティング・システムの資料を参照してください。

他の verb との相互作用: この verb が使用できるのは、**RUI_READ** が発行されていて、その完了が保留状態にある（つまり 1 次戻りコードが IN_PROGRESS である）場合だけです。このセッションで他の **RUI_PURGE** が実行中である場合は、このverb は発行しないでください。

RUI_INIT_STATUS

アプリケーションは **RUI_INIT_STATUS** 指示を発行することはできません。 Communications Server は、**RUI_INIT** の処理中にこの指示をアプリケーションに送って、LU-SSCP セッションの状況に関する情報を提供します。 **RUI_INIT_STATUS** が送られるのは、アプリケーションが、**RUI_INIT** を発行するときに状況情報を要求した場合だけです（229ページの『RUI_INIT』を参照）。

指定パラメーター: **RUI_INIT_STATUS** では次のパラメーターがセットされます。

lua_verb
LUA_VERB_RUI

lua_verb_length
LUA verb レコードのバイトの長さ（Communications Serverにより sizeof(LUA_COMMON) にセットされます）。

lua_opcode
LUA_OPCODE_RUI_INIT_STATUS

lua_primary_rc
LU-SSCP セッションの状況に関する情報が入ります。 指定できる値は次のとおりです。

LUA_LINK_INACTIVE
ホストへのリンクがまだ活動状態にありません。

LUA_PU_INACTIVE
ACTPU がまだ受信されていないか、または **DACTPU** が受信されました。

LUA_PU_ACTIVE
SSCP から **ACTPU** が受信されました。

LUA_PU_REACTIVATED
PU が活動状態にあるときに **ACTPU(COLD)** が受信されました。

LUA_LU_INACTIVE
ACTLU が拒否されたか、または **DACTLU** が受信されました。

LUA_UNKNOWN
データ・リンク制御のリンク・エラーが原因で LU-SSCP セッションがまだ活動状態になっていません。

lua_correlator
Communications Serverは、このパラメーターのために**RUI_INIT** verb 上で指定された値を使用します。

lua_post_handle
これは、非同期 verb の完了を通知するために使用する４バイトのハンドルです。 Communications Serverは、このパラメーターのために**RUI_INIT** verb 上で指定された値を使用します。

RUI_READ

RUI_READ verb は、ホストからアプリケーションの LU に送られたデータまたは状況情報を受け取ります。データを読み取りたい特定のメッセージ・フロー（LU 通常、LU 急送、SSCP 通常、または SSCP 急送）を指定するか、または複数のメッセージ・フローを指定することができます。 ある **RUI_READ** verb が未完了の状態でも、同じフローを指定していなければ、複数の **RUI_READ** verb を同時に発行することができます。

指定パラメーター: アプリケーションは次のパラメーターを提供します。

lua_verb
LUA_VERB_RUI

lua_verb_length
LUA verb レコードの長さ（バイト数）。 これは sizeof(struct LUA_COMMON) にセットします。

lua_opcode
LUA_OPCODE_RUI_READ

lua_correlator
任意選択。 この verb をアプリケーション内の他の処理に関連付けるために使用できる 4 バイトの値です。 LUA はこの情報を使用せず、変更もしません。

lua_luname
セッションで使用するローカル LU の ASCII 名。これは、活動 LUA セッションの LU 名に一致している必要があります。

このパラメーターが必要なのは、**lua_sid** パラメーターが 0 の場合だけです。 **lua_sid** にセッション ID を指定した場合は、LUA はこのパラメーターを使用しません。

このパラメーターは長さが 8 バイトでなければなりません。名前が 8 文字に満たない場合は、右側をスペース (0x20) で埋めてください。

lua_sid
セッションのセッション ID。 これは、前の **RUI_INIT** verb で戻されたセッション ID に一致していなければなりません。

このパラメーターは任意選択です。ここでセッション ID を指定しない場合は、**lua_luname** パラメーターにセッションの LU 名を指定する必要があります。

lua_max_length
データを受信するために用意されているバッファーの長さ（**lua_data_ptr** の項を参照）。

lua_data_ptr
データを受信するために用意されているバッファーを指すポインター。

lua_post_handle

これは、非同期 verb の完了を通知するために使用する 4 バイトのハンドルです。

lua_flag1

フラグは、次の値のどれか 1 つ、またはいくつかを OR で結んだものです。

- 読取り可能なデータの有無に関係なく、すぐに **RUI_READ** verb を戻すようにしたい場合は、LUA_NOWAIT をセットし、この verb がデータを待ってから戻るようにしたい場合は、これをセットしないでください。
- 最新の **RUI_BID** verb を再び使用可能にしたい場合は、LUA_BID_ENABLE をセットし（これは前とまったく同じパラメーターを指定して **RUI_BID** を再発行するのと同じです）、**RUI_BID** を再び使用可能にしたくない場合は、これをセットしないでください。

  注: 前の **RUI_BID** を再び使用可能にした場合は、初めに割り振られていた LUA_VERB_RECORD が再利用され、LUA_VERB_RECORD は解放も変更もできません。
- どのメッセージ・フローからデータを読み取るかを指示するために、次の 1 つまたは複数のフラグをセットします。

  LUA_SSCP_EXP
  SSCP 急送フロー

  LUA_LU_EXP
  LU 急送フロー

  LUA_SSCP_NORM
  SSCP 通常フロー

  LUA_LU_NORM
  LU 通常フロー

  複数のフラグがセットされている場合は、使用可能な最高優先順位のデータが戻されます。 優先順位（高から低へ）は次のとおりです。

  1. SSCP 急送
  2. LU 急送
  3. SSCP 通常
  4. LU 通常

  どのフローからデータが読み取られたかを示すために、**lua_flag2** の中で同じフラグがセットされます（240ページの『戻りパラメーター』を参照）。

戻りパラメーター: 次のパラメーターは常に戻されます。

lua_flag2

verb が非同期に完了した場合は、LUA_ASYNC がセットされます（verb が同期に完了したときはセットされません）。

RUI_BID が正常に再び使用可能になった場合は、LUA_BID_ENABLE がセットされます（再び使用可能にならなかった場合はセットされません）。

その他の戻りパラメーターは、verb が正常に完了したかどうかによって異なります。次の項を参照してください。

verb が正常に実行された場合は、LUA は次のパラメーターも戻します。

lua_prim_rc
LUA_OK

次のパラメーターは、verb が正常に完了した場合に戻されます。
また、**lua_data_length** パラメーターが小さすぎたために切捨てが起きたデータとともに verb が戻った場合も同様です。

lua_sid
この verb を発行するときに、アプリケーションでセッション ID ではなく **lua_luname** パラメーターを指定してある場合は、LUA はセッション ID を提供します。

lua_data_length
受信したデータの長さ。 LUA は、**lua_data_ptr** に指定したバッファーにデータを入れます。

lua_th 受信したメッセージの伝送ヘッダー (TH) からの情報。

lua_rh 受信したメッセージの要求/応答ヘッダー (RH) からの情報。

lua_message_type
受信したメッセージのメッセージ・タイプ。これは次の値のいずれかです。

LUA_MESSAGE_TYPE_LU_DATA
LUA_MESSAGE_TYPE_SSCP_DATA
LUA_MESSAGE_TYPE_RSP
LUA_MESSAGE_TYPE_BID
LUA_MESSAGE_TYPE_BIND
LUA_MESSAGE_TYPE_BIS
LUA_MESSAGE_TYPE_CANCEL
LUA_MESSAGE_TYPE_CHASE
LUA_MESSAGE_TYPE_CLEAR
LUA_MESSAGE_TYPE_CRV
LUA_MESSAGE_TYPE_LUSTAT_LU
LUA_MESSAGE_TYPE_LUSTAT_SSCP
LUA_MESSAGE_TYPE_QC
LUA_MESSAGE_TYPE_QEC
LUA_MESSAGE_TYPE_RELQ
LUA_MESSAGE_TYPE_RTR
LUA_MESSAGE_TYPE_SBI
LUA_MESSAGE_TYPE_SHUTD
LUA_MESSAGE_TYPE_SIGNAL

LUA_MESSAGE_TYPE_SDT

LUA_MESSAGE_TYPE_STSN

LUA_MESSAGE_TYPE_UNBIND

lua_flag2 parameters

これはどのメッセージ・フローからデータが読み取られたかを示すもので、次のいずれかの値にセットされます。

LUA_SSCP_EXP

SSCP 急送フロー

LUA_LU_EXP

LU 急送フロー

LUA_SSCP_NORM

SSCP 通常フロー

LUA_LU_NORM

LU 通常フロー

次の戻りコードは、他の verb または内部エラーが原因でこの verb が取り消されたため、この verb が正常に完了しなかったことを示します。

lua_prim_rc

LUA_CANCELLED

lua_sec_rc

可能な値は次のとおりです。

LUA_PURGED

RUI_PURGE verb により **RUI_READ** verb が取り消されました。

LUA_TERMINATED

この verb が保留状態にあるときに **RUI_TERM** verb が発行されました。

次の戻りコードは、指定パラメーターのどれかにエラーがあったために、この verb が正常に完了しなかったことを示します。

lua_prim_rc

LUA_PARAMETER_CHECK

lua_sec_rc

可能な値は次のとおりです。

LUA_BAD_DATA_PTR

lua_data_ptr パラメーターに不適切な値が含まれていました。

LUA_BID_ALREADY_ENABLED

RUI_BID verb を再度使用可能にするために **lua_flag1** が LUA_BID_ENABLE にセットされましたが、前の **RUI_BID** verb がまだ進行中です。

LUA_DUPLICATE_READ_FLOW

lua_flag1 のフロー・フラグで、**RUI_READ** verb がすでに未完了状態になっている１つまたは複数のセッション・フローを指定しています。１つのセッション・フロー上で待機できる **RUI_READ** は一度に１つだけです。

LUA_INVALID_FLOW

lua_flag1 フロー・フラグが何もセットされていません。 どのフローから読み取るかを指示するために、これらのフラグの少なくとも１つはセットする必要があります。

LUA_NO_PREVIOUS_BID_ENABLED

RUI_BID verb を再び使用可能にするために **lua_flag1** が LUA_BID_ENABLE にセットされましたが、前の **RUI_BID** verb には使用可能にできるものがありません （詳細は、245ページの『他の verb との相互作用』を参照してください）。

LUA_RESERVED_FIELD_NOT_ZERO

verb レコード内の予約フィールド、またはこの verb では使用されないパラメーターが、0 以外の値にセットされています。

LUA_VERB_LENGTH_INVALID

lua_verb_length パラメーターの値が、この verb に必要な verb レコードの長さに達していません。

次の戻りコードは、セッション状態がこの verb にとって無効であるときに、このverb が発行されたことを示します。

lua_prim_rc

LUA_STATE_CHECK

lua_sec_rc

LUA_NO_RUI_SESSION

RUI_INIT verb がこのセッションではまだ正常に完了していないか、またはセッションで障害が発生しました。

次の１次戻りコードは、次の２つのケースのどちらかを示します。どちらが該当するかは２次戻りコードに示されます。

- Communications Server がホストから受信したデータにエラーを検出しました。 Communications Server は、受信したメッセージを **RUI_READ** verb 上のアプリケーションに渡す代わりに、そのメッセージ（そのメッセージが連鎖の一部である場合は連鎖の残りの部分）を破棄し、ホストに否定応答を送ります。 LUA は、後続の **RUI_READ** verb または **RUI_BID** verb 上のアプリケーションに、否定応答が送られたことを知らせます。
- LUA アプリケーションが、すでに、連鎖の途中にあるメッセージに対して否定応答を送りました。 Communications Server は、この連鎖内の後続のメッセージを除去し、連鎖からすべてのメッセージを受信し、除去したことをアプリケーションに報告しています。

lua_prim_rc

LUA_NEGATIVE_RSP

lua_sec_rc

0 以外の 2 次戻りコードには、否定応答とともにホストに送られたセンス・コードが入っています。 これは、Communications Server がホスト・データにエラーを検出し、ホストに否定応答を送ったことを示します。 戻されるセンス・コード値の解釈の仕方は、175ページの『SNA の層』を参照してください。

0 の 2 次戻りコードは、連鎖の途中のメッセージに対する否定応答の **RUI_WRITE** の後で、Communications Server がその連鎖からすべてのメッセージを受け取り、そして破棄したことを示します。

次の戻りコードは、指定した verb レコードは有効であったが、verb が正常に完了しなかったことを示します。

lua_prim_rc

LUA_UNSUCCESSFUL

lua_sec_rc

可能な値は次のとおりです。

LUA_DATA_TRUNCATED

lua_data_length パラメーターが、受信したメッセージ・データの実際の長さに達していません。
verb に戻されたのはデータの **lua_data_length** バイトだけです。残りのデータは破棄されています。 この 2 次戻りコードの場合は、さらに追加のパラメーターが戻されます。

LUA_NO_DATA

データを待たず即時に戻すことを指示するために、**lua_flag1** が LUA_NOWAIT にセットされていますが、指定したセッション・フロー上に現在使用可能なデータがありませんでした。

LUA_INVALID_PROCESS

この verb を発行したアプリケーション・インスタンスは、このセッションで **RUI_INIT** verb を発行したインスタンスと同じではありません。

次の 1 次戻りコードおよび 2 次戻りコードは、その他の理由で verb が正常に完了しなかったことを示します。

lua_prim_rc

LUA_SESSION_FAILURE

セッションがダウンしました。

lua_sec_rc

可能な値は次のとおりです。

LUA_LU_COMPONENT_DISCONNECTED
通信リンクまたはホスト LU に関する問題が原因で、LUA セッションが失敗しました。

LUA_RUI_LOGIC_ERROR
この戻りコードは次のいずれかを示します。

- ホスト・システムが SNA プロトコルに違反した。
- は LUA で内部エラーが検出されたことを示します。

トレースを活動状態にして問題を再現し、ホストが正しいデータを送っているかどうかを検査してください。

lua_prim_rc
LUA_INVALID_VERB

lua_verb パラメーターまたは **lua_opcode** パラメーターのいずれかが無効でした。 verb は実行されませんでした。

lua_prim_rc
LUA_UNEXPECTED_DOS_ERROR

資源不足などのオペレーティング・システム・エラーが起こりました。

lua_sec_rc
この値はオペレーティング・システムの戻りコードです。 この戻りコードの意味については、オペレーティング・システムの資料を参照してください。

他の verb との相互作用: この verb を発行する前に、**RUI_INIT** verb が正常に完了していなければなりません。 既存の **RUI_READ** が保留状態にあるときに別の **RUI_READ** を発行できるのは、保留状態の **RUI_READ** とは異なるセッション・フローを指定した場合に限られます。つまり、同じセッション・フローについて複数の **RUI_READ** を未完了状態にすることはできません。

lua_flag1 を LUA_BID_ENABLE にセットできるのは、次のすべての条件が満たされている場合に限られます。

- **RUI_BID** がすでに正常に発行され、そして完了している。
- **RUI_BID** verb 用に割り振られている記憶域が、解放も変更もされていない。
- 他の **RUI_BID** が保留状態にない。

使用上の注意: 受信したデータが **lua_max_length** パラメーターの値より長い場合は、そのデータは切り捨てられます。データの **lua_max_length** バイトだけが戻されます。 1 次および 2 次戻りコードとして、LUA_UNSUCCESSFUL および LUA_DATA_TRUNCATED も戻されます。

RUI_READ verb を使用してメッセージが読み取られると、そのメッセージは着信メッセージ待ち行列から除去されるので、再びアクセスすることはできなくなります。

注: RUI_BID verb は、データを待ち行列から除去せずに読取るために使用できます。つまり、このアプリケーションはこの verb を使用して使用可能なデータのタイプをチェックできますが、データはそのまま着信待ち行列上に残っているので、ただちに処理する必要はありません。

1次 から 2 次へのハーフセッション（これはホスト構成で指定します）では、ペーシングを使用して、Communications Server のノードがメッセージであふれるのを防ぐことができます。 LUA アプリケーションのメッセージ読み取りの速度が低下すると、Communications Server は、ホストへのペーシング応答の速度を低下させるために、ペーシング応答の送信を遅延させます。

RUI_TERM

RUI_TERM verb は、特定の LUA LU について、LU-LU セッションと LU-SSCP セッションの両方を終了します。

指定パラメーター: アプリケーションは次のパラメーターを提供します。

lua_verb
LUA_VERB_RUI

lua_verb_length
LUA verb レコードの長さ（バイト数）。 これは sizeof(structLUA_COMMON) にセットします。

lua_opcode
LUA_OPCODE_RUI_TERM

lua_correlator
任意選択。 この verb をアプリケーション内の他の処理に関連付けるために使用できる 4 バイトの値です。 LUA はこの情報を使用せず、変更もしません。

lua_luname
セッションで使用するローカル LU の ASCII 名。これは、活動 LUA セッションの LU 名（または未完了の **RUI_INIT** verb に指定されている LU 名）に一致していなければなりません。

このパラメーターが必要なのは、**lua_sid** パラメーターが 0 の場合だけです。 **lua_sid** にセッション ID を指定した場合は、LUA はこのパラメーターを使用しません。

このパラメーターは長さが 8 バイトでなければなりません。名前が 8 文字に満たない場合は、右側をスペース (0x20) で埋めてください。

lua_sid
セッションのセッション ID。 これは、前の **RUI_INIT** verb で戻されたセッション ID に一致していなければなりません。

このパラメーターは任意選択です。ここでセッション ID を指定しない場合は、**lua_luname** パラメーターにセッションの LU 名を指定する必要があります。

lua_post_handle
これは、非同期 verb の完了を通知するために使用する 4 バイトのハンドルです。

戻りパラメーター: 次のパラメーターは常に戻されます。

lua_flag2
これは、verb が非同期に完了した場合に限り LUA_ASYNC にセットされます。

その他の戻りパラメーターは、verb が正常に完了したかどうかによって異なります。次の項を参照してください。

verb が正常に実行された場合は、LUA は次のパラメーターも戻します。

lua_prim_rc
LUA_OK

次の戻りコードは、指定パラメーターのどれかにエラーがあったために、この verb が正常に完了しなかったことを示します。

lua_prim_rc
LUA_PARAMETER_CHECK

lua_sec_rc
可能な値は次のとおりです。

LUA_RESERVED_FIELD_NOT_ZERO
verb レコード内の予約フィールド、またはこの verb では使用されないパラメーターが、0 以外の値にセットされています。

LUA_VERB_LENGTH_INVALID
lua_verb_length パラメーターの値が、この verb に必要な verb レコードの長さに達していません。

次の戻りコードは、セッション状態がこの verb にとって無効であるときに、このverb が発行されたことを示します。

lua_prim_rc
LUA_STATE_CHECK

lua_sec_rc
LUA_NO_RUI_SESSION

RUI_INIT verb がこのセッションではまだ正常に完了していないか、またはセッションで障害が発生しました。

次の戻りコードは、指定した verb レコードは有効であったが、verb が正常に完了しなかったことを示します。

lua_prim_rc
LUA_UNSUCCESSFUL

lua_sec_rc
可能な値は次のとおりです。

LUA_COMMAND_COUNT_ERROR
この verb を発行したときに、**RUI_TERM** がすでに保留状態になっていました。

LUA_INVALID_PROCESS
この verb を発行したアプリケーション・インスタンスは、このセッションで **RUI_INIT** verb を発行したインスタンスと同じではありません。

次の 1 次戻りコードおよび 2 次戻りコードは、その他の理由で verb が正常に完了しなかったことを示します。

lua_prim_rc

LUA_SESSION_FAILURE

セッションがダウンしました。

lua_sec_rc

可能な値は次のとおりです。

LUA_LU_COMPONENT_DISCONNECTED

通信リンクまたはホスト LU に関する問題が原因で、LUA セッションが失敗しました。

LUA_RUI_LOGIC_ERROR

この戻りコードは次のいずれかを示します。

- ホスト・システムが SNA プロトコルに違反した。
- は LUA で内部エラーが検出されたことを示します。

トレースを活動状態にして問題を再現し、ホストが正しいデータを送っているかどうかを検査してください。

lua_prim_rc

LUA_INVALID_VERB

lua_verb パラメーターまたは **lua_opcode** パラメーターのいずれかが無効でした。 verb は実行されませんでした。

lua_prim_rc

LUA_UNEXPECTED_DOS_ERROR

資源不足などのオペレーティング・システム・エラーが起こりました。

lua_sec_rc

この値はオペレーティング・システムの戻りコードです。 この戻りコードの意味については、オペレーティング・システムの資料を参照してください。

他の verb との相互作用: この verb は、**RUI_INIT** verb を発行した後であれば、それが完了しているかどうかに関係なく、いつでも発行できます。 **RUI_TERM** を発行したときに他の LUA verb が保留状態にある場合は、その保留 verb についてはそれ以上の処理は行われず、その verb は 1 次戻りコード LUA_CANCELLED を伴って戻ります。

この verb の完了後は、このセッションでは他の LUA verb は発行できなくなります。

RUI_WRITE

RUI_WRITE verb は、SNA 要求単位または応答単位を、LU-LU セッションまたはLU-SSCP セッションを介して、 LUA アプリケーションからホストに送ります。

指定パラメーター: アプリケーションは次のパラメーターを提供します。

lua_verb
LUA_VERB_RUI

lua_verb_length
LUA verb レコードの長さ（バイト数）。 これは sizeof(struct LUA_COMMON) にセットします。

lua_opcode
LUA_OPCODE_RUI_WRITE

lua_correlator
任意選択。 この verb をアプリケーション内の他の処理に関連付けるために使用できる 4 バイトの値です。 LUA はこの情報を使用せず、変更もしません。

lua_luname
セッションで使用するローカル LU の ASCII 名。これは、活動 LUA セッションの LU 名に一致している必要があります。

このパラメーターが必要なのは、**lua_sid** パラメーターが 0 の場合だけです。**lua_sid** にセッション ID を指定した場合は、LUA はこのパラメーターを使用しません。

このパラメーターは長さが 8 バイトでなければなりません。名前が 8 文字に満たない場合は、右側をスペース (0x20) で埋めてください。

lua_sid
セッションのセッション ID。 これは、前の **RUI_INIT** verb で戻されたセッション ID に一致していなければなりません。

このパラメーターは任意選択です。ここでセッション ID を指定しない場合は、**lua_luname** パラメーターにセッションの LU 名を指定する必要があります。

lua_data_length
送信されるデータの長さ（**lua_data_ptr** の項を参照）。 LU 通常フローでデータを送信する場合は、最大長は、ホストから受信した **BIND** に指定されている長さです。その他のフローの場合は、最大長は 256 バイトです。

肯定応答を送信するときは、このパラメーターは通常 0 にセットされます。LUA は、提供される順序番号に基づいて応答を完了します（**lua_th.snf** を参照）。 **BIND** または **STSN** に対する肯定応答の場合は、拡張応答が許されるので、0 以外の値を使用できます。

否定応答を送信するときは、このパラメーターを、データ・バッファーで提供される SNA センス・コードの長さ（4バイト）にセットします（**lua_data_ptr** の項を参照）。

lua_data_ptr
提供されるデータが入っているバッファーを指すポインター。

要求の場合、またはデータを必要とする肯定応答の場合は、バッファーには RU 全体が入っていることが必要です。 RU の長さは **data_length** に指定されていなければなりません。

否定応答の場合は、バッファーには SNA センス・コードが入っています。

lua_post_handle
これは、非同期 verb の完了を通知するために使用する4バイトのハンドルです。

lua_th.snf
応答の送信の場合に限り必要です。 これは、この応答の対象となっている要求の順序番号です。

lua_rh 要求を送信するときは、ほとんどの **lua_rh** フラグは、送信するメッセージの RH（要求ヘッダー）に対応するようにセットする必要があります。 LUA_PI および LUA_QRI はセットしないでください。これらは LUA がセットするものです。

応答を送信するときは、次の2つの **lua_rh** フラグだけがセットされます。

LUA_RRI
応答を示すためにセットされます。

LUA_RI
肯定応答の場合はセットされず、否定応答の場合はセットされます。

lua_flag1
どのメッセージ・フローでデータを送信するのかを示すために、次のいずれかのフラグをセットします。

LUA_LU_EXP
LU 急送フロー

LUA_SSCP_NORM
SSCP 通常フロー

LUA_LU_NORM
LU 通常フロー

これらのフラグのどれか1つだけをセットする必要があります。

注: Communications Server では、アプリケーションが SSCP 急送フロー上でデータを送ること (LUA_SSCP_EXP) はできません。

戻りパラメーター: 次のパラメーターは常に戻されます。

lua_flag2
これは、verb が非同期に完了した場合に限り LUA_ASYNC にセットされます。

その他の戻りパラメーターは、verb が正常に完了したかどうかによって異なります。次の項を参照してください。

verb が正常に実行された場合は、LUA は次のパラメーターも戻します。

lua_prim_rc
LUA_OK

lua_sid
この verb を発行するときに、アプリケーションでセッション ID ではなく **lua_luname** パラメーターを指定してある場合は、LUA はセッション ID を提供します。

lua_th 送信完了済みのメッセージの TH。これは、LUA により記入されたフィールドも含みます。 ホストからの応答との対応付けのために、**lua_th.snf**（順序番号）の値の保管が必要になることがあります。

lua_rh 送信完了済みのメッセージの RH。これは、LUA により記入されたフィールドも含みます。

lua_flag2
これは、どのメッセージ・フローでデータが受信されたかを示すために、次のいずれかの値にセットされます。

LUA_SSCP_EXP
SSCP 急送フロー

LUA_LU_EXP
LU 急送フロー

LUA_SSCP_NORM
SSCP 通常フロー

LUA_LU_NORM
LU 通常フロー

次の戻りコードは、他の verb によって取り消されたことが原因で、この verb が正常に完了しなかったことを示します。

lua_prim_rc
LUA_CANCELLED

lua_sec_rc
LUA_TERMINATED

RUI_TERM verb がこのセッションのために発行されたので verb は取り消されました。

次の戻りコードは、指定パラメーターのどれかにエラーがあったために、この verb が正常に完了しなかったことを示します。

lua_prim_rc

LUA_PARAMETER_CHECK

lua_sec_rc

可能な値は次のとおりです。

LUA_BAD_DATA_PTR

lua_data_ptr パラメーターに不適切な値が含まれていました。

LUA_DUPLICATE_WRITE_FLOW

この verb で指定したセッション・フローについて、**RUI_WRITE** はすでに未完了状態です（セッション・フローは、**lua_flag1** フロー・フラグのどれかをセットすることにより指定します)。 各セッション・フローについて未完了状態にできる **RUI_WRITE** は、一度に 1 つだけです。

LUA_INVALID_FLOW

lua_flag1 が LUA_SSCP_EXP にセットされています。これは、メッセージを SSCP 急送フローで送信することを示します。
Communications Server では、アプリケーションがこのフロー上でデータを送ること (LUA_SSCP_EXP) はできません。

LUA_MULTIPLE_WRITE_FLOWS

lua_flag1 フロー・フラグが 2 つ以上セットされています。 これらのフラグは、どのセッション・フローによりデータを送信するかを指示するためのもので、どれか 1 つだけをセットする必要があります。

LUA_REQUIRED_FIELD_MISSING

この戻りコードは次のいずれかの場合を示します。

- **lua_flag1** フロー・フラグが何もセットされていません。 これらのフラグはどれか 1 つだけセットする必要があります。
- 応答を送信するために **RUI_WRITE** verb が使用されましたが、応答には提供されているものより多くのデータが必要です。

LUA_RESERVED_FIELD_NOT_ZERO

verb レコード内の予約フィールド、またはこの verb では使用されないパラメーターが、0 以外の値にセットされています。

LUA_VERB_LENGTH_INVALID

lua_verb_length パラメーターの値が、この verb に必要な verb レコードの長さに達していません。

次の戻りコードは、セッション状態がこの verb にとって無効であるときに、このverb が発行されたことを示します。

lua_prim_rc

LUA_STATE_CHECK

lua_sec_rc

可能な値は次のとおりです。

LUA_MODE_INCONSISTENCY
RUI_WRITE で送られた SNA メッセージが現時点では無効でした。これは、LU-LU セッションにおいて、そのセッションがバインドされる前にデータを送信しようとしたことが原因です。送信された SNA メッセージの順序を検査してください。

LUA_NO_RUI_SESSION
このセッションで、まだ **RUI_INIT** verb が正常に完了していないか、またはセッション障害が発生しました。

次の戻りコードは、指定した verb レコードは有効であったが、verb が正常に完了しなかったことを示します。

lua_prim_rc
LUA_UNSUCCESSFUL

lua_sec_rc
可能な値は次のとおりです。

LUA_FUNCTION_NOT_SUPPORTED
この戻りコードは次のいずれかの場合を示します。

- **lua_rh** が LUA_FI（フォーマット標識）にセットされていますが、提供された RU の最初のバイトが、認識される要求コードではありませんでした。
- **lua_rh** が LUA_RH_NC にセットされています（RU カテゴリーがネットワーク制御 (NC) カテゴリーを指定しています）。Communications Server では、アプリケーションはこのカテゴリーの要求を送信することはできません。

LUA_INVALID_PROCESS
この verb を発行したアプリケーション・インスタンスは、このセッションで **RUI_INIT** verb を発行したインスタンスと同じではありません。

LUA_INVALID_SESSION_PARAMETERS
アプリケーションは、**RUI_WRITE** を使用して、ホストから受信した **BIND** メッセージに対する肯定応答を送信しました。しかし、Communications Server ・ノードは指定された**BIND** パラメーターを受け入れることができず、否定応答をホストへ送りました。Communications Serverが受け入れる **BIND** プロファイルの詳細は、175ページの『SNA の層』を参照してください。

LUA_RSP_CORRELATION_ERROR
RUI_WRITE を使用して応答を送信するときに、**lua_th.snf** パラメーター（これは応答の対象となっている受信メッセージの順序番号を示します）に有効な値が含まれていませんでした。

LUA_RU_LENGTH_ERROR
lua_data_length パラメーターに正しくない値が含まれていました。

LU 通常フローでデータを送信する場合は、最大長は、ホストから受信した **BIND** に指定されている長さです。その他のフローの場合は、最大長は 256 バイトです。

（その他の値）
その他の 2 次戻りコードは、提供された SNA データが無効かまたは送信できなかったことを示す SNA センス・コードです。 戻される SNA センス・コードの解釈の仕方については、175ページの『SNA の層』を参照してください。

次の 1 次戻りコードおよび 2 次戻りコードは、その他の理由で verb が正常に完了しなかったことを示します。

lua_prim_rc
LUA_SESSION_FAILURE

セッションがダウンしました。

lua_sec_rc
可能な値は次のとおりです。

LUA_LU_COMPONENT_DISCONNECTED
通信リンクまたはホスト LU に関する問題が原因で、LUA セッションが失敗しました。

LUA_RUI_LOGIC_ERROR
この戻りコードは次のいずれかを示します。ホスト・システムが SNA プロトコルに違反した。 は LUA で内部エラーが検出されたことを示します。

トレースを活動状態にして問題を再現し、ホストが正しいデータを送っているかどうかを確認してください。

lua_prim_rc
LUA_INVALID_VERB

lua_verb パラメーターまたは **lua_opcode** パラメーター
のいずれかが無効でした。
verb は実行されませんでした。

lua_prim_rc
LUA_UNEXPECTED_DOS_ERROR

資源不足などのオペレーティング・システム・エラーが起こりました。

lua_sec_rc
この値はオペレーティング・システムの戻りコードです。 この戻りコードの意味については、オペレーティング・システムの資料を参照してください。

他の verb との相互作用: この verb を発行する前に、**RUI_INIT** verb が正常に発行されていることが必要です。 既存の **RUI_WRITE** が保留状態にあるときに別の **RUI_WRITE** を発行できるのは、保留状態の **RUI_WRITE** とは異なるセッション・

フローを指定した場合に限られます。つまり、同じセッション・フローについて複数の **RUI_WRITE** を未完了状態にすることはできません。

SSCP 通常フローでは、**RUI_INIT** verb が正常に完了した後はいつでも **RUI_WRITE** verb を発行できます。LU 急送フローまたは LU 通常フローで **RUI_WRITE** verb を使用できるのは、**BIND** を受信した後に限られ、また、**BIND** に指定されているプロトコルを守る必要があります。

使用上の注意: **RUI_WRITE** が正常に完了した場合、メッセージが、データ・リンクへの待ち行列に正常に入れられたことを示します。これは、必ずしも、メッセージが正常に送信されたこと、またはホストがそれを受け入れたことを示すものではありません。 2 次から 1 次へのハーフセッション（これは **BIND** で指定します）では、ペーシングを使用して、LUA アプリケーションがローカル LU またはリモート LU の処理容量を超えるデータを送信するのを防ぐことができます。 その場合は、LUA は LU 通常フローでの **RUI_WRITE** を遅延させることがあり、その完了までに少々時間がかかることがあります。

注: Communications Server では、アプリケーションが SSCP 急送フロー上でデータを送ること (LUA_SSCP_EXP) はできません。

第3部 共通サービス API

第14章 エントリー・ポイント

Communications Server は、共通サービス・プログラミング・インターフェースを提供します。 この API は、Communications Server API を使用するアプリケーション・プログラムで使用できる共通サービス verb (CSV) から成っています。

Communications Server のアプリケーション・プログラムはこれらの共通サービスverbを使用して、次の 1 つまたは複数の処理を実行することができます。

- １バイト言語用のコード・ページ変換テーブルを保持する (**GET_CP_CONVERT_TABLE**)。
- ASCII ストリングを EBCDIC に、または EBCDIC を ASCII に変換する (**CONVERT**)。
- ２バイト文字ストリングを、あるコード・ページから別のコード・ページに変換する (**TRNSDT**)。

共通サービス・プログラムの作成

下記の表は、共通サービス・プログラムをコンパイルし、リンクするのに必要な、提供されているヘッダー・ファイルとライブラリーのソース・モジュールを示しています。

表 17. オペレーティング・システムのためのヘッダー・ファイルとライブラリー

オペレーティング・システム	ヘッダー・ファイル	ライブラリー	DLL 名
WINNT & WIN95	WINCSV.H	WINCSV32.LIB	WINCSV32.DLL
WIN3.1	WINCSV.H	WINCSV.LIB	WINCSV.DLL
OS/2	ACSSVC.H	ACSSVC.LIB	ACSSVC.DLL

以下に、共通サービス用のエントリー・ポイントについて説明します。

ACSSVC

これは、OS/2 の下でサポートされる唯一のエントリー・ポイントです。

これは、すべての CSV verb 用の同期エントリー・ポイントです。 Communications Serverは、既存のアプリケーションとの互換性を確保するために、このエントリー・ポイントを提供しています。

構文

```
void ACSSVC (long)
```

入力パラメーターは verb 制御ブロックのポインターです。

戻り値

戻り値については、1 次戻りコードおよび 2 次戻りコードを調べてください。

WinCSV()

この関数は、CSV API 用の同期エントリー・ポイントを提供します。

構文

```
void WINAPI WinCSV(long vcb)
```

パラメーター

	説明
vcb	verb 制御ブロックへのポインター。

戻り値

戻り値はありません。 verb 制御ブロック中の **primary_rc** および **secondary_rc** フィールドがエラーを示します。

注: 263ページの『WinAsyncCSV()』 も参照してください。

WinCSVCleanup()

この関数は、アプリケーションを終了し、CSV API からアプリケーションの登録を取り消します。

構文

```
BOOL WINAPI WinCSVCleanup(void);
```

戻り値

戻り値は、登録の取消しが成功したかどうかを示します。　値が0　以外であれば、Communications Serverはアプリケーションの登録を正常に取り消しています。 値が0の場合は、Communications Serverはアプリケーションの登録を取り消していません。

使用上の注意

WinCSVCleanup() は、CSV API アプリケーションの登録を CSV API から取り消すために使用します。たとえば、特定のアプリケーションに割り振られている資源を解放するためなどに使用します。

WinAsyncCSV()

このエントリー・ポイントは、サーバーでサポートされるアプリケーションのみを対象としています。

この関数は、**TRANSFER_MS_DATA** 専用の非同期エントリー・ポイントを提供します。 アプリケーションが他の verb にこの関数を使用しても、同期をとって動作します。

構文

```
HANDLE WlNAPI WinAsyncCSV(HHWND hWnd,
                          long vcb);
```

パラメーター

	説明
hwnd	完了メッセージを受け取るウィンドウ・ハンドル。
vcb	verb 制御ブロックへのポインター。

戻り値

戻り値は、非同期完了要求が成功したかどうかを示します。 この関数が成功した場合は、実際の戻り値は非同期タスク・ハンドルです。関数が成功しなかった場合は、Communications Server は `0` を戻します。

使用上の注意

非同期操作が完了すると、アプリケーションのウィンドウ *hWnd* は、**WinAsyncCSV** を入力ストリングとする **RegisterWindowMessage** から戻されたメッセージを受け取ります。 *wParam* 引数には、元の関数呼出しから戻された非同期のタスク・ハンドルが入っています。*lParam* 引数には元の VCB ポインターが入っていて、これを参照して最終戻りコードを判別できます。

この関数が正常に戻った場合は、Communications Serverは、操作が完了したとき、または会話が取り消されたときに、**WinAsyncCSV()** メッセージをアプリケーションに渡します。

WinCSVStartup()

この関数を使用すると、アプリケーションは要求された共通サービス verb API のバージョンを指定し、特定の CSV API の詳細情報を検索することができます。 この呼出しは必須ではありませんが、これを使用する場合は、**WinCSVCleanup** 呼出しも使用する必要があります。

構文

```
int WINAPI WinCSVStartup (WORD wVersionRequired,
                          LPWCSVDATA csvdata);
```

パラメーター

説明

wVersionRequired
要求された CSV API サポートのバージョンを指定します。 高位バイトはリリース番号（改訂番号）を示し、低位バイトはバージョン番号を示します。

lpwCSVDATA
基盤の CSV API DLL に関する情報が入ります。

戻り値

戻り値は、CSV API が正常にアプリケーションを登録したかどうか、および指定したバージョン番号をサポートするかどうかを示します。 戻された値が 0 である場合は、CSV API は指定したバージョンをサポートしており、アプリケーションを正常に登録しています。 その他の場合は、次のいずれかの値が戻されます。

WCSVVERNOTSUPPORTED
この CSV API は、要求されているバージョンの CSV API サポートを提供していません。

WCSVINVALID
CSV API は要求されているバージョンを判別できませんでした。

使用上の注意

WinCSVStartup() は、API の将来のリリースとの互換性を維持することを目的としています。 サポートされている現行バージョンは J1.0 です。

次の構造は、実際の CSV API 機能の詳細を記述しています。

```
typedef struct tagWCSVDATA { WORD wVersion;
                char szDescription[WCSVDESCRIPTION_LEN+1];
                } WCSVDATA, FAR *LPWCSVDATA;
```

アプリケーションは、最後の CSV API 呼出しの後で、**WinCSVCleanup()** を呼び出します。

GetCsvReturnCode()

このエントリー・ポイントは、VCB 中の 1 次および 2 次戻りコードを印刷可能ストリングに変換するために使用します。 このエントリー・ポイントは、アプリケーション・プログラムが使用する標準エラー・ストリングを戻します。

構文

```
int WINAPI GetCsvReturnCode (struct csv_hdr *vcb,
                             UINT buffer_length,
                             unsigned char *buffer_addr);
```

パラメーター

説明

vcb
verb 制御ブロックのアドレス。

buffer_length
buffer_addr が指し示すバッファーの長さ。 この長さの推奨値は 256 です。

buffer_addr
NULL 文字で終了する定型ストリングが入るバッファーのアドレス。

戻り値

0x20000001
パラメーターが無効です。この関数は、指定した **vcb** からの読取り、または指定したバッファーへの書込みができませんでした。

0X20000002
指定したバッファーが小さすぎます。

使用上の注意

buffer_addr に戻されるエラー・ストリングは、改行文字 (**\n**) で終わりません。

TrnsDt

このエントリー・ポイントは、サーバー上のアプリケーションのみにサポートされます。

これは、サーバーにのっているアプリケーションに対してだけサポートされます。

この関数は、SBCS ストリングおよび DBCS ストリングを、 1 つのコード・ページから別のコード・ページに変換します。 Communications Serverは、**TRNSDT.DLL** ファイルで **TrnsDt** を提供しています。 **TransDt** は DBCS セッションでのみ使用できます。

構文

```
TrnsDt (PASSSTRUCT *passparm);
```

PassParm フォーマット

この関数は、SBCS ストリングおよび DBCS ストリングを、 1 つのコード・ページから別のコード・ページに変換します。 次の表で、『✔』 は Communications Server がコード・ページのペアの間での変換をサポートしていることを示し、**-** (ハイフン) は Communications Server がその変換をサポートしていないことを示します。

日本		932	930	931	939	290	037	1027
	932	-	✔	✔	✔	✔	✔	✔
	930	✔	-	-	-	-	-	-
	931	✔	-	-	-	-	-	-
	939	✔	-	-	-	-	-	-
	290	✔	-	-	-	-	-	-
	037	✔	-	-	-	-	-	-
	1027	✔	-	-	-	-	-	-
韓国		949	833	834	933			
	949	-	✔	✔	✔			
	833	✔	-	-	-			
	834	✔	-	-	-			
	933	✔	-	-	-			
台湾		950	037	835	937			
	950	-	✔	✔	✔			
	037	✔	-	-	-			
	835	✔	-	-	-			
	937	✔	-	-	-			
中国		1381	836	837	935			
	1381	-	✔	✔	✔			
	836	✔	-	-	-			
	837	✔	-	-	-			
	935	✔	-	-	-			

コンパイルにはヘッダー・ファイル **TRNSDT.H** を使用し、リンクにはCommunications Server LIB サブディレクトリーからの **TRNSDT.LIB** ファイルを使用します。

PassParm フォーマット

passparm 形式は次の通りです。

WORD *parm_length*
この構造の長さ（入力）

WORD *exit_code*
終了コード（出力）

0000H 正常終了。

0001H サポートしていない変換が指定されました。

000CH
Exit_code フィールドが 0 に初期設定されていません。

0080H 最後の文字が 2 バイト文字の最初のバイトです。 2 バイト文字の最初のバイトの 代わりに NULL 文字が挿入されます。

WORD *in_length*
ソース・バッファーの長さ（入力）

LPBYTE *in_addr*
ソース・バッファー・アドレス（入力）

WORD *out_length*
宛先バッファーの長さ（入力）

指定した長さが小さすぎて、変換したすべてのデータを戻せない場合は、必要な長さが戻されます。

LPBYTE *out_addr*
宛先アドレス・バッファー（入力）

WORD *trns_id*
ゼロに予約済み（入力）

WORD *in_page*
変換元コード・ページ（入力）

WORD *out_page*
変換先コード・ページ（入力）

WORD *option*
オプション（入出力）

入力 入力オプションには次のものがあります。

ビット 15-9
0 に予約済み

ビット 8
変換されたストリングには SO/SI がある

ビット 7-3
0 に予約済み

ビット 2
編集不能 SBCS テーブルを使用

ビット 1
ソース・ストリングは DBCS で開始

ビット 0
ソース・ストリングには SO/SI がある

出力 出力オプションには次のものがあります。

4 DBCS で終了
0 非 DBCS で終了

注:

1. ビット 8 およびビット 0 は次のようにセットします。

   PC からホストへの変換 ビット 8=1

   PC からホストへの変換 ビット 0=0

   ホストから PC への変換 ビット 8=0

   ホストから PC への変換 ビット 0=1

2. **TrnsDt** が使用するカスタマイズ済みのテーブルの名前を指定するには、**SYSCTBL.EXE** を使用します。 SBCS ストリングを変換するには、**TrnsDt** は、**Option** パラメーターのビット 2 を FALSE にセットして、カスタマイズ済みのテーブルを使用します。 ビット 2 がセットされていてもテーブルの名前が指定されていない場合は、**TrnsDt** は省略時のテーブルを使用します。 **SYSCTBL.EXE** を使用してテーブルの名前が指定されているときに DBCS ストリングを変換するには、 **TrnsDt** は常にカスタマイズ済みのテーブルを使用します。 その場合は、ビット 2 の **Option** パラメーターは使用されません。
3. 一般に、**TrnsDt** では、ホスト・データに SO/SI 制御文字が対になって含まれていることが必要です。 しかし、混合データ・ストリングの一部を変換するには、データは、SO 制御文字なしの 2 バイト文字で始まっていなければなりません。その場合は、データは 2 バイト文字を識別しません。ビット 1 はこのような場合に役立ちます。 ビット 1 を 1 にセットすると、**TrnsDt** は、バッファーの先頭を2バイト文字または SO 制御文字として処理します。

PassParm フォーマット

0 NO_ERROR

3 ERROR_FILE_NOT_FOUND

TrnsDt は、指定されたコードの変換に使用するテーブルを見つけることができません。

87 ERROR_INVALID_PARAMETER

パラメーターが無効です。

111 ERROR_BUFFER_OVERFLOW

宛先バッファーが小さすぎます。

150 ERROR_MEMORY_ALLOCATE

メモリー割振りエラー。

PassParm フォーマット

TrnsDt の終了コードとオプション・パラメーターを使用すれば、小さいバッファーでも大規模なデータ変換を取り扱うことができます。まず、小さいソース・バッファーと、その 2 倍または 3 倍程度の大きさの宛先バッファー（PC からホストへの場合）を使用して **TrnsDt** を開始し、受信した終了コードに基づいて、変換がどのように終了したかを確認します。 そして、その結果に応じて操作を進めます。

たとえば、変換の結果として 2 バイト文字が 2 つの部分に分割されたり、SO制御文字と SI 制御文字との間で不完全に終了したりしている場合は、バッファー・ポインターとその位置を定義してから、次の呼出しを行います。

PassParm フォーマット

次の例は、ホスト・コード `0x4040` を PC コードに変換しています。

```
#include "trnsdt.h"

PASSSTRUCT      passparm;
char            bufs[20], buft[20];
int             rc;
//Setup the string to be translated
bufs[0] = 0x0e;
bufs[1] = 0x40;
bufs[2] = 0x40;
bufs[3] = 0x4f;
//Setup the parameter
passaparm.parm_length = 24;
passparm.exit_code    = 0;
passaparm.in_length   = 4;
passaparm.in_addr     = &[0];
passaparm.out_length  = 20;
passaparm.out_addr    = &[0];

passaparm.trns_id    = 0;
passaparm.in_page    = 930;
passaparm.out_page   = 932;
passaparm.option     = 1;
//Translate the string via TrnsDt
if (rc = TrnsDt(&passaparm))
   printf("Error Return Code = %d\n\r", rc);
   printf("Exit Code = %d\n\r", passaparm.exit_code);
   exit(0);
else
  .....
```

第15章 共通サービス verb (CSV)

Communications Serverは、共通サービス API 用として次の verb を提供しています。

GET_CP_CONVERT_TABLE

この verb は、 1 つのコード・ページから別のコード・ページへの変換テーブルを作成するユーティリティー・サービスを提供します。 この verb が戻す 256 バイトの変換テーブルを使用して、アプリケーションは、文字を対象とするテーブル・ルックアップにより文字ストリングを変換することができます。

データの変換が必要になるのは、プログラムが、異なるコード・ページでコード化されたデータを期待しているノードと通信するときです。

VCB 構造

```
struct get_cp_convert_table
  {
  unsigned short  opcode;          /* Verb identifying operation code.      */
  unsigned char   opext;           /* Reserved.                             */
  unsigned char   reserv2;         /* Reserved.                             */
  unsigned short  primary_rc;      /* Primary return code from verb.        */
  unsigned long   secondary_rc;    /* Secondary (qualifying) return code.   */
  unsigned short  source_cp;       /* Source code page for conversion table */
  unsigned short  target_cp;       /* Target code page for conversion table */
  unsigned char  *conv_tbl_addr;   /* Address to put conversion table at    */
  unsigned char   char_not_fnd;    /* Character not found option: either    */
                                   /* substitute character or round trip    */
  unsigned char   substitute_char; /* Substitute character to use.          */
  } GET_CP_CONVERT_TABLE;
```

指定パラメーター

source_code_page

置換文字が指定されるコード・ページ番号。コード・ページの番号は次のいずれかです。

- ASCII コード・ページ (10 進数)
  - 437 米国 IBM PC
  - 813 ギリシャ
  - 819 ANSI 規格
  - 850 複数言語
  - 852 チェコ/スロバキア/ハンガリー/ポーランド/旧ユーゴスラビア
  - 855 キリル語
  - 857 トルコ
  - 860 ポルトガル
  - 861 アイスランド
  - 862 ヘブライ語
  - 863 カナダ・フランス語
  - 864 アラビア語

- 865 北欧語
- 866 キリル語
- 874 タイ
- 912 ラテン語 2
- 915 キリル語
- 920 トルコ
- 1250 ラテン語 2
- 1251 ANSI

• EBCDIC コード・ページ（10 進数）
- 037 米国/カナダ・フランス語/オランダ/ポルトガル/ブラジル
- 273 ドイツ/オーストリア
- 277 デンマーク/ノルウェー
- 278 フィンランド/スウェーデン
- 280 イタリア
- 284 ラテン・アメリカ/スペイン
- 285 英国
- 297 フランス
- 500 ベルギー/スイス・フランス語/スイス・ドイツ語
- 870 チェコスロバキア/ハンガリー/ポーランド/ユーゴスラビア
- 871 アイスランド
- 875 ギリシャ
- 1025 キリル語
- 1026 トルコ

• ユーザー定義のコード・ページ
- 65280 〜 65534
- ユーザー定義コード・ページを使用するときは、まず、以下のように、ユーザーが定義した CPT ファイルへのパスをレジストリーのエントリーに定義する必要があります。

```
HKEY_LOCAL_MACHINE/SOFTWARE/IBM/Communications Server/CurrentVersion/COMCPT
```

注: 変換元コード・ページと変換先コード・ページ上の同じ文字についてのみ、相互間の変換が保証されます。 標準で設計されている文字ペアであっても、単に互いに類似しているだけでは、通常相互に変換されません。

target_code_page

変換される目的ストリングのコード・ページ番号。この番号は、**source_code_page** の項に示してあるもののうちのどれでも構いません。

convert_table addr

256 バイトの変換テーブルを受け入れるバッファーのアドレス。 このバッファーは読取り/書込みセグメント内にあるものである必要があります。

character_not_found

変換元コード・ページ内の文字が変換先コード・ページにない場合に行うアクション。 次のいずれか 1 つを指定してください。

SV_ROUND_TRIP

このオプションを使用すると、変換元コード・ページと変換先コード・ページを逆転する形で変換テーブルを生成した場合に、変換元から変換先コード・ページに変換し、さらにもう一度逆に変換したときに元の文字になるように、変換テーブルに値が格納されます。 **ROUND_TRIP** オプションを有効に稼働させるには、両方のテーブル生成についてこのオプションを選択する必要があります。

SV_SUBSTITUTE

パラメーター **substitute_character** に指定された文字を変換テーブルに格納します。

substitute_character

変換元コード・ページ内の文字が変換先コード・ページになく、**character_not_found** パラメーターが **SV_SUBSTITUTE** にセットされている場合に、変換テーブルに格納されるバイト。

OK 戻りコードは、**GET_CP_CONVERT_TABLE** verb が正常に実行されたことを示します。

戻りコードが OK のときは、次のパラメーターが戻されます。

convert_table

CONV_table_addr に指定したアドレスに変換テーブルが作成されました。

primary_rc

SV_PARAMETER_CHECK

secondary_rc

SV_INVALID_CHAR_NOT_FOUND

SV_INVALID_DATA_SEGMENT
SV_INVALID_SOURCE_CODE_PAGE
SV_INVALID_TARGET_CODE_PAGE

CONVERT

この verb は ASCII 文字ストリングを EBCDIC へ、また EBCDIC 文字ストリングを ASCII へ変換します。

プログラムがデータ変換を行うのは、EBCDIC データを予期しているノードと通信するとき、または、APPC などのように EBCDIC 名を必要とするインターフェースを介して渡すために、名前を変換する必要があるときなどです。

注: CONVERT verb は DBCS ではサポートされません。 2バイト文字を含むストリングは、**TrnsDt** を使用して変換できます。

VCB 構造

```
struct convert  {
  unsigned short  opcode;          /* Verb identifying operation code.      */
  unsigned char   opext;           /* Reserved.                             */
  unsigned char   reserv2;         /* Reserved.                             */
  unsigned short  primary_rc;      /* Primary return code from verb.        */
  unsigned long   secondary_rc;    /* Secondary (qualifying) return code.   */
  unsigned char   direction;       /* Direction of conversion - ASCII to    */
                                   /* EBCDIC or vice-versa.                 */
  unsigned char   char_set;        /* Character to use for the conversion   */
                                   /* A, AE, or user-defined G.             */
  unsigned short  len;             /* Length of string to be converted.     */
  unsigned char  *source;          /* Pointer to string to be converted.    */
  unsigned char  *target;          /* Address to put converted string at.   */
  } CONVERT;
```

戻りパラメーターの構文

return_code OK エラー・コード

指定パラメーター

direction

コード変換の特性。

SV_ASCII_TO_EBCDIC

ASCII 文字を EBCDIC に変換します。

SV_EBCDIC_TO_ASCII

EBCDIC 文字を ASCII に変換します。

character_set

ソース・ストリング内での使用が許されている文字セット。 **CONVERT** で使用するために、 3つのタイプの ASCII/EBCDIC 変換テーブルを指定できます。すなわち、SV_A, SV_AE, および SV_G です。タイプ A およびタイプ AE のテーブルは Communications Serverの中で定義されています。タイプ G はユー

ザー定義のテーブルです。このテーブルのファイル名は、Communications Server 構成メニューで指定します。 タイプ G のテーブルを使用すると、任意の文字も変換することができます。

変換テーブルのフォーマットは、32 文字 32 行から成っています。 1 つの行は、16 個の印刷可能 16 進文字と、それに続く 1 個の復帰改行文字から成っています。 最初の 16 行は、ASCII から EBCDIC への変換のための情報を提供します。 次の 16 行は、EBCDIC から ASCII への変換のための情報を提供します。 テーブルには 32 行すべてが含まれていなければなりません。

変換を行うときに、Communications Serverは、各入力文字に相当する数値を、変換テーブルへの 0 起点指標として使用します。 この指標は、変換する文字の 16 進値を含むテーブル位置を指定します。 たとえば、テーブル内の48番目の位置に X'F0'という値が入っているとします。
この場合、Communications Serverは、48 (X'30') の値を持つ入力文字を、240 (X'F0') という値に変換します。

テーブル A

テーブル A は、大文字の A ～ Z、数字 0 ～ 9、および特殊文字 $、#、@ を変換します。 ソース・ストリングの最初の 1 文字は、英字の大文字か、または 3 つの特殊文字のどれかでなければなりません。そうでない場合は、変換は行われず、 2 次戻りコード INVALID_FIRST_CHARACTER が戻されます。 ASCII から EBCDIC への変換では、小文字の ASCII 文字は大文字の EBCDIC 文字へ変換されます。

後書きブランク（ソース・ストリングの末尾のブランク）は、どちらの方向の変換の場合もブランクに変換されます。 これに対して、組込みブランク（埋め込まれたブランク）は X'00' に変換されます。

ソース文字のどれかが X'00' に変換された場合は、CONVERSION_ERROR が戻されます。 ただし、全体の変換は完了します。

テーブル AE

テーブル AE は英数字 (A から Z、a から z、0 から 9)、特殊文字 $、#、および @、およびピリオド (.) を変換します。.ストリングの最初の文字に関する制約はありません。

後書きブランク（ソース・ストリングの末尾のブランク）は、どちらの方向の変換の場合もブランクに変換されます。 これに対して、組込みブランク（埋め込まれたブランク）は X'00' に変換されます。

ソース文字のどれかが X'00' に変換された場合は、CONVERSION_ERROR が戻されます。 ただし、全体の変換は完了します。

テーブル G

Gテーブルは、任意の文字を他の任意の文字に（ACSCII から EBCDIC へ、または EBCDIC から ASCII へだけでなく）変換するために使用できます。ただし、テーブルの前半を使用するには **CONVERT** verb で ASCII_TO_EBCDIC を指定し、後半を使用するには EBCDIC_TO_ASCII を指定する必要があります。

Communications Serverは次のレジストリーを調べます。

HKEY_LOCAL_MACHINE/SOFTWARE/IBM/Communications Server/CurrentVersion/COMTBLG

この項目から、G テーブルの完全パス名を入手します。 **CONVERT** verb のその他の指定パラメーターには、次のものがあります。

length 変換する文字数。

ストリングの長さは、**source_addr** または **target_addr** に割り振られているセグメント・サイズを超えるものであってはなりません。

source_addr
変換する文字ストリングのアドレス。

target_addr address
変換した文字ストリングを受け取るアドレス。

注: アプリケーションがソース・ストリングを保存することを必要としない場合は、**source_addr** と **target_addr** に同じ変数を指定することができます。

戻りパラメーター

OK 戻りコードは、**CONVERT** verb が正常に実行されたことを示します。

次に、**CONVERT** verb に関連した 1 次および 2 次エラー戻りコードと、戻りコードの説明がある場所を示します。

primary_rc
SV_PARAMETER_CHECK

secondary_rc
SV_INVALID_DIRECTION

SV_TABLE_ERROR
SV_INVALID_CHARACTER_SET
SV_INVALID_FIRST_CHARACTER
SV_CONVERSION_ERROR
SV_INVALID_DATA_SEGMENT

primary_rc
SV_UNEXPECTED_DOS_ERROR

第4部 EHNAPPC API

第16章 EHNAPPC アプリケーション・プログラム・インターフェース

このインターフェースはサーバーでは使用できません。

EHNAPPC コミュニケーション API は、パーソナル・コンピューターと AS/400 システムとの間の共同処理アプリケーションを作成する方法を提供します。 このインターフェースを使用すると、プログラマーは、低レベルの通信プログラミングやハードウェアの接続タイプにかかわらずに済みます。 アプリケーション・プログラマーは、この API を使用するときは、AS/400 と PC の両方のプログラムを作成する必要があります。 ホスト・アプリケーションでアクセスできるほとんどすべてのものが、パートナー PC アプリケーションでもアクセスできます。 この API は、パフォーマンスが重要課題であるアプリケーションに使用することができます。

この章では、Windows NT および Windows 95 Communications Server クライアントのための 32 ビット EHNAPPC API を構成するルーチン、データ構造、および戻りコードについて説明します。 これらの機能は、そのほとんどが Windows 3.1 の 16 ビット API でも使用できます。

EHNAPPC プログラムの作成

下記の表は、EHNAPPC プログラムをコンパイルし、リンクするのに必要な、提供されるヘッダー・ファイルとライブラリーで使われているソース・モジュールを示しています。

表 18. オペレーティング・システムのためのヘッダー・ファイルとライブラリー

オペレーティング・システム	ヘッダー・ファイル	ライブラリー	DLL 名
WINNT & WIN95	E32APPC.H	E32APPC.LIB	E32APPC.DLL
WIN3.1	EHNAPPC.H	EHNAPPC.LIB	EHNAPPC.DLL

EHNAPPC ルーチン

以下に、各クライアント Windows API ルーチンを次の事項について説明します。

- 目的
- プロシージャー宣言
- パラメーター
- 戻りコード

EHNAPPC_Allocate

目的

この機能は、パートナー・トランザクション・プログラムとの会話を開始させます。

プロシージャー宣言

```
#include <WINDOWS.H>
include "E32APPC.H"
extern int EHNAPPC_Allocate
HWND                    hWnd,
unsigned                nBufferLength,
ConversationType        bType,
SyncLevelEnumb          SynchLevel,
LPSTR                   lpszLocationName,
LPSTR                   lpszTpn,
int                     nPipLength,
LPVOID                  lpPipData,
LPDWORD                 lpdwConversation);
```

パラメーター

hWnd は、アプリケーションの現行ウィンドウを指定します。

bType は、割り振るべき会話のタイプを指定します。 指定できる値は次のとおりです。

EHNAPPC_BASIC (0)

EHNAPPC_MAPPED (1)

bSynchLevel は、ローカル・プログラムとパートナー・プログラムの同期レベルを指定します。 指定できる値は次のとおりです。

EHNAPPC_SYNCLEVELNONE (0)

EHNAPPC_SYNCLEVELCONFIRM (1)

lpszLocationName は、ホスト・システム名を指定する NULL 終了の文字ストリングを指します。 このポインターを NULL にセットすると、省略時のシステムが使用されます。

lpszTpn は、パートナー・プログラム名を指定する NULL 終了の文字ストリングを指します。 最初の文字が 0x40 より小さいと、ASCII から EBCDIC への変換は行われません。

nPipLength は、プログラム初期設定パラメーター (PIP) データの長さを指定します。この変数が 0 の場合は、PIP データは送られません。

lpPipData は PIP データを指します。 PIP データは GDS フォーマットの EBCDIC である必要があります。

lpdwConversation は、これ以降の呼出しで使用されるハンドルを戻すためのダブルワード変数を指します。 このハンドルは、各会話で固有の値をもちます。

戻りコード

戻りコードについては、304ページの『EHNAPPC API の戻りコード』を参照してください。

EHNAPPC_Confirm

目的

この機能は、パートナーがこれまでに送信されたデータを受信していることの確認を要求します。

プロシージャー宣言

```
#include <WINDOWS.H>
#include "E32APPC.H"
extern int far pascal EHNAPPC_Confirm(
HWND           hWnd,
DWORD          dwConversation,
LPBYTE         lpRequestToSendRcvd);
```

パラメーター

hWnd は、アプリケーションの現行ウィンドウを指定します。

dwConversation は、EHNAPPC_Allocate または EHNAPPC_ExtendedAllocate のいずれかから戻される会話ハンドルを指定します。

lpRequestToSendRcvd は、パートナー・トランザクション・プログラムが REQUEST_TO_SEND verb を発行したかどうかを記憶するために使用される変数を指します。 TRUE 標識の値は、パートナー・トランザクション・プログラムが REQUEST_TO_SEND verb を発行したことを示します。

戻りコード

戻りコードについては、304ページの『EHNAPPC API の戻りコード』を参照してください。

EHNAPPC_Confirmed

目的

この機能は、確認を要求したパートナーに確認を送信します。

プロシージャー宣言

```
#include <WINDOWS.H>
#include "E32APPC.H"
extern int far pascal EHNAPPC_Confirmed(
     HWND          hWnd,
     DWORD         dwConversation);
```

パラメーター

hWnd は、アプリケーションの現行ウィンドウを指定します。

dwConversation は、EHNAPPC_Allocate または EHNAPPC_ExtendedAllocate のいずれかから戻される会話ハンドルを指定します。

戻りコード

戻りコードについては、304ページの『EHNAPPC API の戻りコード』を参照してください。

EHNAPPC_Deallocate

目的

この機能は、割り振られている会話の割振りを解除します。

プロシージャー宣言

```
#include "E32APPC.H"
extern int far pascal EHNAPPC_Deallocate(
         HWND                hWnd,
         DWORD               dwConversation,
         DeallocateEnum      bType);
```

パラメーター

hWnd は、アプリケーションの現行ウィンドウを指定します。

dwConversation は、EHNAPPC_Allocate または EHNAPPC_ExtendedAllocate のいずれかから戻される会話ハンドルを指定します。

bType は、クライアントが実行しようとする割振り解除のタイプを指定します。指定できる値は次のとおりです。

EHNAPPC_DEALLOCATESYNCLEVEL (0)

EHNAPPC_DEALLOCATEFLUSH (1)

EHNAPPC_DEALLOCATEABEND (2)

戻りコード

戻りコードについては、304ページの『EHNAPPC API の戻りコード』を参照してください。

EHNAPPC_ExtendedAllocate

目的

この機能は、パートナー・トランザクション・プログラムとの会話を開始します。そのセキュリティーまたはモードを指定変更することもできます。

プロシージャー宣言

```
#include <WINDOWS.H>
#include "E32APPC.H"
extern int EHNAPPC_ExtendedAllocate(
HWND                    hWnd,
unsigned                nBufferLength,
ConversationType        bType,
SyncLevelEnum           bSynchLevel,
LPSTR                   lpszLocationName,
LPSTR                   lpszTpn,
LPSTR                   lpszModeName,
SecurityType            bSecurityType,
LPSTR                   lpszUserId,
LPSTR                   lpszPassword,
in                      nPipLength,
LPVOID                  lpPipData,
LPDWORD                 lpdwConversation);
```

パラメーター

hWnd は、アプリケーションの現行ウィンドウを指定します。

bType は、割り振るべき会話のタイプを指定します。 指定できる値は次のとおりです。

EHNAPPC_BASIC (0)

EHNAPPC_MAPPED (1)

bSynchLevel は、ローカル・プログラムとパートナー・プログラムの同期レベルを指定します。 指定できる値は次のとおりです。

EHNAPPC_SYNCLEVELNONE (0)

EHNAPPC_SYNCLEVELCONFIRM (1)

lpszLocationName は、ホスト・システム名を指定する NULL 終了の文字ストリングを指します。このポインターを NULL にセットすると、省略時のシステムが使用されます。

lpszTpn は、パートナー・プログラム名を指定する NULL 終了の文字ストリングを指します。 最初の文字が X'40' より小さいと、ASCII から EBCDIC への変換は行われません。

lpszModeName は、指定変更したいモードを指定します。 モードの命名規則は、次のとおりです。

モード名 は 1 文字から 8 文字までです。 各部分の最初の文字は、英字の大文字 (A-Z) または特殊文字 (@、#、$) のいずれかである必要があります。 残りの文字は、英大文字 (A-Z)、数字 (0-9) または特殊文字 (@、#、$) とすることができます。

bSecurityType は、使用したいセキュリティー・タイプを指定します。 指定できる値は次のとおりです。

EHNAPPC_SECURITY_NONE (0)

EHNAPPC_SECURITY_SAME (1)

EHNAPPC_SECURITY_PGM (2)

lpszUserId は、ユーザー ID を含む NULL 終了の文字ストリングを指します。 文字ストリングの最大の長さは 10 文字です。

lpszPassword は、パスワードを含む NULL 終了の文字ストリングを指します。文字ストリングの最大の長さは 10 文字です。

nPipLength は、PIP データの長さです。この変数が 0 の場合は、PIP データは送られません。

lpPipData は PIP データを指します。 PIP データは GDS フォーマットの EBCDIC である必要があります。

lpdwConversation は、これ以降の呼出しで使用されるハンドルを戻すためのダブルワード変数を指します。

戻りコード

戻りコードについては、304ページの『EHNAPPC API の戻りコード』を参照してください。

EHNAPPC_Flush

目的

この機能は、クライアントに、そのバッファーに入っているデータをすべて送信させます。

プロシージャー宣言

```
#include <WINDOWS.H>
#include "E32APPC.H"
extern int EHNAPPC_Flush(
     HWND          hWnd,
     DWORD         dwConversation);
```

パラメーター

hWnd は、アプリケーションの現行ウィンドウを指定します。

dwConversation は、EHNAPPC_Allocate または EHNAPPC_ExtendedAllocate のいずれかから戻される会話ハンドルを指定します。

戻りコード

戻りコードについては、304ページの『EHNAPPC API の戻りコード』を参照してください。

EHNAPPC_GetAttributes

目的

指定された会話の属性を戻します。この中には、ローカルおよびパートナーのトランザクション・プログラムの LU 名、処理同期のレベル、およびセキュリティーのために提供されているユーザー ID が含まれます。

プロシージャー宣言

```
#include <WINDOWS.H>
#include "E32APPC.H"
extern unsigned EHNAPPC_GetAttributes(
     HWND             hWnd,
     DWORD            dwConversation,
     LPBYTE           lpbSyncLevel,
     LPSTR            lpszModeName,
     LPSTR            lpszLuName,
     LPSTR            lpszPluName,
     LPSTR            lpszUserId);
```

パラメーター

hWnd は、アプリケーションの現行ウィンドウを指定します。

dwConversation は、EHNAPPC_Extended から戻される会話ハンドルを指定します。

lpbSyncLevel は、同期レベルを戻すために使用される 1 バイトの変数を指します。

lpszModeName は、8 文字のモード名を戻すために使用される、NULL で終了する文字ストリングを指します。

lpszLuName は、ローカル・トランザクション・プログラムの LU を戻すために使用される、NULL で終了する文字ストリングを指します。

lpszPluName パートナー LU の名前を戻すために使用される、NULL 文字で終了する文字ストリングを指します。

lpszUserId は、この会話を確立するために使用されたユーザー ID を戻すための、NULL で終了する文字ストリングを指します。

戻りコード

戻りコードについては、304ページの『EHNAPPC API の戻りコード』を参照してください。

EHNAPPC_GetCapabilities

目的

この機能は、現在ロードされているクライアントの機能を示すデータ構造を満たします。

プロシージャー宣言

```
#include <WINDOWS.H>
#include "E32APPC.H"
extern unsigned EHNAPPC_GetCapabilities(
     HWND      hWnd,
     LPSTR     lpList);
```

パラメーター

hWnd は、アプリケーションの現行ウィンドウを指定します。

lpList は、機能情報を検索するために使用される機能リストを指します。 機能リストは、ヘッダーと、それに続く可変数の機能構造から構成されます。 入力では、このリストには照会したい機能を指定します。 出力では、この中には機能情報が入っています。

注： この構造については、 302ページの『appcrtrcap_hdr』、303ページの『appcrtrcap_mult』、および 303ページの『appcrtrcap_query』 を参照してください。

戻りコード

戻りコードについては、304ページの『EHNAPPC API の戻りコード』を参照してください。

EHNAPPC_GetDefaultSystem

目的

この機能は、クライアントが接続されている省略時のシステム名を戻します。

プロシージャー宣言

```
#include <WINDOWS.H>
#include "E32APPC.H"
extern unsigned pascal EHNAPPC_GetDefaultSystem(
     HWND      hWnd,
     LPSTR     lpszDefSysName);
```

パラメーター

hWnd は、アプリケーションの現行ウィンドウを指定します。

lpszDefSysName は、省略時のシステム名を戻すために使用される文字バッファーを指します。 システム名は、このバッファーに、NULL で終了する文字ストリングとして記憶されます。

戻りコード

戻りコードについては、304ページの『EHNAPPC API の戻りコード』を参照してください。

EHNAPPC_IsClientLoaded

目的

この機能は、クライアントがメモリーにロードされているかどうかを判別します。

プロシージャー宣言

```
#include <WINDOWS.H>
#include "E32APPC.H"
extern bool EHNAPPC_IsClientLoaded(
     HWND     hWnd);
```

パラメーター

hWnd は、アプリケーションの現行ウィンドウを指定します。

戻りコード

戻りコードは、Communications Server クライアントがロードされていなければ FALSE (0) になります。 ロードされていれば、戻りコードは TRUE (1) です。

EHNAPPC_PrepareToReceive

目的

この機能は、プログラムがデータを受信できるように準備します。 この機能に続けて EHNAPPC_ReceiveImmediate を使用すると、EHNAPPC_ReceiveAndWait を使用するのと同じになります。

プロシージャー宣言

```
#include <WINDOWS.H>
#include "E32APPC.H"
extern int EHNAPPC_PrepareToReceive(
     HWND          hWnd,
     DWORD         dwConversation);
```

パラメーター

hWnd は、アプリケーションの現行ウィンドウを指定します。

dwConversation は、EHNAPPC_Allocate または EHNAPPC_ExtendedAllocate のいずれかから戻される会話ハンドルを指定します。

戻りコード

戻りコードについては、304ページの『EHNAPPC API の戻りコード』を参照してください。

EHNAPPC_QueryConfiguredSystems

目的

この機能は、Communications Server に構成されているシステムの名前を戻します。

プロシージャー宣言

```
#include <WINDOWS.H>
#include "E32APPC.H"
extern unsigned EHNAPPC_QueryConfiguredSystems(
     HWND               hWnd,
     LPINT              lpSysCount,
     LPSYSSTRUC         lpSys);
```

パラメーター

hWnd は、アプリケーションの現行ウィンドウを指定します。

lpSysCount は、接続されているシステム数を戻すために使用される整変数を指します。

lpSys は、システム名を戻すために使用される AS400_Sys 構造を指します。 省略時のシステムは、構造の最初に載せられます。 AS400_Sys 構造については、302ページの『AS400_SYS』を参照してください。

戻りコード

戻りコードについては、304ページの『EHNAPPC API の戻りコード』を参照してください。

EHNAPPC_QueryConvState

目的

この機能は、指定された会話の状態を戻します。

プロシージャー宣言

```
#include <WINDOWS.H>
#include "E32APPC.H"
extern unsigned pascal EHNAPPC_QueryConvState(
     HWND          hWnd,
     DWORD         dwConversation);
```

パラメーター

hWnd は、アプリケーションの現行ウィンドウを指定します。

dwConversation は、EHNAPPC_Allocate または EHNAPPC_ExtendedAllocate のいずれかから戻される会話ハンドルを指定します。

戻りコード

戻り値は会話の現在の状態を示します。 指定できる値は次のとおりです。

EHNAPPC_RESET_STATE (0)

EHNAPPC_SEND_STATE (1)

EHNAPPC_RECEIVE_STATE (2)

EHNAPPC_RCVD_CONF_STATE (3)

EHNAPPC_RCVD_CONF_SEND_STATE (4)

EHNAPPC_RCVD_CONF_DEALL_STATE (5)

EHNAPPC_PEND_DEALLOCATE_STATE (6)

EHNAPPC_INVALID_STATE (7)

EHNAPPC_QueryFullSystems

目的

この機能は、クライアントが接続されているシステムの名前とネットワーク名を戻します。

プロシージャー宣言

```
#include <WINDOWS.H>
#include "E32APPC.H"
extern unsigned EHNAPPC_QueryFullSystems(
     HWND            hWnd,
     LPINT           lpSysCount,
     LPFULLSYSSTRUC  lpSys);
```

パラメーター

hWnd は、アプリケーションの現行ウィンドウを指定します。

lpSysCount は、接続されているシステム数を戻すために使用される整変数を指します。

lpSys は、システム名を戻すために使用される AS400_Sys 構造を指します。

戻りコード

戻りコードについては、304ページの『EHNAPPC API の戻りコード』を参照してください。

EHNAPPC_QueryUserid

目的

この機能は、指定されたシステムに接続するために使用されているユーザー ID を戻します。

プロシージャー宣言

```
#include <WINDOWS.H>
#include "E32APPC.H"
extern unsigned EHNAPPC_QueryUserId(
     HWND             hWnd,
     LPSTR            lpszLocationName,
     LPSTR            lpszUserId);
```

パラメーター

hWnd は、アプリケーションの現行ウィンドウを指定します。

lpszLocationName は、照会したいシステム名を含んでいる、NULL で終了する文字ストリングを指します。省略時システムのユーザー ID を照会するには、NULL と指定します。lpszUserId は、指定のシステムのユーザー ID を戻すために使用される、NULL で終了する文字ストリングを指します。

戻りコード

戻りコードについては、304ページの『EHNAPPC API の戻りコード』を参照してください。

EHNAPPC_QuerySystems

目的

この機能は、クライアントが接続されているシステムの名前を戻します。

プロシージャー宣言

```
#include <WINDOWS.H>
#include "E32APPC.H"
extern unsigned EHNAPPC_QuerySystems(
     HWND               hWnd,
     LPINT              lpSysCount,
     LPSYSSTRUC         lpSys);
```

パラメーター

hWnd は、アプリケーションの現行ウィンドウを指定します。

lpSysCount は、接続されているシステム数を戻すために使用される整変数を指します。

lpSys は、システム名を戻すために使用される AS400_Sys 構造を指します。 省略時のシステムは、構造の最初に載せられます。 AS400_Sys 構造については、302ページの『AS400_SYS』を参照してください。

戻りコード

戻りコードについては、304ページの『EHNAPPC API の戻りコード』を参照してください。

EHNAPPC_ReceiveAndWait

目的

この機能は、会話に到着する情報を待機し、次にその情報を受け取ります。

プロシージャー宣言

```
#include "E32APPC.H"
extern int EHNAPPC_ReceiveAndWait(
     HWND           hWnd,
     DWORD          dwConversation,
     FillEnu        bFill,
     int            nMaxLength,
     LPVOID         lpReceiveData,
     LPBYTE         lpWhatReceived,
     LPBYTE         lpRequestToSendRcvd,
     LPWORD         lpReceiveDataLength );
```

パラメーター

hWnd は、アプリケーションの現行ウィンドウを指定します。

dwConversation は、EHNAPPC_Allocate または EHNAPPC_ExtendedAllocate のいずれかから戻される会話ハンドルを指定します。

bFill は、プログラムがデータを受信するときの形式を指示します。 指定できる値は次のとおりです。

EHNAPPC_BUFFER (0)（バッファーを埋める）

EHNAPPC_LL (1)（完全なまたは切り捨てられた論理レコードを受信する）

nMaxLength は、受け入れることのできる最大のデータ量を指示します。

lpReceiveData は、データを受けとるバッファーを指します。

lpWhatReceived は、クライアントがすでに受信しているものを示します。 指定できる値は次のとおりです。

EHNAPPC_DATA (0)

EHNAPPC_DATACOMPLETE (1)

EHNAPPC_DATAINCOMPLETE (2)

EHNAPPC_RECEIVEDCONFIRM (3)

EHNAPPC_RECEIVEDCONFIRMSEND (4)

EHNAPPC_RECEIVEDCONFIRMDEALLOC (5)

EHNAPPC_RECEIVEDSEND (6)

lpRequestToSendRcvd は、パートナー・トランザクション・プログラムが REQUEST_TO_SEND verb を発行したかどうかを記憶するために使用する変数を指します。 TRUE (1) 標識の値は、パートナー・トランザクション・プログラムが REQUEST_TO_SEND verb を発行したことを示します。

lpReceiveDataLength は、クライアントが受信したデータ量を戻すために使用される変数を指します。

戻りコード

戻りコードについては、304ページの『EHNAPPC API の戻りコード』を参照してください。

EHNAPPC_ReceiveImmediate

目的

この機能は、何かが受信されているかどうかをチェックします。 受信されていれば、そのデータが戻されます。

プロシージャー宣言

```
#include <WINDOWS.H>
#include "E32APPC.H"
extern int EHNAPPC_ReceiveImmediate(
     HWND           hWnd,
     DWORD          dwConversation,
     FillEnum       bFill,
     int            nMaxLength,
     LPVOID         lpReceiveData,
```

```
LPBYTE          lpWhatReceived,
LPBYTE          lpRequestToSendRcvd,
LPWORD          lpReceiveDataLength );
```

パラメーター

hWnd は、アプリケーションの現行ウィンドウを指定します。

dwConversation は、EHNAPPC_Allocate または EHNAPPC_ExtendedAllocate のいずれかから戻される会話ハンドルを指定します。

bFill は、プログラムがデータを受信するときの形式を指示します。 指定できる値は次のとおりです。

EHNAPPC_BUFFER (0)（バッファーを埋める）

EHNAPPC_LL (1)（完全なまたは切り捨てられた論理レコードを受信する）

nMaxLength は、受け入れることのできる最大のデータ量を指示します。

lpReceiveData は、データを受けとるバッファーを指します。

lpWhatReceived は、クライアントが受信したものを示します。 指定できる値は次のとおりです。

EHNAPPC_DATA (0)

EHNAPPC_DATACOMPLETE (1)

EHNAPPC_DATAINCOMPLETE (2)

EHNAPPC_RECEIVEDCONFIRM (3)

EHNAPPC_RECEIVEDCONFIRMSEND (4)

EHNAPPC_RECEIVEDCONFIRMDEALLOC (5)

EHNAPPC_RECEIVEDSEND (6)

lpRequestToSendRcvd は、パートナー・トランザクション・プログラムが REQUEST_TO_SEND verb を発行したかどうかを記憶するために使用される変数を指します。 TRUE (1) 標識の値は、パートナー・トランザクション・プログラムが REQUEST_TO_SEND verb を発行したことを示します。

lpReceiveDataLength は、クライアントが受信したデータ量を戻すために使用される変数を指します。

戻りコード

戻りコードについては、304ページの『EHNAPPC API の戻りコード』を参照してください。

EHNAPPC_RemoteProgramStart

目的

この機能を使用すると、Windows アプリケーションでリモート AS/400 システムのプログラムを開始できます。

プロシージャー宣言

```
#include <WINDOWS.H>
#include "E32APPC.H"
extern word EHNAPPC_RemoteProgramStart(
    HWND          hWnd,
    LPSTR         lpszHostSystemName,
    LPSTR         lpszHostProgramName,
    LPSTR         lpszHostLibraryName,
    char FAR      *lpchPipData,
    WORD          wPipDataLength);
```

パラメーター

hWnd は、アプリケーションの現行ウィンドウを指定します。

lpszHostSystemName は、リモート・システムの名前を含んでいる、NULL で終了する文字ストリングを指します。 このストリングの最大長は 8 文字です。 このポインターが NULL の場合は、省略時のシステム名が使用されます。

lpszHostProgramName は、開始したいホスト・プログラムの名前を含んでいる、NULL で終了する文字ストリングを指します。

lpszHostLibraryName は、ホスト・プログラムのライブラリー・パスを含んでいる、NULL で終了する文字ストリングを指します。 このポインターが NULL の場合は、ユーザーのライブラリー・リストが検索されます。

lpchPipData は、ホスト・プログラムのためのプログラム初期設定パラメーター(PIP)データ域を指します。 このポインターが NULL の場合は、PIP データは送信されません。

wPipDataLength には、PIP データの長さが入ります。

戻りコード

戻りコードについては、304ページの『EHNAPPC API の戻りコード』を参照してください。

EHNAPPC_RqsToSend

目的

この機能は、パートナーに会話の制御権を放すように要求します。ローカル・トランザクション・プログラムが、パートナー・トランザクション・プログラムから出された Receive verb の lpWhatReceived パラメーターに EHNAPPC_RECEIVEDSEND (6) を受け取ると、クライアントは会話を送信状態にします。

プロシージャー宣言

```
#include <WINDOWS.H>
#include "E32APPC.H"
extern int EHNAPPC_RqsToSend(
     HWND          hWnd,
     DWORD         dwConversation);
```

パラメーター

hWnd は、アプリケーションの現行ウィンドウを指定します。

dwConversation は、EHNAPPC_Allocate または EHNAPPC_ExtendedAllocate のいずれかから戻される会話ハンドルを指定します。

戻りコード

戻りコードについては、304ページの『EHNAPPC API の戻りコード』を参照してください。

EHNAPPC_SendData

目的

この機能は、パートナー・トランザクション・プログラムにデータを送ります。

プロシージャー宣言

```
#include <WINDOWS.H>
#include "E32APPC.H"
extern int EHNAPPC_SendData(
     HWND      hWnd,
     DWORD     dwConversation,
     int       nSendDataLength,
     LPVOID    lpSendDataBuffer,
     LPBYTE    lpRequestToSendRcvd);
```

パラメーター

hWnd は、アプリケーションの現行ウィンドウを指定します。

dwConversation は、EHNAPPC_Allocate または EHNAPPC_ExtendedAllocate のいずれかから戻される会話ハンドルを指定します。

nSendDataLength は、送信バッファー内のデータの長さを指定します。

lpSendDataBuffer は送信バッファーのアドレスを指示します。

lpRequestToSendRcvd は、パートナー・トランザクション・プログラムが REQUEST_TO_SEND verb を発行したかどうかを記憶するために使用する変数を指

します。 TRUE 標識の値は、パートナー・トランザクション・プログラムが REQUEST_TO_SEND verb を発行したことを示します。

戻りコード

戻りコードについては、304ページの『EHNAPPC API の戻りコード』を参照してください。

EHNAPPC_SendError

目的

この機能は、エラーが検出されたことをパートナー・トランザクション・プログラムに知らせます。 この機能を使用した後、このローカル・プログラムは受信状態になります。

プロシージャー宣言

```
#include <WINDOWS.H>
#include "E32APPC.H"
extern int EHNAPPC_SendError(
         HWND      hWnd,
         DWORD     dwConversation,
         LPBYTE    lpRequestToSendRcvd);
```

パラメーター

hWnd は、アプリケーションの現行ウィンドウを指定します。

dwConversation は、EHNAPPC_Allocate または EHNAPPC_ExtendedAllocate のいずれかから戻される会話ハンドルを指定します。

lpRequestToSendRcvd は、パートナー・トランザクション・プログラムが REQUEST_TO_SEND verb を発行したかどうかを記憶するために使用する変数を指します。 TRUE 標識の値は、パートナー・トランザクション・プログラムが REQUEST_TO_SEND verb を発行したことを示します。

戻りコード

戻りコードについては、304ページの『EHNAPPC API の戻りコード』を参照してください。

EHNAPPC_StartHostProgram

目的

この機能を使用すると、Windows アプリケーションは、リモート AS/400 システム上のプログラムを開始することができます。会話は活動状態のままなので、アプリケ

ーションでは、ホスト・プログラムが実行中であることを確認することができます。アプリケーションは、会話を終了するには、EHNAPPC_Deallocate 機能を使用する必要があります。

プロシージャー宣言

```
#include <WINDOWS.H>
#include "E32APPC.H"
extern word EHNAPPC_RemoteProgramStart(
    HWND          hWnd,
    LPSTR         lpszHostSystemName,
    LPSTR         lpszHostProgramName,
    LPSTR         lpszHostLibraryName,
    char FAR      *lpchPipData,
    WORD          wPipDataLength);
```

パラメーター

hWnd は、アプリケーションの現行ウィンドウを指定します。

lpszHostSystemName は、リモート・システムの名前を含んでいる、NULL で終了する文字ストリングを指します。 このストリングの最大長は 8 文字です。 このポインターが NULL の場合は、省略時のシステム名が使用されます。

lpszHostProgramName は、開始したいホスト・プログラムの名前を含んでいる、NULL で終了する文字ストリングを指します。

lpszHostLibraryName は、ホスト・プログラムのライブラリー・パスを含んでいる、NULL で終了する文字ストリングを指します。 このポインターが NULL の場合は、ユーザーのライブラリー・リストが検索されます。

lpchPipData は、ホスト・プログラムのためのプログラム初期設定パラメーター(PIP)データ域を指します。 このポインターが NULL の場合は、PIP データは送信されません。

wPipDataLength には、PIP データの長さが入ります。

戻りコード

戻りコードについては、304ページの『EHNAPPC API の戻りコード』を参照してください。

EHNAPPC の構造

AS400_SYS

目的

この構造は、クライアントが接続しているシステムの名前を記憶するために使用されます。

プロシージャー宣言

```
struct AS400_sys
     (
unsigned char EHNAPPC_SysNameｽEHNAPPC_MAX_SYSTEMS|
     ｽEHNAPPC_SYSNAME_SYSNAME_LENGTH|;
     );
```

パラメーター

EHNAPPC_SysName は、接続システムの名前を記憶するために使用されます。 システム名は、NULL 文字で終了するストリングとして戻されます。 配列に戻される最初のシステムは、省略時システム（EHNAPPC_MAX_SYSTEMS = 32 で EHNAPPC_SYSNAME_SYSNAME_LENGTH = 10）です。

appcrtrcap_hdr

目的

これは、クライアントの機能リスト・ヘッダーの構造です。

プロシージャー宣言

```
struct appcrtrcap_hdr
     (
     unsigned char rc;
     unsigned char opcode;
     unsigned int length;
     );
```

パラメーター

rc は、機能要求のすべての戻りコードを記憶するために使用されます。

opcode は、機能入手要求のシグナルとなるものです。 この値は EHNAPPC_OC_CAPABILITIES (0x17) でなければなりません。

length は、機能リストの全長です。 この長さは、ヘッダーのサイズと各機能構造のサイズを加えたものです。

appcrtrcap_mult

目的

これは、最適の通信バッファー乗数を判別するために使用される機能構造です。

プロシージャー宣言

```
struct appcrtrcap_mult
     (
     unsigned int length;
     unsigned char identifier;
     unsigned char rc;
     unsigned int data;
     );
```

パラメーター

length は、この機能構造の長さを示します。

identifier は、最適の通信バッファー乗数のシグナルとなるものです。この値は EHNAPPC_CAP_OPTIMAL_COM_SIZE (X'02') である必要があります。

rc は、この機能要求の戻りコードを記憶するために使用されます。

data は、最適の通信バッファー乗数を戻すために使用されます。

appcrtrcap_query

目的

これは、クライアントが指定の機能をサポートするかどうかを照会するために使用されます。

プロシージャー宣言

```
struct appcrtrcap_query
     (
     unsigned int length;
     unsigned char identifier;
     unsigned char rc;
     unsigned char data;
      );
```

パラメーター

length は、この機能構造の長さを示します。

identifier は、照会したい機能を指示します。指定できる値は次のとおりです。

EHNAPPC_CAP_QUERY_CONV_STATE (3)

EHNAPPC_CAP_EXT_ALLOCATE (4)

rc は、この機能要求の戻りコードを記憶するために使用されます。

data は、指定の機能がサポートされているかどうかを戻すために使用されます。

EHNAPPC API の戻りコード

クライアント Windows API では、E32APPC.H に定義されている以下の戻りコードが使用されます。

表 19. 戻りコード

戻りコード	16 進数	説明
EHNAPPC_OK	0	コマンドは正常に完了。
ENHAPPC_DEALLOCNORMAL	1	正常に割振り解除。
ENHAPPC_PROGRAMMERNOTRUNCATION	3	プログラム・エラー。切捨てなし。
ENHAPPC_PROGRAMMERTRUNCATION	4	プログラム・エラー。切捨てあり。
ENHAPPC_PROGRAMMERPURGING	4	プログラム・エラー。除去。
ENHAPPC_RESOURCEFAILURETRY	5	資源障害の再試行。
ENHAPPC_RESOURCEFAILURENORETRY	6	資源障害の再試行なし。
ENHAPPC_UNSUCCESSFUL	7	不成功。
ENHAPPC_APPCBUSY	8	APPC 使用中。
ENHAPPC_PARMCHKINVALIDVERB	14	パラメーター・チェック（verb が正しくない）。
ENHAPPC_PARMCHKINVALIDCONVERID	15	パラメーター・チェック（会話 ID が正しくない）。
ENHAPPC_PARMCHKBUFFERCROSSEG	16	パラメーター・チェック（バッファー・クロス・セグメント）。
ENHAPPC_PARMCHKTPNAMELENGTH	17	パラメーター・チェック（トランザクション・プログラム名の長さ）。
ENHAPPC_PARMCHKINVCONVERTYPE	18	パラメーター・チェック（会話のタイプが正しくない）。
ENHAPPC_PARMCHKBADSYNCLVLALLOC	19	パラメーター・チェック（同期レベル割振りの誤り）。
ENHAPPC_PARMCHKBADRETURNCNTRL	1A	パラメーター・チェック（戻り制御の誤り）。
ENHAPPC_ ENHAPPC_PARMCHKPIPTOOLONG	1B	パラメーター・チェック（PIP データが長過ぎ）。
ENHAPPC_PARMCHKBADPARTNERNAME	1C	パラメーター・チェック（パートナー名の誤り）。
ENHAPPC_PARMCHKCONFNOTALLOWED	1D	パラメーター・チェック（確認は許されない）。
ENHAPPC_PARMCHKBADDEALLOCTYPE	1E	パラメーター・チェック（割振り解除タイプの誤り）。
ENHAPPC_PARMCHKPREPTORCVTYPE	1F	パラメーター・チェック（タイプ受信準備）。
ENHAPPC_PARMCHKBADFILLTYPE	20	パラメーター・チェック（充てんタイプの誤り）。
ENHAPPC_PARMCHKRECMAXLEN	21	パラメーター・チェック（最大長の受信）。
ENHAPPC_PARMCHKUNKNOWNSECTYPE	22	パラメーター・チェック（セキュリティー・タイプ不明）。
ENHAPPC_PARMCHKRESFLDNOTZERO	23	パラメーター・チェック（セキュリティー・タイプ不明）。
ENHAPPC_STATECHKNOTINCONFSTAT	28	状態チェック（確認状態ではない）。
ENHAPPC_STATECHKNOTINRECEIVE	29	状態チェック（受信状態ではない）。
ENAHAPPC_STATECHKREQSNDBADSTATN	2A	状態チェック（送信要求状態不良）。
ENHAPPC_STATECHKSNDINBADSTATE	2B	状態チェック（不良状態での送信）。
ENHAPPC_STATECHKSNDERRBADSTAT	2C	状態チェック（エラー送信不良状態）。
ENHAPPC_ALLOCERRNORETRY	32	割振りエラー。再試行なし。
ENHAPPC_ALLOCERRRETRY	33	割振りエラー。再試行。
ENHAPPC_ALLOCERROGMNOTAVAILNR	34	割振りエラー（プログラム利用不能）。再試行なし。
ENHAPPC_ALLOCERRTPNNOTRECOG	35	割振りエラー（トランザクション・プログラム名が認識できない）。
ENHAPPC_ALLOCERRPGMNOTAVAILR	36	割振りエラー（プログラム利用不能）。再試行。
ENHAPPC_ALLOCERRSECNOTVALID	37	割振りエラー（セキュリティー無効）。

表 19. 戻りコード (続き)

戻りコード	16 進数	説明
ENHAPPC_ALLOCERRCONVTYP	38	割振りエラー（会話タイプの不一致）。
ENHAPPC_ALLOCERRPIPNOTALLOWED	39	割振りエラー（PIP データは許されない）。
ENHAPPC_ALLOCERRPIPNOTCORRECT	3A	割振りエラー（PIP データが正しくない）。
ENHAPPC_ALLOCERRSYNCHLEVEL	3B	割振りエラー（同期レベルがサポートされていない）。
ENHAPPC_DEALLOCABENDPROGRAM	46	割振り解除。プログラムの異常終了。
ENHAPPC_INSUFFICIENTMEMORY	47	メモリー不足。
ENHAPPC_MEMORYALLOCERROR	47	メモリー割振りエラー。
ENHAPPC_MEMORYALLCERROR	48	メモリー割振りエラー。
ENHAPPC_TOOMANYCONVERSATIONS	4A	会話数超過。
ENHAPPC_CONVTABLEFULL	4B	変換テーブル一杯。
ENHAPPC_CLIENTNOTINSTALLED	4C	クライアント未導入。
ENHAPPC_CLIENTWRONGLEVEL	4C	クライアント・レベルの誤り。
ENHAPPC_PCSWINNOTLOADED	4D	PSWIN がロードされていない。
ENHAPPC_PCSWINOUTOFMEMORY	4E	PCSWIN メモリー不足。
ENHAPPC_INVALIDUSERIDLEN	4F	ユーザー ID の長さの誤り。
ENHAPPC_INVALIDPASSWORDLEN	50	パスワードの長さの誤り。
ENHAPPC_INVALIDUNAME	51	LU 長さの誤り。
ENHAPPC_UNDEFINED	63	未定義。

Windows 95 と Windows NT における 16 ビット EHNAPPC プログラムの実行

Communications Server Windows 95 および Windows NT のクライアントは、すでにある 16 ビットの EHNAPPC プログラムを Windows 95 および Windows NT で実行する機能を提供しています。このためには、Communications Server クライアントのサブディレクトリーからプログラム EHNAPPCD を開始し、それから 16 ビットの EHNAPPC アプリケーションを開始してください。このプログラムを使用すると、32 ビットの E32APPC.DLL が使用可能になります。

第17章 データ形式変更 Windows アプリケーション・プログラム・インターフェース

データ形式変更 API は、AS/400 フォーマットと PC フォーマットとの間でデータを変換する機能を提供します。この変換は、AS/400 との間でデータを送受信するときに必要になることがあります。 データ形式変更 API は、テキストおよび多数の種類の数値フォーマットの変換をサポートします。

この章では、データ形式変更 API を構成する個々のルーチンと戻りコードについて説明します。

データ形式変更 Windows API ルーチン

各データ形式変更 API ルーチンについて、下記の各項を説明します。

- 目的
- プロシージャー宣言
- パラメーター
- 戻りコード

EHNDT_ANSIToEBCDIC

目的

この機能は、ストリングを Windows ANSI コード・ページから EBCDIC に変換します。 このルーチンが ASCII-EBCDIC 変換テーブルにアクセスできるように、ルーターがロードされている必要があります。

ターゲット・ストリングの大きさが変換済みストリングを含むには小さすぎると、変換は、ターゲット・ストリングの終わりで終了します。 ターゲット・ストリングが必要な長さよりも大きければ、ストリングの終わりまでブランクが埋め込まれます。

プロシージャー宣言

```
#include <WINDOWS.H>
#include "E32APPC.H"
extern unsigned int EHNDT_ANSIToEBCDIC(
     HWND            hWnd,
     LPSTR           lpsSource,
     LPSTR           lpsTarget,
     unsigned in     wSource,
     LPWORD          lpwTarget );
```

パラメーター

hWnd は、アプリケーションの現行ウィンドウを指定します。

lpsSource は、変換したいソース (ANSI) ストリングを指します。

lpsTarget は、ターゲット（変換済み）ストリングを指します。**wSource** はソース・ストリングの長さをバイト数で示します。

lpwTarget は、ターゲット・バッファーのサイズを含んでいる 1 ワードの変数を指します。 この変数は、ターゲット・バッファーの中に入れられた、変換された文字数で更新されます。

戻りコード

機能が正常に終了すると、EHNDT_SUCCESS (X'0000') が戻ります。ルーターがロードされていないと、EHNDT_A2E_TABLE_NOT_FOUND (X'FFFC') が戻ります。 一時バッファーを割り振ろうとしたときにエラーが発生すると、EHNDT_MEMALLOC (X'FFFF') が戻ります。変換中に不適切なデータが検出されると、変換されなかった最初の文字位置に 1 を加えたものが戻りコードになります。

EHNDT_ASCIIToEBCDIC

目的

この機能は、ストリングを ASCII から EBCDIC に変換します。 このルーチンが ASCII-EBCDIC 変換テーブルにアクセスできるように、ルーターがロードされている必要があります。 ターゲット・ストリングの大きさが変換済みストリングを含むには小さすぎると、変換は、ターゲット・ストリングの終わりで終了します。 ターゲット・ストリングが必要な長さよりも大きければ、ストリングの終わりまでブランクが埋め込まれます。

プロシージャー宣言

```
#include <WINDOWS.H>
#include "E32APPC.H"
extern unsigned int EHNDT_ASCIIToEBCDIC(
     HWND            hWnd,
     LPSTR           lpsTarget,
     LPSTR           lpsSource,
     unsigned in     wSource,
     LPWORD          lpwTarget );
```

パラメーター

hWnd は、アプリケーションの現行ウィンドウを指定します。

lpsTarget は、ターゲット（変換済み）ストリングを指します。

lpsSource は、変換したいソース (ASCII) ストリングを指します。

wSource はソース・ストリングの長さをバイト数で示します。

lpwTarget は、ターゲット・バッファーのサイズを含んでいる 1 ワードの変数を指します。 この変数は、ターゲット・バッファーの中に入れられた、変換された文字数で更新されます。

戻りコード

機能が正常に終了すると、EHNDT_SUCCESS (X'0000') が戻ります。 ルーターがロードされていないと、EHNDT_A2E_TABLE_NOT_FOUND (X'FFFC') が戻ります。

変換中に不適切なデータが検出されると、変換されなかった最初の文字位置に 1 を加えたものが戻りコードになります。

EHNDT_EBCDICToANSI

目的

この機能は、ストリングを EBCDIC から Windows ANSI コード・ページに変換します。 このルーチンが ASCII-EBCDIC 変換テーブルにアクセスできるように、ルーターがロードされている必要があります。

ターゲット・ストリングの大きさが変換済みストリングを含むには小さすぎると、変換は、ターゲット・ストリングの終わりで終了します。 ターゲット・ストリングが必要な長さよりも大きければ、ストリングの終わりまでブランクが埋め込まれます。

プロシージャー宣言

```
#include <WINDOWS.H>
#include "E32APPC.H"
extern unsigned int EHNDT_EBCDICToANSI(
     HWND            hWnd,
     LPSTR           lpsTarget,
     LPSTR           lpsSource,
     unsigned int    wSource,
     LPWORD          lpwTarget );
```

パラメーター

hWnd は、アプリケーションの現行ウィンドウを指定します。

lpsTarget は、ターゲット（変換済み）ストリングを指します。

lpsSource は、変換したいソース (EBCDIC) ストリングを指します。

wSource はソース・ストリングの長さをバイト数で示します。

lpwTarget は、ターゲット・バッファーのサイズを含んでいる 1 ワードの変数を指します。 この変数は、ターゲット・バッファーの中に入れられた、変換された文字数で更新されます。

戻りコード

機能が正常に終了すると、EHNDT_SUCCESS ('0000') が戻ります。 ルーターがロードされていないと、EHNDT_E2A_TABLE_.NOT_FOUND ('FFFC') が戻ります。 変換中に不適切なデータが検出されると、変換されなかった最初の文字位置に 1 を加えたものが戻りコードになります。

EHNDT_EBCDICToASCII

目的

この機能は、ストリングを EBCDIC から ASCII に変換します。 このルーチンが ASCII-EBCDIC 変換テーブルにアクセスできるように、ルーターがロードされている必要があります。

ターゲット・ストリングの大きさが変換済みストリングを含むには小さすぎると、変換は、ターゲット・ストリングの終わりで終了します。 ターゲット・ストリングが必要な長さよりも大きければ、ストリングの終わりまでブランクが埋め込まれます。

プロシージャー宣言

```
#include <WINDOWS.H>
#include "E32APPC.H"
extern unsigned int EHNDT_EBCDICToASCII(
     HWND            hWnd,
     LPSTR           lpsTarget,
     LPSTR           lpsSource,
     unsigned int    wSource,
     LPWORD          lpwTarget );
```

パラメーター

hWnd は、アプリケーションの現行ウィンドウを指定します。

lpsTarget は、ターゲット（変換済み）ストリングを指します。

lpsSource は、変換したいソース (EBCDIC) ストリングを指します。

wSource はソース・ストリングの長さをバイト数で示します。

lpwTarget は、ターゲット・バッファーのサイズを含んでいる 1 ワードの変数を指します。 この変数は、ターゲット・バッファーの中に入れられた、変換された文字数で更新されます。

戻りコード

機能が正常に終了すると、EHNDT_SUCCESS ('0000') が戻ります。 ルーターがロードされていないと、EHNDT_.E2A_TABLE_NOT_FOUND ('FFFC') が戻ります。 変換中に不適切なデータが検出されると、変換されなかった最初の文字位置に 1 を加えたものが戻りコードになります。

付録 A. APPC 共通戻りコード

この付録では、いくつかの APPC verb に共通の 1 次戻りコード（および、該当する場合は 2 次戻りコード）について説明します。

個々の verb 固有の戻りコードについては、各 verb の項で説明してあります。

AP_ALLOCATION_ERROR

Communications Serverは会話の割振りに失敗しました。 会話状態は RESET にセットされます。 このコードは、**ALLOCATE** または **MC_ALLOCATE** の後で発行された verb から戻されます。 関連する 2 次コードは次のとおりです。

AP_ALLOCATION_FAILURE_NO_RETRY

構成エラーやセッション・プロトコル・エラーなどのような永続条件が原因で、会話の割振りができませんでした。 エラーを判別するには、システム管理者はエラー・ログ・ファイルを調べる必要があります。 エラーが訂正されるまでは、割振りを再試行しないでください。

AP_ALLOCATION_FAILURE_RETRY

リンク障害などの一時的条件が原因で、会話の割振りができませんでした。障害の理由はシステム・エラー・ログに記録されています。 できればタイムアウトによりこの条件が解消されてから、割振りを再試行してください。

AP_SECURITY_NOT_VALID

割振り要求で指定したユーザー 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。

AP_TRANS_PGM_NOT_AVAIL_RETRY

リモート LU が、要求されたパートナー・トランザクション・プログラムを始動できないため、割振りを拒否しました。 タイムアウトなどの一時的条件が原因で、要求したトランザクション・プログラム (TP) が使用できません。 エラーの理由はリモート・ノードに記録されている場合があります。この条件は、オペレーターの介入なしで自然に解消されることがあります。できればタイムアウトによりこの条件が解消されてから、トランザクション・プログラムで会話を再試行してください。

AP_TRANS_PGM_NOT_AVAIL_NO_RTRY

リモート LU が、要求されたパートナー・トランザクション・プログラムを始動できないため、割振りを拒否しました。 永続条件または半永続条件が原因で、要求したトランザクション・プログラム (TP) が使用できません。エラーの理由はリモート・ノードに記録されている場合があります。 この条件は、オペレーターの介入なしで自然に解消されることはありません。エラーが訂正されるまでは、トランザクション・プログラムで会話を再試行しないようにしてください。

AP_TP_NAME_NOT_RECOGNIZED

割振り要求で指定したトランザクション・プログラムは、パートナー LU により認識されません。

AP_PIP_NOT_ALLOWED

要求したトランザクション・プログラムが、プログラム初期設定パラメーター (PIP) を受信できません。 これは、ローカル・トランザクション・プログラムとパートナー・トランザクション・プログラムの不一致を示しています。

AP_PIP_NOT_SPECIFIED_CORRECTLY

要求したトランザクション・プログラムは、プログラム初期設定パラメーター (PIP) を受信できますが、提供された PIP にエラーを発見しました。 これは、ローカル・トランザクション・プログラムとパートナー・トランザクション・プログラムの不一致を示しています。

AP_CONVERSATION_TYPE_MISMATCH

要求したトランザクション・プログラムは、割振り要求に指定した会話のタイプ（基本またはマップ式）をサポートできません。 これは、ローカル・トランザクション・プログラムとパートナー・トランザクション・プログラムの不一致を示しています。

AP_SYNC_LEVEL_NOT_SUPPORTED

要求したトランザクション・プログラムは、割振り要求した **sync_level** (AP_NONE、AP_CONFIRM_SYNC_LEVEL、または AP_SYNCPT) の会話をサポートできません。 これは、ローカル・トランザクション・プログラムとパートナー・トランザクション・プログラムの不一致を示しています。

AP_CANCELLED

会話が取り消された（つまりトランザクション・プログラムが **CANCEL_CONVERSATION** verb を発行した）ために、この verb が戻りました。

AP_CONV_FAILURE_NO_RETRY

セッション・プロトコル・エラーなどの永続条件が原因で、会話が終了しました。 システム管理者は、システム・エラー・ログを調べてエラーの原因を判別する必要があります。 エラー条件が解消されるまでは、会話を再試行しないでください。

AP_CONV_FAILURE_RETRY

一時エラーが原因で会話が終了しました。 トランザクション・プログラムを再始動して、問題が再発するかどうかを確認してください。 再発する場合は、システム管理者はエラー・ログを調べて、エラーの原因を判別する必要があります。

AP_CONVERSATION_TYPE_MIXED

トランザクション・プログラムは、異なる会話タイプの会話 verb を同じ会話で混用しようとしました。 たとえば、トランザクション・プログラムは、**MC_ALLOCATE** verb に続いて **CONFIRM** verb を発行しました。

AP_DEALLOC_ABEND

次のいずれかの理由で会話が割振り解除されました。

- パートナー・トランザクション・プログラムが、**dealloc_type** を AP_ABEND にセットして **MC_DEALLOCATE** verb を発行した。

- パートナー・トランザクション・プログラムが異常終了したため、パートナー LU が **MC_DEALLOCATE** 要求を送信した。

AP_DEALLOC_ABEND_PROG
次のいずれかの理由で会話が割振り解除されました。
- パートナー・トランザクション・プログラムが、**dealloc_type** を AP_ABEND_PROG にセットして **DEALLOCATE** verb を発行した。
- パートナー・トランザクション・プログラムが異常終了したため、パートナー LU が **DEALLOCATE** 要求を送信した。

AP_DEALLOC_ABEND_SVC
パートナー・トランザクション・プログラムが、**dealloc_type** を AP_ABEND_SVC にセットして **DEALLOCATE** verb を発行したため、会話が割振り解除されました。

AP_DEALLOC_ABEND_TIMER
パートナー・トランザクション・プログラムが、**dealloc_type** を AP_ABEND_TIMER にセットして **DEALLOCATE** verb を発行したため、会話が割振り解除されました。

AP_DEALLOC_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・トランザクション・プログラムは、**dealloc_type** を次のいずれかの値にセットして、**DEALLOCATE** verb または **MC_DEALLOCATE** verb を発行しました。
- AP_FLUSH
- AP_SYNC_LEVEL（会話の同期レベルは AP_NONE として指定されている）

AP_DUPLEX_TYPE_MIXED
トランザクション・プログラムは会話verb を発行しようとしましたが、異なった **duplex_type** の会話でした。 たとえば、トランザクション・プログラムは、全二重会話で半二重の**MC_FLUSH** verb (**opext** 中に **AP_FULL_DUPLEX_CONVERSATION** をセットしないで) を発行しました。

AP_ERROR_INDICATION
この戻りコードは全二重会話でのみ使用されます。 パートナー・トランザクション・プログラムが会話を終わらせたために、送信待ち行列操作が失敗しました。 会話状態が送信専用である場合、会話は終了しました。 会話状態が送受信または受信専用である場合、適切な戻りコードが受信待ち行列 verb へ戻されたとき、会話は終了します。 関連する 2 次戻りコードは次のとおりです。

AP_ALLOCATION_ERROR_PENDING
リモート LU は割振り要求を拒否しました。

AP_DEALLOC_ABEND_PROG_PENDING
次のいずれかの理由で会話が割振り解除されました。
- パートナー・トランザクション・プログラムが、**dealloc_type** を AP_ABEND_PROG にセットして **DEALLOCATE** verb を発行した。

- パートナー・トランザクション・プログラムが異常終了したためパートナー LU が **DEALLOCATE** 要求を送信した。

AP_DEALLOC_ABEND_SVC_PENDING
パートナー・トランザクション・プログラムが、**dealloc_type** を AP_ABEND_SVC にセットして **DEALLOCATE** verb を発行したため、会話が割振り解除されました。

AP_DEALLOC_ABEND_TIMER_PENDING
パートナー・トランザクション・プログラムが、**dealloc_type** を AP_ABEND_TIMER にセットして **DEALLOCATE** verb を発行したため、会話が割振り解除されました。

AP_UNKNOWN_ERROR_TYPE_PENDING
会話はパートナー・トランザクション・プログラムによって割振り解除されましたが、ローカル LU はその理由を認識していません。

AP_OPERATION_INCOMPLETE
トランザクション・プログラムは処理を開始する非ブロッキング verbを発行しましたが、完了しませんでした。 verb の処理が完了したとき、最後の戻りコードがセットされ、スタブはトランザクション・プログラムに通知します。

AP_PROG_ERROR_NO_TRUNC
パートナー・トランザクション・プログラムが、会話が SEND 状態にあるときに次のいずれかの verb を発行しました。

- **err_type** を AP_PROG にセットした **SEND_ERROR**
- **MC_SEND_ERROR**

データは切り捨てられていません。

AP_PROG_ERROR_PURGING
パートナー・トランザクション・プログラムが、RECEIVE、PENDING_POST、CONFIRM、CONFIRM_SEND、または CONFIRM_DEALLOCATE 状態のときに、次のいずれかの verb を発行しました。

- **err_type** を AP_PROG にセットした **SEND_ERROR**
- **MC_SEND_ERROR**

送信済みであるが、まだパートナー・トランザクション・プログラムによって受信されていないデータが除去されました。

AP_PROG_ERROR_TRUNC
SEND 状態のときに、パートナー・トランザクション・プログラムが、論理レコード全体を送りきってしまう前に、**err_type** を AP_PROG にセットして **SEND_ERROR** verb を発行しました。 ローカル・トランザクション・プログラムは、**RECEIVE** verb を介して切り捨ての生じた論理レコードを受信している可能性があります。

AP_SVC_ERROR_NO_TRUNC
SEND 状態にあるときに、パートナー・トランザクション・プログラム（ま

たはパートナー LU）が、**err_type** を AP_SVC にセットして **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。

AP_SVC_ERROR_PURGING

RECEIVE、PENDING_POST、CONFIRM、CONFIRM_SEND、または CONFIRM_DEALLOCATE 状態のときに、パートナー・トランザクション・プログラム（またはパートナー LU）が、**err_type** を AP_SVC にセットして **SEND_ERROR** verb を発行しました。 パートナー・トランザクション・プログラムに送られたデータは除去されている場合があります。

AP_SVC_ERROR_TRUNC

SEND 状態のときに、パートナー・トランザクション・プログラム（またはパートナー LU) が、論理レコード全体を送りきってしまう前に、**SEND_ERROR** verb を発行しました。 ローカル・トランザクション・プログラムは、切り捨ての生じた論理レコードをすでに受け取っている場合があります。

AP_TP_BUSY

Communications Server が同じ会話で他の verb を処理しているときに、ローカル・トランザクション・プログラムが Communications Server にブロッキング verb を発行しました。

AP_UNEXPECTED_SYSTEM_ERROR

Communications Serverは予期しないシステム・エラーを検知し、verb を完了できません。 通常、この種のエラーはシステム資源（メモリーなど）の不足が原因で生じるものであり、多くの場合一時的な現象です。詳細についてはシステム・ログを調べてください。

AP_SEC_REQUESTED_NOT_SUPPORTED

パートナー LU がパスワード置換をサポートしていないので、ローカル LU は会話を割り振ることができません。 ALLOCATE または SEND_CONVERSATION 上で要求されたセキュリティー・タイプが AP_PGM_STRONG でした。これはパスワード置換のサポートを必要とします。

付録 B. APPC 会話状態の変化

次の表は、各 APPC verb を発行できる会話の状態と、その verb の完了時に起こる状態変化を示しています。場合によっては、状態変化は verb に返される **primary_rc** パラメーターによって異なることもあります。その場合は、適用される **primary_rc** の値を戻りコードの欄に示してあります。

戻りコードが示されていない場合は、状態変化はどの戻りコードのときも同じです（表の後の注 2 および注 3 に述べる場合を除きます）。

起こりうる会話状態は、表の一番上の欄に示されています。 各 verb について、その verb を各状態で発行したときの結果が、各欄の見出しの下に次のように表示されます。

- **X** は、この状態では該当の verb を発行できないことを示します。
- **S**、**SP**、**R**、**C**、**CS**、**CD**、または **P** は、verb の完了後の会話の状態を示します。**R**eset（リセット）、**S**end（送信）、**S**end **P**ending（送信保留）、**R**eceive（受信）、**C**onfirm（確認）、**C**onfirm **S**end（送信確認）、**C**onfirm **D**eallocate（割振り解除確認）、または **P**ending Post（通知保留）。
- **/** は、この欄の状態が適用されないことを示します。 これが該当するのは、**[MC_]ALLOCATE** verb と**RECEIVE_ALLOCATE** verb です。これらの verb は、常に、リセット状態の場合のように新しい会話を始動し、したがってこれらの verb を発行した会話には影響はありません。
- ブランクは、該当の戻りコードがこの状態では発生しないものであることを示します。

全二重会話の状態変移については、322ページの表 21を参照してください。

表 20. APPC 半二重会話の状態変移

verb 戻りコード	リセット (T)	送信 (S)	送信保留 (SP)	Receive (R)	Confirm (C)	送信確認 (CS)	割振り解除確認 (CD)	通知保留 (PS)
[MC_]ALLOCATE								
AP_OK	S T	/	/	/	/	/	/	/
(その他)								
CANCEL_CONVERSATION	X	T	T	T	T	T	T	T
[MC_]CONFIRM								
AP_OK	X	S R	S R	X	X	X	X	X
AP_ERROR								
[MC_]CONFIRMED	X	X	X	X	R	S	T	X

表 20. APPC 半二重会話の状態変移 (続き)

verb 戻りコード	リセット (T)	送信 (S)	送信保留 (SP)	Receive (R)	Confirm (C)	送信確認 (CS)	割振り解除確認 (CD)	通知保留 (PS)
[MC_]DEALLOCATE（異常終了）								
[MC_]DEALLOCATE（その他）	X	T	T					
				T	T	T	T	T
	X	R	R					
AP_ERROR (その他)		T	T	X	X	X	X	X
[MC_]FLUSH	X	S	S	X	X	X	X	X
[MC_]GET_ATTRIBUTES	X	S	SP	R	C	CS	CD	P
GET_STATE	X	S	SP	R	C	CS	CD	P
GET_TYPE	X	S	SP	R	C	CS	CD	P
[MC_]PREPARE_TO_RECEIVE	X	R	R	X	X	X	X	X
RECEIVE_ALLOCATE								
	R							
AP_OK	T	/	/	/	/	/	/	/
(その他)								
[MC_]RECEIVE_AND_POST	X	P	P	P	X	X	X	X
（注 4）								
[MC_]RECEIVE_AND_WAIT	X	注5	注5	注5	X	X	X	X
[MC_]RECEIVE_IMMEDIATE	X	X	X	注5	X	X	X	X
[MC_]REQUEST_TO_SEND	X	X	X	R	C	X	X	P
[MC_]SEND_DATA								
		S						
AP_OK	X	S	S	X	X	X	X	X
AP_ERROR								
[MC_]SEND_ERROR								
		S						
AP_OK	X	R	S	S	S	S	S	S
AP_ERROR								
[MC_]TEST_RTS	X	S	S	R	C	C	C	P

注:

1. 表の「戻りコード」欄で、AP_ERROR という省略語は次の戻りコードを表しています。

   AP_PROG_ERROR_TRUNC

   AP_PROG_ERROR_NO_TRUNC

   AP_PROG_ERROR_PURGING

   AP_SVC_ERROR_TRUNC

   AP_SVC_ERROR_NO_TRUNC

   AP_SVC_ERROR_PURGING.

2. 次のいずれかの戻りコードを受信した場合は、会話は常にリセット状態になります。

   AP_ALLOCATION_ERROR

   AP_COMM_SUBSYSTEM_ABENDED

   AP_COMM_SUBSYSTEM_NOT_LOADED

   AP_CONV_FAILURE_RETRY

   AP_CONV_FAILURE_NO_RETRY

   AP_DEALLOC_ABEND

   AP_DEALLOC_ABEND_PROG

   AP_DEALLOC_ABEND_SVC

   AP_DEALLOC_ABEND_TIMER

   AP_DEALLOC_NORMAL

3. 次のような異常戻りコードを受信した場合は、状態変化は生じません。 会話は、常に、verb を発行したときの状態のままとなります。

   AP_CONVERSATION_TYPE_MIXED

   AP_PARAMETER_CHECK

   AP_STATE_CHECK

   AP_TP_BUSY

   AP_UNEXPECTED_SYSTEM_ERROR

   AP_UNSUCCESSFUL

4. **[MC_]RECEIVE_AND_POST** が発行され、初期 **primary_rc** として AP_OK を受信した後は、会話は通知保留状態に変化します。 verb が完了したことを示すコールバック・ルーチンが呼び出された後の新しい会話状態は、注 5 に示すように、**primary_rc** および **what_rcvd** パラメーターによって異なります。

5. **RECEIVE** verb の後の状態変化は、**primary_rc** および **what_rcvd** の両方のパラメーターによって異なります。

   primary_rc パラメーターが、AP_PROG_ERROR*、AP_SVC_ERROR*、または（[MC_]RECEIVE_IMMEDIATE の場合のみ）AP_UNSUCCESSFUL である場合は、新しい状態は RECEIVE になります。

   primary_rc パラメーターが AP_DEALLOC* である場合は、新しい状態は RESET になります。

   primary_rc パラメーターが AP_OK である場合は、新しい状態は **what_rcvd** パラメーターの値によって異なります。

   受信状態

   AP_DATA、AP_DATA_COMPLETE、AP_DATA_INCOMPLETE

   送信状態

   AP_SEND

   送信保留状態

   AP_DATA_SEND、AP_DATA_COMPLETE_SEND

確認状態

AP_CONFIRM_WHAT_RECEIVED、AP_DATA_CONFIRM、
AP_DATA_COMPLETE_CONFIRM

送信確認状態

AP_CONFIRM_SEND、AP_DATA_CONFIRM_SEND、
AP_DATA_COMPLETE_CONFIRM_SEND

割振り解除確認状態

AP_CONFIRM_DEALLOCATE、AP_DATA_CONFIRM_DEALLOCATE、
AP_DATA_COMPLETE_CONFIRM_DEALL

半二重会話の状態変移については、319ページの表 20を参照してください。

表 21. APPC 全二重会話の状態変移

verb 戻りコード	リセット (T)	送信と受信 (SR)	送信のみ (S)	受信のみ (R)
[MC_]ALLOCATE				
	SR			
AP_OK	T	/	/	/
(その他)				
CANCEL_CONVERSATION	X	T	T	T
[MC_]DEALLOCATE（異常終了）	X	T	T	T
[MC_]DEALLOCATE（フラッシュ）	X	R	T	X
[MC_]FLUSH	X	SR	S	X
[MC_]GET_ATTRIBUTES	X	SR	S	R
GET_STATE	X	SR	S	R
GET_TYPE	X	SR	S	R
RECEIVE_ALLOCATE				
AP_OK	SR	/	/	/
(その他)	T			
[MC_]RECEIVE_AND				
WAIT	X	SR	X	R
AP_OK	X	SR	X	R
AP_ERROR	X	S	X	T
AP_DEALLOC_NORMAL				
RECEIVE_EXPEDITED_DATA	X	SR	S	R
[MC_]RECEIVE_IMMEDIATE				
AP_OK	X	SR	X	R
AP_ERROR	X	SR	X	R
AP_DEALLOC_NORMAL	X	S	X	T
[MC_]SEND_DATA				
AP_OK	X	SR	S	X
AP_ERROR_INDICATION	X	SR	T	X

表 21. APPC 全二重会話の状態変移 (続き)

verb 戻りコード	リセット (T)	送信と受信 (SR)	送信のみ (S)	受信のみ (R)
[MC_]SEND_ERROR				
AP_OK	X	SR	S	X
AP_ERROR_INDICATION	X	SR	T	X

注:

1. 表の「戻りコード」欄で、AP_ERROR という省略語は次の戻りコードを表しています。

   AP_PROG_ERROR_TRUNC

   AP_PROG_ERROR_NO_TRUNC

   AP_SVC_ERROR_TRUNC

   AP_SVC_ERROR_NO_TRUNC

2. 次のいずれかの戻りコードを受信した場合は、会話は常にリセット状態になります。

   AP_ALLOCATION_ERROR

   AP_COMM_SUBSYSTEM_ABENDED

   AP_COMM_SUBSYSTEM_NOT_LOADED

   AP_CONV_FAILURE_RETRY

   AP_CONV_FAILURE_NO_RETRY

   AP_DEALLOC_ABEND

   AP_DEALLOC_ABEND_PROG

   AP_DEALLOC_ABEND_SVC

   AP_DEALLOC_ABEND_TIMER

3. 次のような異常戻りコードを受信した場合は、状態変化は生じません。 会話は、常に、verb を発行したときの状態のままとなります。

   AP_CONVERSATION_TYPE_MIXED

   AP_PARAMETER_CHECK

   AP_STATE_CHECK

   AP_TP_BUSY

   AP_UNEXPECTED_SYSTEM_ERROR

   AP_UNSUCCESSFUL

用語集

A

アベンド (abend). (1) タスクの異常終了。タスクが実行されているとき、回復機能によっては解決できないエラー状態が起こったために、完了前にタスクが終了すること。(2) *異常終了 (abnormal termination)*の同義語。

異常終了 (abnormal termination). (1) 予定どおりの正常な手順で終了する前に処理が停止されること。 (T)(2) ジョブの不成功終了の原因となるようなシステムの障害または操作員のアクション。(3) *アベンド (abend)*と同義。

受入れ、受け入れる (accept). (1) VTAM アプリケーション・プログラムにおいて、システム・サービス制御点 (SSCP) からの CINIT 要求に応えて、論理装置 (LU) とのセッションを確立すること。セッション開始要求は、端末ユーザーがログオンしたとき、VTAM アプリケーション・プログラムがマクロ命令を発行したとき、またはVTAMオペレーターがコマンドを発行したときに始まる。*獲得 (する)(acquire)* も参照。(2) 配布されたコードおよび MVS タイプ・プログラムを配布ライブラリーに移すための SMP プロセス。

ACCESS. 簡易ネットワーク管理プロトコル (SNMP) の管理情報ベース (MIB) モジュール内の文節。管理ノードが１つのオブジェクトについて提供する最低レベルのサポートを定義する。

アクセス方式 (access method). (1) ネットワークを介した情報の流れを制御するためにソフトウェアに組み込まれている技法。(2) 主記憶装置と入出力装置の間でデータを移動する技法。

ACF/VTAM. VTAM 拡張通信機能アクセス方式 (Advanced Communications Function for the Virtual Telecommunications Access Method)。 *VTAM (仮想記憶通信アクセス方式)* の同義語。

肯定応答 (acknowledgment). (1) 送信側に肯定応答として受信側が確認文字を伝送すること。 (T) (2) 送信した項目が受信されたことを示す指示。

獲得 (する) (acquire). (1) VTAM において、前に他の定義域内でアクセス方式により制御されていた資源を引き継ぐこと、または、他の定義域内で制御されていたが現在は解放されている資源を再び制御すること。*解放 (する)(release)* と対比。*資源引継ぎ (resource takeover)* も参照。(2) VTAM アプリケーション・プログラムにおいて、他の論理装置 (LU) とのセッションを開始し確立すること。獲得プロセスは、アプリケーション・プログラムがマクロ命令を発行したときに始まる。*受入れ、受け入れる (accept)* も参照。

アクション（action）. (1) 管理オブジェクトに対する操作の１つ。この操作の意味は、管理オブジェクトのクラス定義の一環として定義される。(2) AIX オペレーティング・システムにおいて、アプリケーションが行う定義済みのタスク。アクションは、オブジェクトの特性を修正したり、オブジェクトを何らかの方法で操作する。

活動化 (する) (activate). 資源の機能が実行されるように資源を準備すること。*非活動化 (する)(deactivate)* と対比。

活動、活動状態 (active). (1) 操作可能。(2) ノードまたは装置が、他のノードまたは装置に接続されているか、または接続できる状態にあること。(3) 資源が活動化され、作動可能になっているときの状態。

ACTLU. 論理装置活動化 (Activate logical unit)。SNA において、論理装置でのセッションを開始するために使用するコマンド。

ACTPU. 物理装置活動化。 SNA において、物理装置でのセッションを開始するために使用するコマンド。

アダプター. (1) SDLC、LAN、非同期 DFT、または他の通信接続機構に接続するために (おそらくモデムを介して)、パーソナル・コンピューターに導入しなければならないハードウェア構成要素。(2) 装置をコンピューターまたは他の装置に電気的または物理的に接続する部品。

適応ペーシング (adaptive pacing). *最適セッション・レベル・ペーシング (adaptive session-levelpacing)* の同義語。

最適セッション・レベル・ペーシング (adaptive session-level pacing). セッション・レベル・ペーシングの形式の１つ。この形式では、セッションの途上で、各種構成要素がそれぞれサイズの異なるペーシング・ウィンドウを交換し合う。 その結果、ネットワーク内の伝送を、セッション単位で動的にバッファーの可用度と需要の変化に適応させることができる。 セッション・パスに沿った個々の段階で、中間ノードおよび端点ノードにおけるローカルの輻輳（ふくそう）に応じて、セッション・レベ

ル・ペーシングが行われる。 *適応ペーシング (adaptive pacing)* および*最適セッション・ペーシング (adaptive session pacing)* と同義。 *固定セッション・レベル・ペーシング(fixed session-level pacing)* と対比。

最適セッション・ペーシング (adaptive session pacing). *最適セッション・レベル・ペーシング (adaptive session-levelpacing)* の同義語。

アドレス (address). データ通信では、ネットワークに接続された各装置、ワークステーション、またはユーザーに割り当てられた固有のコード。

管理定義域 (Administrative Domain). 単一の管理権限により管理するホストとルーター、およびそれを相互に接続するネットワークの集合。

拡張対等通信ネットワーク機能 (Advanced Peer-to-Peer Networking (APPN)). SNA に対する拡張機能の 1 つで、次のような特色を備えている。(a) 分散ネットワーク制御の能力が強化された結果、大きい階層依存関係がなくなるので、単一個所での障害を分離して影響をその個所のみに限定できる。(b) 接続、再構成、および最適経路選択を容易にするための、ネットワーク・トポロジー情報の動的交換。(c) ネットワーク資源の動的定義。(d) 自動化された資源登録とディレクトリー・ルックアップ。APPN は、エンドユーザー・サービス用の LU 6.2 対等オリエンテーションをネットワーク制御にまで拡張し、LU 2、LU 3、LU 6.2 など複数の LU タイプをサポートする。

拡張対等通信ネットワーキング (APPN) エンド・ノード (Advanced Peer-to-Peer Networking (APPN) end node). 広範なエンドユーザー・サービスを提供し、自身のローカル制御点 (CP) と隣接ネットワーク・ノード内の CP との間のセッションをサポートする。 このエンド・ノードは、自身の資源を隣接 CP (エンド・ノードのネットワーク・ノード・サーバー) に動的に登録し、ディレクトリー検索要求を送受信し、そして管理サービスを取得する。 APPN エンド・ノードは他のエンド・ノードに接続されることもある。

拡張対等通信ネットワーク機能 (APPN) ネットワーク (Advanced Peer-to-Peer Networking (APPN) network). 相互に接続されたいくつかのネットワーク・ノードとそれぞれのクライアント・エンド・ノードの集合。

拡張対等通信ネットワーク機能 (APPN) ネットワーク・ノード (Advanced Peer-to-Peer Networking (APPN) network node). 広範なエンドユーザー・サービスを提供し、次の機能を備えているノード。

- 分散ディレクトリー・サービス。たとえば、中央ディレクトリー・サーバーへの定義域資源の登録など。
- 他の APPN ネットワーク・ノードとのトポロジー・データベースの交換。これによって、ネットワーク全域に散在するネットワーク・ノードが、要求されたサービス・クラスに応じて、LU-LU セッション用の最適な経路を選択できるようになる。
- ローカル LU およびクライアント・エンド・ノード用のセッション・サービス。
- APPN ネットワーク内での中間経路指定サービス

拡張対等通信ネットワーキング (APPN) ノード (Advanced Peer-to-Peer Networking (APPN) node). APPN ネットワーク・ノード、すなわち APPN エンド・ノード。

拡張プログラム間通信 (advanced program-to-program communication (APPC)). (1) (2) 相互接続されたシステムがプログラミング・タスクを共用できるようにする、SNA の LU 6.2 論理装置プロトコルの実施形態。LU 6.2 アーキテクチャー、および各種プロダクトでの同アーキテクチャーの各種の実装を特徴づけている一般機能。(3) LU 6.2 アーキテクチャーとそれを実装するプロダクトを包括的に表すこともあり、また、APPC アプリケーション・プログラミング・インターフェースなどのように、LU 6.2 プロダクトの特定の機能を限定的に表すこともある。

AID. アテンション ID (Attention identifier)。

割振り、割り振る (allocate). (1) セッションを会話に割り当てて、そのセッションをその会話に使用させるための LU 6.2 アプリケーション・プログラミング・インターフェース (API) verb。(2) *割振り解除 (deallocate)* と対比。

全点アドレス可能 (all points addressable (APA)). コンピューター・グラフィックスにおいて、表示画面上の個々の画素 (ペル) をアドレス指定し、各画素を表示するか表示しないかを決定できる能力。

米国規格協会 (American National Standards Institute (ANSI)）. 製造業者、消費者、および一般利害関係者の団体により構成された組織であって、認定された企業が米国において自主的な業界標準を作成および維持するための手順を確立する組織。 (A)

AND 演算 (AND operation). *論理積 (conjunction)* の同義語。

ANSI. 米国規格協会 (American National Standards Institute)。

AP. 代替プリンター (Alternate printer)

APA. 全点アドレス可能 (All points addressable)。

API. アプリケーション・プログラミング・インターフェース (application programming interface)。

APPC. 拡張プログラム間通信 (Advanced program-to-program communication)。

アプリケーション (application). コンピューターでの特定タイプのユーザー指向作業を実行するために使用されるソフトウェア構成要素の集合。

アプリケーション・プログラム(application program). (1) 特定のユーザーの作業に使用するために、そのユーザーが作成するか、またはそのユーザー用として作成されたプログラム。たとえば、在庫管理や給与計算を行うプログラムなど。(2) ネットワーク内のステーションと接続または通信するために使用されるプログラムであって、ユーザーがアプリケーション指向活動を実行できるようにするもの。

アプリケーション・プログラミング・インターフェース (application programming interface (API)). (1) (2) IBM システム制御プログラムまたは IBM ライセンス・プログラムとプログラム・ユーザーとの間の、定義されたプログラミング言語インターフェース。基礎となっているオペレーティング・システムまたはサービス・プログラムが提供する特定の機能およびサービスを利用するために、アプリケーション・プログラムの中でコーディングできるプログラム言語の構成およびステートメントのセット。(3) VTAM において、アプリケーション・プログラムが制御ブロックを参照し、また VTAM に対してそれ自体を識別させるように制御ブロックの中で使用される言語構造。

適用 (Apply). ウィンドウ内で選択した選択項目を、そのウィンドウをクローズせずに実行する押しボタン。

APPN. 拡張対等通信ネットワーキング (AdvancedPeer-to-Peer Networking)。

APPN エンド・ノード (APPN end node). *拡張対等通信ネットワーキング (APPN) エンド・ノード (Advanced Peer-to-Peer Networking (APPN) end node)*を参照。

引数 (argument). 呼出しプログラムと呼び出されるプログラムとの間で渡されるパラメーター。

ASCII (情報交換用米国標準コード) (American National Standard Code for Information Interchange). 7 ビットのコード化文字 (パリティー検査を含めると 8 ビット) によって構成されるコード化文字セットを使用した標準コード。このコードはデータ処理システム、データ通信システム、および関連した装置の間の情報交換に使用される。 ASCII セットは、制御文字とグラフィック文字から成る。 (A)

ASYNC. 非同期 (asynchronous)。

非同期 (asynchronous (ASYNC)). (1) 共通タイミング信号のように、特定のイベントの発生に依存しない複数のプロセスの状態を表す。 (T)(2) 規則的な時間関係が存在しないこと。プログラム命令の実行に関して予測不能または予期不能であること。

非同期操作 (asynchronous operation). ソフトウェアまたはハードウェアの同時操作。ソフトウェアでは、たとえばセッション確立要求またはデータ転送などのように、アプリケーション・プログラムの実行を続けたままで実施できる操作。操作が完了すると、アクセス方式がアプリケーション・プログラムに通知する。*同期操作 (synchronous operation)* と対比。

非同期要求 (asynchronous request). VTAM において、非同期操作を求める要求。*同期要求 (synchronous request)* と対比。

タスク生成する、接続する、付加する (attach). (1) プログラミングにおいて、メインライン・コードの実行とは非同期的に実行できるタスクを作成すること。(2) 装置をリング・ネットワークへ論理的に接続すること。

アテンション識別子 (attention identifier (AID)). データ・ストリーム内の文字であって、ユーザーが、Enter キーのようなキーを押して、システムによるアクションを要求したことを示す文字。

属性 (attribute). 論理上あるオブジェクトの一部となっており、そのオブジェクトの特性を示す変数データ。たとえば、製造番号は装置オブジェクトの属性の 1 つである。

権限、許可 (authorization). (1) コンピューター・セキュリティーにおいて、ユーザーがコンピューター・システムと通信したり使用したりするために、与えられる権利。 (T) (2) アクセス権。(3) オブジェクト、資源、または機能について完全または制限されたアクセスをユーザーに与えるプロセス。

自動活動化 (automatic activation). VTAM において、あるサブエリア・ノードの名前を指定した活動化コマンドに関連するチャネル装置名または RNAME 仕様の結果として、隣接サブエリア・ノードの中のリンクおよびリンク・ステーションが活動化されること。*直接活動化 (direct activation)* も参照。

自動タスク (autotask). (1) 端末またはログオンしているユーザーを必要としない不在時 NetView 操作員ステーション・タスク。自動タスクは VTAM から独立して実行でき、一般に自動コンソール操作用に使用される。(2) *ログオン操作員 (logged-on operator)* と対比。

使用可能 (available). VTAM において、活動化され、接続され、使用可能にされていて、しかもセッション限度に達していない論理装置を表す。

B

バックグラウンド・プロセス (background process). (1) 操作員の介入を必要とせず、ワークステーションで他の作業を行いながらコンピューターにより実行できるプロセス。(2) AIX オペレーティング・システムにおけるプログラム実行モードの 1 つ。このモードでは、シェルは、プログラムの完了を待たずにユーザーに新たなコマンドの入力を求める。(3) *フォアグラウンド・プロセス (foreground process)* と対比。

バックアウト (backout). IMS/VS において、異常終了したアプリケーション・プログラムが行ったすべてのデータベース更新を削除するプロセス。

ベース・セット (base set). (1) 特定のアーキテクチャーを実装するすべての製品でサポートされている（verb、パラメーター、戻りコード、受信内容指示を含む）機能のセット。(2) *オプション・セット (option set)* と対比。

基本会話（basic conversation）. 割振りトランザクション・プログラムが指定する LU 6.2 会話タイプ。 基本会話を使用するトランザクション・プログラムは、さらに多種多様な LU 6.2 機能を使用できるが、そのためには、独自にエラー回復を行い、会話で使用するデータ・ストリームの詳細を管理する必要がある。 *マップ式対話（mapped conversation）* と対比。

基本情報単位 (basic information unit (BIU)). SNA において、複数のハーフセッションの間で渡されるデータおよび制御情報の単位。 要求応答ヘッダー (RH) とそれに続く要求応答単位 (RU) から成る。

バッチ (batch). (1) 処理すべきデータの蓄積。(2) 処理または伝送のために 1 グループにまとめられたレコードまたはデータ処理ジョブ。(3) ユーザーの処置をほとんど、あるいはまったく含まない活動に関連する用語。*対話式 (interactive)* と対比。

バッチ・ファイル (batch file). (1) IBM DOS において、順次に処理されるコマンドを含んでいるファイル。DOS バッチ・ファイルが実行可能であるためには、.BAT エクステンションが付いている必要がある。(2) 順次に処理する一連のコマンドが入ったファイル。

開始ブラケット (begin bracket). SNA において、ブラケットの最初の連鎖の中にある最初の要求の要求ヘッダー (RH) の中にあるブラケット開始標識の値 (2 進数の 1)。*終了ブラケット (end bracket)* と対比。*ブラケット (bracket)* も参照。

動作（behavior）. (1) 理論上は、管理オブジェクトがとれる状態を記述した断定の集合。断定には、事前条件、事後条件、または不変関係がある。現実には、動作は、属性、操作、および通知の意味を表わすくだけた表現であることが多い。(2) 管理オブジェクト、名前バインド、属性、通知、および命令が、相互に対話し、またそれぞれに対応する実際の資源と対話する方法。

送信権要求 (bid). 送信勧誘または選択において競合が生じたときに、コンピューターまたはステーションの 1 つが、データ伝送のために回線の制御権を獲得しようとすること。

送信権要求者 (bidder). *送信権要求者セッション (bidder session)* を参照。

送信権要求者セッション (bidder session). セッション活動化の時点で定義され、ブラケット開始の許可を相手のハーフセッションに要求し受け取らなければならないハーフセッション。*ファースト・スピーカー・セッション (first-speaker session)* と対比。*競合敗者セッション (contention-loser session)* の同義語。

2 進、2 進法、2 値、2 進数、バイナリー (binary). 基数 2 の数体系に関するもの。 2 進数字は 0 と 1 である。実行可能ファイルは、通常、文字ストリング形式ではなく 2 進形式である。テキスト・ファイルは文字ストリング形式である。

BIND. SNA において、2 つの論理装置 (LU) の間でセッションを活動化するための要求。 *セッション活動化要求 (session activation request)* も参照。 *UNBIND* と対比。

bis. ドイツ (連邦共和国) の予備的標準。

ビット (bit). 2 進数体系で使用される数字 0 または 1。 (T)

BIU. 基本情報単位 (basic information unit)。

BIU セグメント (BIU segment). SNA において、パス情報単位 (PIU) の中に含まれる基本情報単位 (BIU) 部分。(a) 要求応答ヘッダー (RH) の後に要求応答単位 (RU) の全体または一部を続けたものか、または、(b) RU の一部から成っている。 セグメント *(segment)* と同義。

ブロック (block). 1 単位として記録または転送されるデータ要素のストリング。 要素は、文字、語、または物理レコードのいずれでもよい。 (T)

ブロッキング・モード (blocking mode). (1) インターフェースを介してサービスを要求する方法の 1 つ。要求がただちに完了できない場合、要求が完了するまで要求元のプロセスは延期される。(2) *非ブロッキング・モード* と対比。

ブラケット. SNA において、2 人のセッション・パートナーとの間で交換される要求単位とその応答から成る 1 つまたは複数の連鎖であって、セッション・パートナーの間のトランザクションを表すもの。 1 つのブラケットが完了してからでないと、次のブラケットは開始できない。 ブラケットの例としては、データベースへの照会と応答、更新トランザクション、ワークステーションへのリモート・ジョブ入力の出力文字ストリングなどがある。

ブラケット・プロトコル (bracket protocol). SNA において、ブラケットを使用して 2 つのセッション・パートナー間の交換を行うデータ・フロー制御プロトコル。セッション活動化の時点で一方のパートナーがファースト・スピーカーとして指定され、もう一方のパートナーが送信権要求者として指定される。 ブラケット・プロトコルは、ブラケット開始および終了規則に従って働く。

ブリッジ (bridge). (1) 2 つのローカル・エリア・ネットワーク (LAN) が同じ論理リンク制御プロトコルを使用しているが、異なった媒体アクセス制御プロトコルを使用する可能性がある場合に、それらローカル・エリア・ネットワークを相互接続する機能単位。 (T)(2) 複数の LAN を（ローカルまたはリモートにより）相互に接続する機能単位。これらの LAN は、同じ論理リンク制御プロトコルを使用していなければならないが、媒体アクセス制御プロトコルは違っていてもよい。 ブリッジは、媒体アクセス制御 (MAC) アドレスに基づいて、フレームを他のブリッジに転送する。(3) ローカル・ループ、チャネル、またはリングの接続で、回線を突き合わせて正確なデータ転送を実現するために使用される装置と技法。(4)ゲートウェイ *(gateway)* およびルーター *(router)* と対比

ブラウズ (browse). (1) ファイル内のレコードを見ること。(2) NetView グラフィック監視機能において、NetView プログラムからの状況変更を受け取れないビューをオープンすること。モニター *(monitor)* と対比。

バッファー（buffer）. (1) 1 つの装置から他の装置にデータを転送するときに、データ・フロー速度の差異、またはイベント発生時間の差異を補償するために使用されるルーチンまたは記憶域。 (A) (2) 入力データもしくは出力データを一時的に保留するための記憶域の一部分。

バス (bus). (1) 2 つのエンド・ポイント間に位置するいくつかの装置間でデータを転送するための機能。 1 時点で 1 つの装置だけが伝送を行うことができる。 (T)(2)プロセッサーが直列的に相互接続されているコンピューター構成。

バス・マスター (bus master). 自身とその従属装置との間のデータ転送を制御する装置またはサブシステム。

バイト（byte）. (1)ストリングで、1単位として扱われ、1 つの文字を表すもの。 (T) (2) 1 単位として扱われ、通常、1 つのコンピューターの 1 ワードよりも短い 2 進文字。(A)(3) 1 個の EBCDIC 文字を表す、 8 個の隣接した2 進数字のグループ。

C

呼出し、呼び出す（call）. (1) コンピューター・プログラム、ルーチン、またはサブルーチンを実効化するアクション。この実効化は、通常、入口条件を指定して入口点にジャンプすることによって行われる。 (I) (A) (2) データ通信において、交換回線上の 2 つのステーションの間で接続を行うために必要なアクション。(3) 通信において、2 人のユーザーの間の会話。(4)プロシージャー、プログラム、ルーチン、またはサブルーチンへ制御を移すこと。(5) ユーザーと連絡をとるための試み。その試みの成功または不成功を問わない。

コール・バック (call back). AIX オペレーティング・システムにおいて、指定した条件が満たされたときに呼び出されるプロシージャー。

呼出し、呼出し側（calling）. (1) データ・ステーション間の接続を確立するために選択信号を伝送するプロセ

ス。 (I) (A) (2) X.25 通信において、呼出しを行うロケーションまたはユーザーを表す。

取消 (Cancel). ウィンドウに何の変更も加えないで、そのウィンドウを削除する押しボタン。

大文字・小文字の区別 (case-sensitive). 大文字と小文字を区別する能力のことを指す。

CD. コンパクト・ディスク (Compact disc)。

連鎖 (chain). (1) LU 6.2 によって処理される、論理的にリンクされたユーザー・データ・レコードのグループ。 (2) 開始連鎖および終了連鎖によって区切られた要求単位のグループ。 応答は常に単一単位連鎖である。*RU 連鎖 (RU chain)* を参照。

チャネル（channel）. (1) シグナルを送信するパス。データ・チャネル、出力チャネルなどがある。 (A) (2) データ通信における、片方向伝送の手段。(3) プロセッサーによって制御される機能単位であって、主記憶装置とローカル周辺装置との間でデータ転送を処理するもの。

チャネル接続 (channel-attached). (1) 装置を直接に入出力チャネルによってホスト・プロセッサーに接続することを指す。(2) 通信回線ではなくケーブルによって装置を制御装置に接続することを指す。(3) *リンク接続(link-attached)* と対比。(4) *ローカル (local)* と同義。

文字セット (character set). キーボードまたは出力装置のために定義された文字の、有限のグループ。

子 (child). セキュリティーの対象になる資源 (ファイルまたはライブラリー) であって、親資源のユーザー・リストを使用するもの。子資源は、親資源を 1 つだけもつことができる。 子は親プロセスにより開始されるプロセスで、親プロセスの資源を共用する。*親 (parent)* と対比。

子プロセス (child process). AIX および OS/2 オペレーティング・システムにおいて、親プロセスにより開始され、親プロセスの資源を共用するプロセス。 フォーク *(fork)* も参照。

CICS. 顧客情報管理システム (Customer Information Control System)。

回線（circuit）. (1) 電流を流すことができる 1 つまたは複数の導体。 *物理回線 (physical circuit)* および*仮想回線 (virtual circuit)* を参照。(2) 論理装置。

C 言語 (C language). 最小限の変更を加えるだけで複数の異なるタイプのコンピューターで実行できるようにするために、コンパクトで効率的なコードを使用してソフトウェア・アプリケーションを開発するための言語。

クラス（class）. (1) オブジェクト指向の設計またはプログラミングにおいて、共通の定義を共用し、したがって共通の特性、操作、および性質を共用するオブジェクトのグループ。このグループのメンバーをクラスのインスタンスと呼ぶ。(2) AIX オペレーティング・システムにおいて、装置の入出力特性を表す。システム装置は、ブロック装置またはキャラクター型装置のいずれかに分類される。

クリーンアップ (cleanup). SNA 製品において、システム・サービス制御点 (SSCP) が論理装置 (LU) に送るネットワーク・サービス要求。この要求の結果として、他の LU またはその SSCP の関与を必要とせずに、目的のLUとの特定の LU-LU セッションがただちに終了する。

送信可 (clear to send (CTS)). データ通信において、データ回線終端装置 (DCE) が、一般に送信要求 (RTS) に対する応答として、データの受入れが可能になった時点で立ち上げるシグナル。

クライアント (client). (1) サーバーから共用サービスを受け取る機能単位。 (T) (2) ユーザー。(3) AIX 分散ファイル・システム環境において、サーバーに従属して、サーバーにプログラムを提供したりプログラムへのアクセスを提供するシステム。(4) *リクエスター (requester)* と同義。

クライアント/サーバー (client/server). 通信において、１つのサイトに存在するプログラムが他のサイトに存在するプログラムに要求を送って応答を待つような、分散データ処理の対話モデル。 要求を出す側のプログラムをクライアント、応答する側のプログラムをサーバーという。

クローズ (Close). １つのウィンドウおよびそれに関連したすべてのウィンドウをワークプレースから削除する選択項目。たとえば、ユーザーがウィンドウ内で作業を行っているときに、メッセージが表示されたりユーザーがヘルプを要求したりした場合に、ユーザーが元のウィンドウをクローズすると、メッセージとヘルプ・ウィンドウの両方が消滅する。

CNOS. セッション数変更 (Change number of sessions)。

コード・ページ (code page). (1) コード化文字セットを定義するテーブルであって、その定義は、言語用または国別テーブルの各コード・ポイントに文字の意味を割り当てることによって行われる。(2) 文字とその内部 2 進表

現との間のマッピング。(3) 図形文字および制御機能の意味をすべてのコード・ポイントに割り当てること。たとえば、8 ビット・コードについては文字と意味を 256 個のコード・ポイントに割り当て、7 ビット・コードについては文字と意味を128 個のコード・ポイントに割り当てる。(4) 印刷管理機能において、コード・ポイントと文字識別コードを関連付けるフォント・ライブラリー・メンバー。 コード・ページは、無効なコード・ポイントも識別する。(5) 16 進識別コードを図形文字に割り当てる、特別の割当て。(6) AFP サポートにおいて、コード・ポイントと図形文字識別コードとを関連付けるフォント・ファイル。

コマンド (command). (1) 端末から出される要求であって、特定のプログラムの動作または実行が行われるように要求するもの。(2) SNA において、伝送ヘッダー (TH)、要求ヘッダー (RH)、および場合によっては要求単位 (RU) の一部で設定されるフィールドであって、アクションまたはプロトコルを開始するもの。例として、(a) LU-LU セッションを活動化するコマンドである Bind Session (セッション制御要求単位)、(b) 連鎖の最後のRU の RH の中にある命令変更標識、(c) FID4 伝送ヘッダーの中にある仮想経路リセット・ウィンドウ標識がある。

コマンド域 (command area). 基本 CUA アーキテクチャーにおいて、コマンド入力フィールドを含んでいるパネル領域。

コマンド行 (command line). (1) 表示画面上で、一般に画面の最下部にあって、コマンドだけを入力することのできる表示行。(2) CUA アーキテクチャーにおいて、*コマンド域 (command area)* を意味する不適切な用語。

コマンド・プロンプト (command prompt). 処理されるコマンドをユーザーが入力してもよいことを示す 1 つの文字、または文字のストリング。

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 (Common Programming Interface for Communications (CPI-C)). 多様なアプリケーション環境からの増大する需要を満たし、通信プログラミング用の業界標準としてのオープン性を達成するための機能を含む、発展的なアプリケーション・プログラミング・インターフェース (API)。CPI-C を使用すると、(a) データの送受信、(b) プログラム間の処理の同期化、そして(c) パートナーへの通信エラーの通知などのプログラム間サービスにアクセスできる。

共通サービス API. コミュニケーション・マネージャー/2 において、ユーザー作成プログラム用のコミュニケーション・マネージャー/2 提供のサービスにアクセスするために使用できるアプリケーション・プログラミング・インターフェース (API) の verb。

共通トランスポート・セマンティックス (common transport semantics (CTS)). ネットワーキング・ブループリントにおいて、トランスポート層の上にあって、トランスポート提供者のサービスをトランスポート・ユーザーが使用できるようにするための層。トランスポート層プロトコル境界 *(transport-layer protocol boundary (TLPB))* も参照。

コミュニケーション・マネージャー/2 (Communications Manager/2). *Communications Server*およびパーソナル・コミュニケーションズ製品ファミリー *(Personal Communications product family)*を参照。 コミュニケーション・マネージャー/2 製品の機能は、Communications Server 製品とパーソナル・コミュニケーションズ製品ファミリーの中に組み込まれている。

Communications Server. 接続された複数のシステムまたはワークステーションに関連するアプリケーション・プログラムの開発および使用をサポートするIBM ライセンス・プログラム。 通信サーバーは、異なったプロトコルを使用してアプリケーション・プログラムを他のシステムまたはワークステーションへ接続する複数の並行接続機能を提供する。それはいくつかのアプリケーション・プログラミング・インターフェース(API) をサポートするが、これらのインターフェースは同時に呼び出されることが可能であり、またクライアント/サーバーおよび分散アプリケーション・プログラムを対象として設計されている。 通信サーバーは、ネットワーク管理に必要なインターフェースを含んでいる。

コンパクト・ディスク (compact disc (CD)). (1) 通常直径が 4.75 インチのディスクで、レーザーを使用してそこから光学的にデータを読み取ることができる。(2) 情報がらせんトラックに沿ってピット形式で記憶されているディスク。情報はコンパクト・ディスク・プレーヤーによりデコードされ、ほとんどのコンピューターで処理できるディジタル・オーディオ・データとして解釈される。

コンパイルする (compile). (1) 高水準言語で表現されたプログラムのすべてまたは一部を、中間言語、アセンブリー言語、または機械語で表現されたコンピューター・プログラムに変換すること。 (T) (2) 機械語以外のプログラム言語で書かれたコンピューター・プログラムから機械語プログラムを作成すること。その作成作業では、プログラムの全体的な論理構造を使用するか、各記号ステートメントについて複数のコンピューター命令を生成

するか、またはこれらの両方の作業を行い、さらにアセンブラーの機能を実行する。 (A) (3) ソース・プログラムを実行可能プログラム (オブジェクト・プログラム) に変換すること。(4) 高水準プログラム言語で書かれたプログラムを機械語プログラムに変換すること。

コンパイラー (compiler). (1) ソース・プログラムを実行可能プログラム（オブジェクト・プログラム）に変換するプログラム。(2) 擬似コードとして書かれた命令をデコードし、後で実行する機械語プログラムを作成するプログラム。

構成要素、構成装置 (component). 機能単位の一部である、ハードウェアまたはソフトウェア。

コンピューター (computer). さまざまな算術演算や論理演算など大量の計算を、実行中に人間の介在なしで行える機能単位。情報処理においては、コンピューターという用語は一般にディジタル・コンピューターを意味する。コンピューターは、1 つの独立した単位から成ることもあり、いくつかの相互接続された単位から成ることもある。 (T)

構成（configuration）. (1) 情報処理システムのハードウェアおよびソフトウェアが編成され、相互接続される様式。 (T) (2) システム、サブシステム、またはネットワークを構成する装置およびプログラム。(3) パーソナル・コミュニケーションズにおいて、パーソナル・コンピューターが、1 つまたは複数の接続タイプによって 1 つまたは複数のホスト・システムに接続されるときの配置。例として、SDLC、LAN、ASYNCH、X.25、または DFT がある。

構成ファイル(configuration file). システム装置またはネットワークの特性を指定するファイル。

構成管理 (configuration management). 物理および論理情報システム資源の両方、およびそれらの相互関係を識別するために必要な情報の制御。

構成サービス (configuration services). 制御ポイント (SSCP、NNCP、または ENCP) におけるネットワーク・サービス・タイプの 1 つ。 構成サービスは、物理装置、リンク、およびリンク・ステーションを活動化および非活動化し、またそれらの状況を記録する。

構成する (configure). システムに導入される装置、任意選択機能、およびプログラムを、そのシステムに対して記述すること。

輻輳（ふくそう）(congestion). *ネットワーク過密 (network congestion)*を参照。

論理積 (conjunction). 各オペランドのブール値がそれぞれ 1 である場合に限り、演算結果がブール値 1 になるようなブール演算。 (I) (A) *AND 演算 (AND operation)* と同義。

接続状態 (connected). VTAM において、ある物理装置 (PU) または論理装置 (LU) が、それぞれを制御するシステム・サービス制御点 (SSCP) を含むホスト・プロセッサーへの活動物理パスを持っている状態。

接続（connection）. (1) データ通信において、情報を伝達するために複数の機能単位の間で確立される関連付け。 (I) (A) (2) SNA において、異なったノードに存在する 2 つの論理装置 (LU) を相互にリンクして、それらの論理装置に通信を確立させるネットワーク・パス。(3) TCP/IP において、2 つのプロトコル・アプリケーションの間で信頼性のあるデータ・ストリーム送達サービスを提供するパス。 インターネットでは、1 つのシステムの TCP アプリケーションから他のシステムの TCP アプリケーションへと、接続が拡張される。

結合点マネージャー (connection point manager). SNA では、伝送制御層の構成要素の 1 つで、(a) 通常フロー要求のセッションレベル・ペーシングを行い、(b) 受信した要求単位の順序番号を検査し、(c) 要求単位が許容最大サイズを超えていないことを確認し、(d) ハーフセッションで着信要求単位をそれぞれの宛先に経路指定し、そして、(e) 暗号化が選択されている場合に FMD 要求単位の暗号化および暗号解読を行う。結合点マネージャーは、1 つのハーフセッションでの通常フローと急送フローの間の調整を行う。

注: ハーフセッションでは、送信側の結合点マネージャーは、発信要求応答単位 (RU) 用の要求応答ヘッダー (RH) を作成し、受信側の結合点マネージャーは、着信要求応答単位の先頭にある要求応答単位ヘッダーを解釈する。

接続性 (connectivity). (1) システムまたは装置を、他のシステムまたは装置に、変更なしに接続できる能力。(T)(2) さまざまな機能単位を無修正で接続できる能力。

競合 (contention). セッションにおいて、両方の NAU が同時に同じアクションを開始しようとしたときの状態。たとえば、半二重プロトコルで両方がデータを送信しようとした場合（半二重競合）や、両方がブラケットを開始しようとした場合（ブラケット競合）など。 セッションの開始時点で、一方の NAU が競合勝者として定義さ

れ、競合が生じたときはその NAU が優先される。 競合敗者がアクションを開始するには、競合勝者から明示または暗黙の許可を得る必要がある。

競合敗者セッション (contention-loser session). (1) ある NAU にとって、セッション開始時にその NAU が競合敗者として定義されているセッション。(2) *送信権要求者セッション (bidder session)* と同義。

競合勝者セッション (contention-winner session). (1) ある NAU にとって、セッション開始時にその NAU が競合勝者として定義されているセッション。(2) *ファースト・スピーカー・セッション (first-speaker session)* と同義。

制御ブロック(control block). (1) コンピューター・プログラムが制御情報を保管するために使用する記憶域。 (I) (2) IBM トークンリング・ネットワークにおいて、ある操作を要求するためにアプリケーション・プログラムからアダプター・サポート・インターフェースへ与えられる、特殊形式の情報ブロック。

制御文字 (control character). ある文字が特定の文脈で現れて、その文字が制御機能を指定する場合、その文字を制御文字と呼ぶ。 (T)

制御点 (control point (CP)). (1) APPN ノードまたはLEN ノードの構成要素の 1 つで、そのノードの資源を管理する。 APPN ノードでは、CP は他の APPN ノードとのCP-CP セッションに従事することができる。 APPN ネットワーク・ノードでは、CP は APPN ネットワーク内の隣接エンド・ノードへのサービスも提供する。(2) ノードの構成要素の 1 つで、そのノードの資源を管理するほか、オプションとしてネットワーク内の他のノードへのサービスも提供する。 例としては、タイプ 5 のサブエリア・ノードにおけるシステム・サービス制御点 (SSCP)、APPN ネットワーク・ノードにおけるネットワーク・ノード制御点 (NNCP)、および、APPN または LEN エンド・ノードにおけるエンド・ノード制御点 (ENCP) などがある。 SSCP および NNCP は他のノードにサービスを提供できる。

コントロール・プログラム (Control Program (CP)). VM/ESA において、単一のコンピューターの資源を管理して、あたかも複数のコンピューティング・システムが存在するかのような働きをさせるための構成要素。このような見せかけのシステム、つまり仮想計算機は、IBM システム/370、370-XA、または ESA などのコンピューターと機能的に同等のものとなる。

会話（conversation）. LU 6.2 セッションを使用して2 つのトランザクション・プログラム間で行われる論理接続。会話は、セッションの排他使用を獲得するためにブラケットで区切られる。

会話グループ識別子 (conversation group ID). 2 つの特定の LU 間または CP 間の特定のセッションの識別子。会話割振りコマンドで会話グループ識別子を指定すれば、要求された会話を、そのコマンドで識別するセッションに割り振ることができる。 その結果、一対のトランザクション・プログラム（各 LU について 1 つずつ）が、同じ会話グループ識別子を使用する関連のトランザクション・プログラムのペアとの間で、そのセッションを逐次的に共用することができる。

世界協定時 (coordinated universal time (UTC)). 国際無線通信諮問委員会 (CCIR) が定義および勧告し、国際度量衡局 (BIPM) が原子時計により維持している国際単位系 (SI) 秒を基準とする時間目盛。

注: 国際単位系は、メートル (meter)、キログラム (kilogram)、秒 (second) の 3 つの基本計測単位を基準とするもので、これらの単位の頭文字をとって「MKS 単位系」とも呼ばれる。

現実的なほとんどの目的に関しては、世界協定時 (UTC) は、イギリスのグリニッジを通る本初子午線（経度 0）における平均太陽時、つまり*グリニッジ標準時 (Greenwich mean time (GMT))* に等しいものとみなされる。*Z 時 (Z time)* および*ズールー時 (Zulu time)* と同義。

複写 (copy). 選択されたオブジェクトのコピーをクリップボードに置く選択機能。

相関係数. 事物間の関係を識別する情報。 例としては、応答の中にあって、対応する要求を識別する変数フィールドなどがある。

CP. (1) 制御点 (control point)。(2) VM では、制御プログラム (Control Program)のこと。

CRV. 暗号確認 (Cryptography verification)。

暗号確認 (CRV) 要求 (cryptography verification (CRV) request). 暗号セッション確立の一環として、 1 次論理装置 (PLU) が2 次論理装置に送る要求単位。これにより、SLU は、PLU が正しいセッション暗号キーおよび初期設定ベクトル (IV) を使用しているかどうかを確認することができる。

CS. 現在の状態 (Current state)。

CTS. (1) 共通トランスポート・セマンティックス (Common transport semantics)(2) 送信の消去。

顧客情報管理システム (Customer Information Control System (CICS)). 複数のリモート端末で入力された複数のトランザクションを、ユーザー作成のアプリケーション・プログラムで同時に処理できるようにした IBM ライセンス・プログラム。 データベースの構築、使用、および保守を行う機能が含まれている。

D

DACTLU. 論理装置非活動化 (Deactivate logical unit)。

DACTPU. 物理装置の非活動化 (deactivate physical unit)。

DAF (DAF). 宛先アドレス・フィールド (destination address field)。

DAF'. 宛先アドレス・フィールド・プライム記号 (destination address field prime)。

データ（data）. (1) 情報の再解釈可能な表現であって、その表現は通信、解釈、または処理に適した形式になっている。 データに対しては、人間または自動的手段により操作を行うことができる。 (T) (2) それらの文字またはアナログ量に意味が割り当てられるか、割り当てられる可能性があるもの。 (A) (3) 人間または自動装置による通信、解釈、または処理に適した形式で事実または説明を表現したもの。 データには、定数、変数、配列、文字ストリングなどがある。

注: プログラマーは、命令とその操作対象のデータを区別して考えるが、一般的な用語の意味としては、データにはプログラムおよびプログラム命令も含まれる。

データ回線 (data circuit). (1) 両方向のデータ通信の手段を提供する、関連付けられた一対の送信チャネルと受信チャネル。 (I)(2) SNA では、*リンク接続 (link connection)* の同義語。(3) *物理回線 (physical circuit)*および*仮想回線 (virtual circuit)*も参照。

注:

1. データ交換装置間のデータ回線には、データ交換装置で使用されているインターフェースのタイプに応じて、データ回線終端装置 (DCE) も含まれることがある。
2. データ・ステーションと、データ交換装置またはデータ集中装置との間のデータ回線には、データ・ステーション側のデータ回線終端装置が含まれるほか、データ交換装置またはデータ集中装置のロケーションにある DCE に類似の装置も含まれることがある。

データ・フロー制御 (data flow control (DFC)). SNA において、1 つのハーフセッションにおけるデータ・フロー制御層とセッション・パートナーにおけるデータ・フロー制御層との間で交換される要求と応答に使用される、要求/応答単位 (RU) のカテゴリー。

データ・フロー制御 (DFC) 層 (data flow control (DFC) layer). SNA において、ハーフセッションの中で次のことを行う層。すなわち、ハーフセッションが要求単位 (RU)を送信、受信、または同時的な送信と受信を行うかどうかを制御すること。関連した RU を RU 連鎖にまとめること。ブラケット・プロトコルを介してトランザクションを限定すること。セッションの活動化によって指定された制御モードに従って要求と応答のインターロックを制御すること。要求と応答を相関付けること。

データ・フロー制御 (DFC) プロトコル (data flow control (DFC) protocol). SNA における要求と応答の順序付け規則であって、この規則によって、セッション内のネットワーク・アドレス単位 (NAU) はデータ転送および他の操作を調整し制御する。例として、ブラケット・プロトコルがある。

データ・リンク（data link）. SNA における *リンク (link)*の同義語。

データ・リンク制御 (data link control (DLC)). 秩序立った情報交換を行うために（SDLC リンクやトークンリングなどの）データ・リンク上でノードが使用する一連の規則。

データ・リンク制御（DLC）層（data link control (DLC) layer）. SNA では、2 つのノード間でのリンクを介したデータ転送をスケジュールし、そのリンクのエラー制御を行うリンク・ステーションから成る層。データ・リンク制御の例としては、ビット順次リンク接続用の SDLC やシステム/370 チャネル用のデータ・リンク制御などがある。

注: DLC 層は、通常は物理的なトランスポート・メカニズムから独立していて、上位の層に送られるデータの整合性を確保できるようになっている。

データ・セット (data set). (1) *ファイル (file)*の同義語。(2) *モデム (modem)* の不適語。

データ・ストリーム（data stream）. (1) データ・リンク上で、通常は 1 つの読取りまたは書込み操作中に送られ

るすべての情報 (データおよび制御コマンド)。(2) 定義された形式を使用して、文字または 2 進数形式で送信されているか、送信されようとしている、データ要素の連続的なストリーム。

データ・タイプ (data types). NetView プログラムにおいて、パネルの編成の記述。データ・タイプには、アラート、イベント、および統計がある。データ・タイプと資源タイプおよび表示タイプを組み合わせたものが、NetView プログラムの表示編成の記述となる。*表示タイプ (display types)* および*資源タイプ (resource types)* も参照。

DBCS. 2 バイト文字セット (double-byte character set)。

非活動化（する） (deactivate). ノードの資源がサービスされないようにし、資源を動作不能にし、または資源の実行設計機能をその資源が実行できない状態にすること。*活動化（する) (activate)* と対比。

割振り解除 (deallocate). LU 6.2 アプリケーション・プログラミング・インターフェース (API) verb であって、会話を終わらせてセッションを将来の会 話のために解放するもの。 *割振り (allocate)* と対比。

暗号解読 (decryption). コンピューター・セキュリティーにおいて、コード化テキストまたは暗号テキストを平文に変換すること。

省略時値 (default). 明白に指定されていないときに想定される属性、条件、またはオプションに関する用語。(I)

確定応答 (definite response (DR)). SNA において、要求の受信側に、要求連鎖に対する応答を肯定か否定かに関係なく無条件に戻すように指示するために、要求ヘッダーの応答形式要求フィールドで要求するプロトコル。*例外応答 (exception response)* および*無応答 (no response)* と対比。

定義ファイル (definition file). 省略時によってロードされるファイルであって、特定のセッションに適合させることのできるキーボードと画面カラーの特性を含むもの。定義ファイルの行は、厳密なレイアウトに従ってコーディングする必要がある。

定義ステートメント. (1) VTAM において、ネットワークの要素を記述するステートメント。(2) NCP において、NCP に対して資源を定義する命令のタイプ。(3) *マクロ命令 (macroinstruction)* も参照。(4) 335ページの図 9、335ページの図 10、および335ページの図 11を参照。

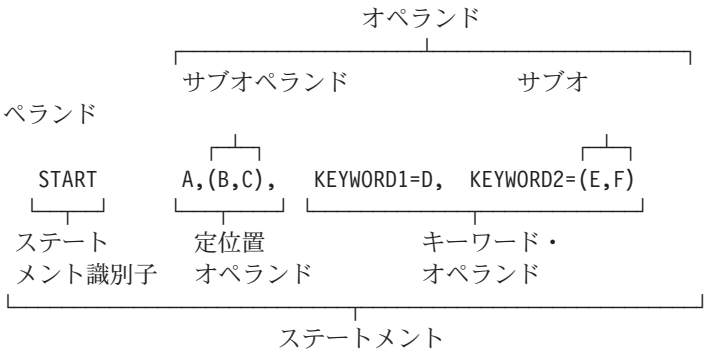


図 9. 言語ステートメントの例

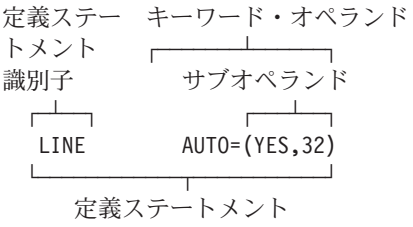


図 10. NCP 定義ステートメントの例

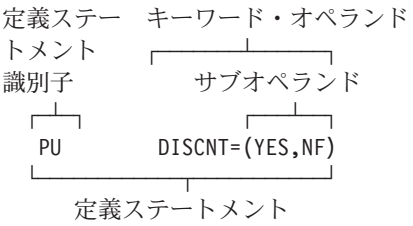


図 11. VTAM 定義ステートメントの例

従属 LU (dependent LU). *SSCP 従属 LU (SSCP-dependent LU)* を参照。

従属 LU リクエスター (dependent LU requester (DLUR)). 従属 LU を所有しているが、それらの従属LUに関する SSCP サービスを従属 LU サーバーが提供することを要求する、APPN エンド・ノードまたは APPN ネットワーク・ノード。

宛先 (destination). (1) ノード、ステーション、または特定の端末のように、情報が送られるポイントまたはロケーション。(2) メッセージまたは他のデータが方向付けられる外部の論理装置 (LU) またはアプリケーション・プログラム。

宛先アドレス (destination address). 情報が送られるロケーションを識別するコード。

宛先アドレス・フィールド (destination address field (DAF)). SNA における FID0 またはFID1 伝送ヘッダーの中のフィールドであって、宛先のネットワーク・アドレスを含むもの。*起点アドレス・フィールド (origin address field (OAF))* と対比。

デバイス、装置 (device). 特定の目的をもつ、機械的、電気的、または電子的な仕組み。

DFC. データ・フロー制御 (data flow control)。

ダイアログ (dialog). (1) ユーザーとコンピューターとの間の対話。(2) 対話型システムで、関連した一連の照会と応答。これは 2 人で交わす会話に似ている。(3) AIXwindows ツールキットにおいて、アプリケーションとそのユーザーとの間の両方向テキスト・インターフェース。このインターフェースは、ウィジェットとガジェットの集合から成っている。これには、DialogShell ウィジェット、BulletinBoard ウィジェット（または、BulletinBoard ウィジェットまたは他のコンテナー・ウィジェットのサブクラス）に加えて、Label、PushButton、Text ウィジェットなどの子が含まれる。

直接活動化 (direct activation). VTAM において、特定の資源を指定する活動化コマンドの結果として、その資源が活動化されること。*自動活動化 (automatic activation)* を参照。*間接活動化 (indirect activation)* と対比。

ディレクトリー（directory）. (1) 対応するデータを項目を識別および参照する表。 (I) (A) (2) ファイル・システムにおいて、複数のファイルを階層的にまとめてグループにし、そのグループに名前をつけたもの。(3) APPN ノード内にあって、資源（主として論理装置）の名前をリストし、各資源が位置しているノードの CP 名を記録するデータベース。*分散ディレクトリー・データベース (distributed directory database)* および*ローカル・ディレクトリー・データベース (local directory database)* を参照。

使用不能にする（disable）. 機能しないようにすること。

表示 (display). データをメッセージに見えるように提示するもの。 (I) (A)

表示レベル (display levels). *表示タイプ (display types)* の同義語。

表示タイプ (display types). NetView プログラムにおいて、パネルの編成を記述する概念。表示タイプは、合計、最新、ユーザー処置、および詳細として定義される。表示タイプと資源タイプおよびデータ・タイプを組み合わせたものが、NetView のパネル編成の記述となる。*データ・タイプ (data types)* および*資源タイプ (resource types)* も参照。*表示レベル (display levels)* と同義。

分散ディレクトリー・データベース (distributed directory database). APPN ネットワーク内に散在する個々のディレクトリー内に維持されている、そのネットワーク内の全資源の完全リスト。 各ノードはそれぞれ完全ディレクトリーの一部ずつを分かち持っているが、 1 つのノードがリストの全体を保有している必要はない。このリストの項目は、システム定義、操作員の処置、自動登録、実行中のネットワーク探索手順により、作成、変更、および削除される。 *分散ネットワーク・ディレクトリー (distributed directory directory)* およびネットワーク・ディレクトリー・データベース *(network directory database)* と同義。

分散ネットワーク・ディレクトリー (distributed network directory). *分散ディレクトリー・データベース (distributed directory database)* の同義語。

分散処理 (distributed processing). 複数のリンクされたシステムにまたがって行われる処理。

DLC. データ・リンク制御 (data link control)。

DLL. ダイナミック・リンク・ライブラリー (dynamic linklibrary)。

定義域 (domain). (1) データ処理の資源が共通の制御の下に置かれるコンピューター・ネットワークの部分。(T)(2) 共用されるネットワーク資源を 1 つの論理システムの中で割り振るサーバーの集合体。(3) SNA では、*エンド・ノード定義域 (endnode domain)*、*ネットワーク・ノード定義域 (network nodedomain)*、およびシステム・サービス制御点 *(SSCP)定義域 (system services control point (SSCP) domain)*を参照。(4) 開放型システム間相互接続 (OSI) において、分散システムの一部、または、共通のポリシーが提供される一組の管理オブジェクト。(5) *管理定義域 (Administrative Domain)* および*定義域名 (domain name)* を参照。

定義域名 (domain name). インターネットのプロトコルの組において、ホスト・システムの名前。定義域名は、区切り文字で区切られた一連のサブネームから成る。たとえば、ホスト・システムの完全修飾定義域名 (FQDN) が ralvm7.vnet.ibm.com であるとすれば、次のどれも定義域名である:

- ralvm7.vnet.ibm.com
- vnet.ibm.com
- ibm.com

定義域操作員 (domain operator). 複数定義域ネットワークにおいて、1 つのシステム・サービス制御点 (SSCP) が制御する資源の働きを制御する人またはプログラム。*ネットワーク操作員 (network operator)* も参照。

DOS. ディスク・オペレーティング・システム。 *IBM ディスク・オペレーティング・システム (IBM Disk Operating System)* を参照。

DOS セッション (DOS session). パーソナル・コンピューターが、ディスク・オペレーティング・システム (DOS) の下で実行しながら、独立型コンピューターとして動作するセッション。ホスト・セッション *(host session)*を参照。

2 バイト文字セット (double-byte character set (DBCS)). 複数の文字の集合であって、各文字が 2 バイトで表されるもの。 日本語、中国語、韓国語など、256 個のコード・ポイントで表せる記号より多くの記号を含む言語では、2 バイト文字セットが必要です。 各文字にそれぞれ 2 バイトが必要なので、DBCS 文字を入力、表示、印刷するには DBCS をサポートするハードウェアおよびプログラムが必要になる。 *1 バイト文字セット (single-byte character set (SBCS))* と対比。

待ち行列処置 (drain). パートナー論理装置とのセッションを非活動化する前に、保留状態にある割振り要求を処理すること。これは LU 6.2 だけに適用される。

ドライブ (drive). 周辺装置の 1 つ。特に記憶媒体を対象として働くもの。

二重、両面印刷 (duplex). データの送信と受信を同時に行うことができる通信に関する用語。 *全二重 (full-duplex)* と同義。 *半二重 (half-duplex)* と対比。

動的 (dynamic). (1) プログラミング言語において、プログラムの実行中にのみ設定することができる特性に関する用語。たとえば、可変長データ・オブジェクトの長さは動的である。 (I)(2) あらかじめ定まった時間または固定した時間ではなく、必要な時間に生じる動作に関する用語。(3) *静的 (static)*と対比。

ダイナミック・リンク (dynamic linking). OS/2 オペレーティング・システムにおいて、ルーチンへのプログラムの接続を、ロード時間または実行時間まで遅延させること。

ダイナミック・リンク・ライブラリー (dynamic link library (DLL)). リンク中ではなく、ロード時間または実行時間中にプログラムにバインドされる実行可能コードおよびデータを含むファイル。 ダイナミック・リンク・ライブラリー内のコードおよびデータは、いくつかのアプリケーションが同時に共用できる。

E

EBCDIC. 拡張 2 進化 10 進コード。 256 個の 8 ビット文字で構成されるコード化文字セット。

EC. 技術変更 (Engineering change)。

編集 (Edit). データを変更するために使用する他の選択項目にアクセスするための、メニュー・バー上の選択項目。

要素 (element). (1) ネットワーク・アドレスのフィールドの 1 つ。(2) SNA において、サブエリアの中にあって要素アドレスで識別される特定の資源。サブエリア *(subarea)* も参照。

エミュレーター (emulator). 1 つの装置を、他の異なった装置のように動作させるプログラム。たとえば、パーソナル・コミュニケーションズは、サポートされるパーソナル・コンピューターおよびプリンターを、それらが 3270 シリーズのワークステーションであるかのように動作させる。

使用可能 (enabled). (1) ある種の割込みを許すような、処理装置の状態に関する用語。(2) 伝送制御装置または音声応答装置が回線上の到着呼出しを受け入れることができる状態に関する用語。

暗号化データ (enciphered data (ED)). 無許可のユーザーや第三者には意味が分からないようになっているデータ。

暗号化 (encryption). コンピューター・セキュリティーにおいて、データを理解不能な形式に変換して、元のデータを入手できないようにするか、または暗号解読処理によってのみ入手できるようにする処理。

終了ブラケット (end bracket). SNA において、ブラケットの最後の連鎖の最初の要求の要求ヘッダー (RH) に存在するブラケット終了標識の値 (2 進の 1)。その値はブラケットの終了を示す。*開始ブラケット (begin bracket)* と対比。 *ブラケット (bracket)* も参照。

エンド・ノード (end node (EN)). (1) *拡張対等通信ネットワーキング (APPN) エンド・ノード (Advanced Peer-to-Peer Networking (APPN) end node)*および*ローエントリー・ネットワーキング (LEN) エンド・ノード (low-entry networking (LEN) endnode)*を参照。(2) 通信において、単一のデータ・リンクに頻繁に接続され、中間経路指定機能として働くことのできないノード。

エンド・ノード定義域 (end node domain). エンド・ノードの制御点、接続したリンク、およびそのローカルLU。

エントリー・ポイント (entry point (EP)). (1) コンピューター・プログラム、ルーチン、またはサブルーチンに入ったときに実行される最初の命令のアドレスまたはラベル。 1 つのコンピューター・プログラム、ルーチン、またはサブルーチンに、それぞれ異なる機能や目的に対応する複数の異なるエントリー・ポイントを設けることもできる。 (I) (A) (2) SNA において、分散ネットワーク管理サポートを提供するタイプ 2.0、タイプ 2.1、タイプ 4、またはタイプ 5 のノード。 この種のノードは、自身に関するネットワーク管理データ、および自身が制御する資源を、集中処理用のフォーカル・ポイントに送り、さらに、資源を管理し制御するためにフォーカル・ポイントが開始したコマンドを受け取って実行する。

環境変数（environment variable）. (a) オペレーティング・システムまたはその他のプログラムをどのように実行するか、または、(b) オペレーティング・システムがどのような装置を認識するかを指定する変数。

ER. 明示経路 (Explicit route) を参照。

エラー (error). 計算、観察、または測定された値または条件が、真の、または指定された、または理論上正しい値または条件と矛盾している状態。 (I) (A)

エラー・ログ (error log). 製品またはシステム内のデータ・セットまたはファイルであって、そこにエラー情報から記憶され、後でアクセスされる。

イーサネット (Ethernet). 複数のステーションが事前の調整を必要とせずに自由に伝送媒体にアクセスできる10 Mbps ベースバンド・ローカル・エリア・ネットワークで、搬送波検知および送信延期を使用して競合を回避し、衝突検出および遅延再送を使用して競合を解決する。イーサネットは、搬送波検知多重アクセス／衝突検出 (CSMA/CD) を使用する。

事象 (event). あるタスクにとって何か意味のあることが発生すること。たとえば、SNMP トラップの発生、ウィンドウまたはサブマップのオープン、または、非同期操作の完了など。

例外 (exception). 異常な状態。たとえば、データ・セットまたはファイルを処理しているときに起こる I/O エラー。

例外要求 (exception request (EXR)). SNA においてエラーが検出された他のメッセージ単位を置換し、そのエラーを識別するセンス・データを伝える要求。

例外応答 (exception response (ER)). SNA において、要求が受信時に受諾不能かまたは処理できない場合に限り応答を戻すこと、つまり、否定応答は必要だが肯定応答は不要であることを受信側に指示するために、要求ヘッダーの応答形式要求フィールドで要求するプロトコル。*確定応答 (definite response)* および*無応答 (no response)* と対比。

実行する (execute). プログラムの全体または一部によって指定されたアクションを実行すること。 (T)

終了する、出口 (exit). コンピューター・プログラムの一部分の実行を終了させるために、その一部分の中の命令を実行すること。 コンピューター・プログラムのこのような部分には、ループ、サブルーチン、モジュールなどがある。 (T)

急送フロー (expedited flow). SNA において、ネットワーク制御、セッション制御、および各種のデータ・フロー制御要求応答単位 (RU) を運ぶために使用される伝送ヘッダー (TH) で指定されるデータ・フロー。急送フローは、通常フロー（主としてエンド・ユーザー・データを運ぶ）とは異なるもので、通常フローに影響を与えるコマンド用として使用できる。*通常フロー (normal flow)* と対比。

注: 通常フローも急送フローも、 1 次から 2 次へ、および 2 次から 1 次への両方向に流れる。通常フローの場合も急送フローの場合も、 1 つのフロー上の要求および応答は、通常、パス内で順次に処理されるが、ハーフセッションでの待合せポイントにおいては、また境界機能のハーフセッション・サポートの場合は、パス内で急送フロー・トラフィックの方が通常フロー・トラフィックより先に進むことがある。

明示経路 (explicit route (ER)). SNA において、2 つのサブエリア・ノードを接続する 1 つまたは複数の伝送グループ。明示経路は、起点サブエリア・アドレス、宛先サブエリア・アドレス、明示経路番号、および反転明示経路番号により識別される。 *仮想経路 (virtual route(VR))* と対比。

EXR. 例外要求 (exception request)。

EXT. 外部トレース・ファイル (external trace file)。

拡張 2 進化 10 進コード (extended binary-coded decimal interchange code (EBCDIC)). 8 ビット・コード化文字より成る文字セットを使用する標準コードであって、パーソナル・コミュニケーションズによってパーソナル・コンピューターとホスト・システムとの間の情報交換に使用されるもの。*情報交換用米国標準コード (American National Standard Code for Information Interchange)*も参照。

F

障害 (fault). ある機能装置の本来の機能が実行できなくなるような不慮の条件。 (I) (A)

FDX. 全二重 (full-duplex)。

機能 (feature). 顧客が別個に発注することのできる IBM 製品の一部分。

FID. フォーマット識別 (Format identification)。

フィールド（field）. (1) データを入れるために使用されるレコードまたはパネルの領域。(2) IBM 3270 データ・ストリームにおいて、同じ特性を有する表示空間の連続した位置より成るグループ。フィールドの開始部分にあるフィールド属性は表示空間の特性を定義する。(3) ウィンドウ内の識別可能領域。 フィールドの例としては、テキストを入力または置くための入力フィールド、 1 つの選択項目を選択するためのラジオ・ボタン選択項目フィールドがある。

ファイル (file). 1 単位として記憶され処理される一組のレコードに名前を付けたもの。 (T) データ・セット *(data set)* と同義。

ファイル指定 (file specification). コミュニケーション・マネージャー/2において、1 つのファイルの完全識別子、つまり、ドライブ、パス、ファイル名、およびファイル拡張子。 そのフォーマットはファイルの記憶媒体に依存する。たとえば、`C:¥PATH¥FILENAME.EXT`。

ファイル転送(file transfer). 1 つまたは複数のファイルを、データ・リンクを介してシステムから別のシステムへと送ること。

先頭連鎖 (first-in-chain (FIC)). 要求ヘッダー (RH) 連鎖開始標識がオンで、RH 連鎖終了標識がオフである要求単位 (RU)。*RU 連鎖 (RU chain)* も参照。

ファースト・スピーカー (first speaker). *ファースト・スピーカー・セッション(first-speaker session)* を参照。

ファースト・スピーカー・セッション (first-speaker session). セッション活動化の時点で、(a) 相手のハーフセッションからの許可を要求せずにブラケットを開始でき、(b) 両方のハーフセッションが同時にブラケットを開始しようとしたときに競合勝者になることが定義されているハーフセッション。 *競合勝者セッション (contention-winner session)* の同義語。 *送信権要求者セッション (bidder session)* と対比。

固定ペーシング (fixed pacing). *固定セッション・レベル・ペーシング (fixed session-level pacing)* の同義語。

固定セッション・レベル・ペーシング (fixed session-level pacing). セッション・レベル・ペーシングの方式の 1 つ。この方式では、セッション活動化の時点で初期化された固定ペーシング・サイズを使用して、データ転送速度が制御される。 *固定ペーシング (fixed pacing)* と同義。*最適セッション・レベル・ペーシング (adaptive session-level pacing)* と対比。

フラグ（flag）. (1) 他の処理のために選択する情報項目にマーク付けすること。 (T) (2) 何らかの条件（たとえばワードの終りなど）の発生を知らせる文字。 (A) (3) ワードの終わりや、データ転送ブロックの開始または終了などのオカレンスまたは境界を、マークする文字またはビットの順序列。

フロー (flow). NetDA/2 において、 1 つのノード、接続、または経路を、一定の時間内に両方向に通過できるトラフィックの量。

フロー制御 (flow control). (1) SNA において、データ・トラフィックがネットワークの複数の構成要素の間を通過する速度を管理する処理。 フロー制御の目的は、ネットワーク内の輻輳（ふくそう）を最小限に抑えながら、メッセージ単位のフロー速度を最大限にすることにある。つまり、受信側および中間経路指定ノードのどのバッファーにもオーバーフローが生じることなく、また受信側

に後続メッセージの待機状態を続けさせることもないような状態を確保することである。(2) ペーシング *(pacing)*も参照。

フラッシュ (flushing). 論理装置 (LU) 6.2 において、トランザクション・プログラムによって生成され、バッファー内に残っているすべてのデータを、ネットワークを介して送信する処理。トランザクション・プログラムは、flush verb を発行してこのプロセスを開始する。 ネットワーク操作員がこのコマンドを出したときも、フラッシュが行われる。

FMD. 機能管理データ (Function management data) を参照。

フォアグラウンド・プロセス (foreground process). (1) AIX オペレーティング・システムにおいて、その実行が完了するまでは他のコマンドをシェルに対して発行することのできないプロセス。フォアグラウンド・プロセスは、フォアグラウンド・プロセス・グループに入っているが、このグループは、端末が生成したシグナルを受信する。(2) *バックグラウンド・プロセス (background process)* と対比。

外部ホスト (foreign host). *リモート・ホスト (remote host)* の同義語。

フォークする (fork). AIX オペレーティング・システムにおいて、子プロセスを作成し開始すること。

形式識別 (FID) フィールド (format identification (FID) field). SNA において、各伝送ヘッダー (TH) の中に存在し、その TH の形式を示すフィールド。それによって、ある種のフィールドの存在または不存在が示される。TH形式は、その TH がどのタイプのノード間を通過するかによって異なる。 FID には次の 6 タイプがある。

- FID0。隣接するサブエリア・ノード間の非 SNA 装置が関与するトラフィックにおいて、一方または両方のノードが明示ルート・プロトコルおよび仮想ルート・プロトコルをサポートしていない場合に使用される。
- FID1。隣接するサブエリア・ノード間の SNA 装置が関与するトラフィックにおいて、一方または両方のノードが明示ルート・プロトコルおよび仮想ルート・プロトコルをサポートしていない場合に使用される。
- FID2。サブエリア・ノードと、それに隣接するタイプ 2 の周辺ノードとの間のトラフィックに使用される。
- FID3。サブエリア・ノードと、それに隣接するタイプ 1 の周辺ノードとの間のトラフィックに使用される。
- FID4。隣接するサブエリア・ノード間のトラフィックで、両方のノードが明示経路プロトコルおよび仮想経路プロトコルをサポートしている場合に使用される。
- FIDF。隣接するサブエリア・ノードが両方とも明示経路プロトコルおよび仮想経路プロトコルをサポートしている場合に、それらのノード間で送られるコマンド（たとえば伝送グループ制御など）に使用される。

FQDN. 完全修飾定義域名 (Fully qualified domain name) を参照。

フレーム（frame）. (1) 開放型システム間相互接続アーキテクチャーにおける構造の 1 つ。特定分野の知識を表現するために使用され、特定の属性の値を受け入れることのできるスロットで構成され、フレームに適切なプロシャージャーを付加することにより推論を導き出すことができる。 (T)(2) IBM トークンリング・ネットワークなど、ある種のローカル・エリア・ネットワークにおける伝送単位。 区切り文字、制御文字、情報、および検査文字から成る。(3) SDLC において、すべてのコマンド、すべての応答、および SDLC 操作手順を使用して伝送されるすべての情報の伝達手段。(4) 通信プロトコルの 1 つのタイプのフィールド仕様に合致したフィールドから構成されるデータ構造 (データ・フレーム)。データ・リンクの間で行われるデータ転送を制御するために、フレームが使用される。(5) SDLC において、開始フラグおよび終了フラグによって限定される、ビットの順序列。X.25 パケット交換データ・ネットワークにおいて、フレームは開始フラグおよび終了フラグによって限定された 8 ビット・バイトの順序列から構成される。X.25 におけるフレームは各種の機能、データ転送、および伝送チェックを制御する。

周波数 (frequency). ヘルツで表される信号の振幅速度。

全二重 (full-duplex (FDX)). *二重 (duplex)*の同義語

完全修飾定義域名 (fully qualified domain name (FQDN)). インターネットのプロトコルの組において、定義域名のすべてのサブネームを含むホスト・システムの名前。
完全修飾定義域名は、たとえば、`ralvm7.vnet.ibm.com` のようになる。 *ホスト名 (host name)* も参照。

完全修飾名 (fully qualified name). (1) SNA において、*ネットワーク修飾名 (network-qualified name)* の同義語。(2) インターネットのプロトコルの組においては、*完全修飾定義域名 (fully qualified domain name (FQDN)* を参照。

関数呼出し (function call). 現在の関数から指定の関数へ実行パスを移し、呼び出された関数によって提供された戻り値を評価する式。 関数呼出しには、制御を渡す相手の関数の名前と、括弧に入れた値のリストが含まれる。

機能管理データ (function management data (FMD)). 論理装置 (LU) 間で交換されるエンド・ユーザー・データ用として、また、LU、PU、および制御点のネットワーク・サービス構成要素間で交換される要求および応答用として使用される RU カテゴリー。

G

ゲートウェイ（gateway）. (1) ネットワーク・アーキテクチャーの異なる2 つのコンピューター・ネットワークを相互に接続する機能単位。 ゲートウェイは、アーキテクチャーの異なるネットワークまたはシステムを接続する。 ブリッジは、同一または類似したアーキテクチャーを持つネットワークまたはシステムを相互に接続する。 (T) (2) 機械とプログラムの組合せであって、独立の SNA ネットワーク間でアドレス変換、名前変換、およびシステム・サービス制御点 (SSCP) 再経路指定を行い、これらのネットワークが通信できるようにするもの。ゲートウェイは、1 つのゲートウェイ NCP と、1 つ以上のゲートウェイ VTAM から成る。(3) IBM トークンリング・ネットワークにおいて、1 つのローカル・エリア・ネットワークを、他のローカル・エリア・ネットワークまたは異なった論理リンク・プロトコルを使用するホストに接続する装置およびその関連ソフトウェア。(4) AIX オペレーティング・システムにおいて、リンク層よりも上部で動作して、必要に応じて、あるネットワークで使用しているインターフェースおよびプロトコルに変換するエンティティー。(5) TCP/IP においては、*ルーター (router)* の同義語。(6) ローカル・エリア・ネットワークの中の 1 つのステーションであって、このステーションを介して、ホスト・システムまたは別個のネットワークへの接続が確立される。

GDS. 汎用データ・ストリーム (general data stream)。

汎用データ・ストリーム (general data stream (GDS)). LU 6.2 セッション中の会話に使用されるデータ・ストリーム。

汎用データ・ストリーム (GDS) 変数 (general data stream (GDS) variable). RU 副構造のタイプの 1 つ。先頭に識別子と長さフィールドがあり、その後にアプリケーション・データ、ユーザー制御データ、または SNA 定義の制御データが続く。

汎用 UNBIND (generic unbind). *セッション非活動化要求 (session deactivation request)*の同義語。

GMT. グリニッジ標準時 (Greenwich mean time)を参照。

グリニッジ標準時 (Greenwich mean time (GMT)). イギリス、グリニッジを通る本初子午線（経度 0）における平均太陽時。グリニッジ標準時は、*Z 時 (Z time)* または *ズールー時 (Zulu time)* とも呼ばれる。

注: *グリニッジ標準時 (Greenwich mean time (GMT))* と *世界協定時 (coordinated universaltime (UTC))* はしばしば混用されるが、この 2 つは同義ではない。グリニッジ標準時は近似値である。現在は、秒は天文現象の観点から定義されてはいないので、この時間目盛を表す名称としては、*世界協定時 (UTC)* の方が適切である。

グループ ID (group ID (GID)). (1) RACF において、1 つのグループを識別する 1 ～ 8 個の文字のストリング。最初の 1 文字は、A～Z、＃、＄または@でなければならない。その他の文字には、A～Z、＃、＄、@または 0 ～ 9 を使用できる。(2) AIX オペレーティング・システムにおいて、特定のグループ名に対応する番号。グループ ID は、しばしば、グループ名を値としてとるコマンドの中に代入できる。

H

半二重 (half-duplex (HD、HDX)). データ通信において、一度に一方向のみに行われる伝送に関する用語。*二重 (duplex)* と対比。

ハーフセッション (half-session). セッション層構成要素の 1 つで、セッションの一方の側を形成するデータ・フロー制御構成要素と伝送制御構成要素の組合せから成っている。 *セッション・コネクター (session connector)* も参照。

handle. 拡張 DOS および OS/2 オペレーティング・システムによって作成された 2 進値であって、ドライブ、ディレクトリー、およびファイルを識別し、そのファイルを見つけてオープンできるようにするもの。

HD. 半二重 (half-duplex)。

ヘッダー (header). (1) ユーザー・データの前に置かれる、システム定義の制御情報。(2) メッセージ用の制御情報を含むメッセージ部分。制御情報としては、1 つまたは複数の宛先フィールド、発信ステーションの名前、

入力順序番号、メッセージのタイプを示す文字ストリング、およびメッセージの優先順位などがある。

ヘッダー・ファイル (header file). *組込みフィールド (include file)*の同義語。

ヘルプ (Help). ユーザーが、オブジェクト、選択項目、タスク、および製品に関する有用な情報にアクセスするために使用できる選択項目。ヘルプ選択項目は、メニュー・バーに表示されることも、押しボタンの 1 つとして表示されることもある。

16 進 (hexadecimal). (1) 16 個の異なった値または状態をもつことができる選択または条件に関する用語。 (I)(2) 基数 16 を使用する固定基数の数体系に関する用語。 (I)(3) 基数 16 の数体系に関する用語。16 進数は 0 から9 および A から F までの範囲にあり、A は 10 を表し F は 15 を表す。

強調表示 (highlighting). 表示する要素またはセグメントを、視覚的な属性を変更することにより強調すること。 (I) (A)

ホスト (host). (1) インターネットのプロトコルの組において、終端システム。 終端システムは任意のワークステーションでよい。つまり、メインフレームでなくてもかまわない。(2) *ホスト・プロセッサー (host processor)* を参照。

ホスト LU (host LU). SNA において、ホスト・プロセッサーにある論理装置。たとえば、VTAM アプリケーション・プログラムなど。

ホスト名 (host name). インターネットのプロトコルの組において、マシンに与えられている名前。場合によっては、「ホスト名」は*完全修飾定義域名 (fully qualified domain name (FQDN))*を意味し、また完全修飾定義域名の、最も具体的なサブネームを意味する。
たとえば、`ralvm7.vnet.ibm.com` が完全修飾定義域名であるとすれば、次のいずれもホスト名とみなすことができる:

- `ralvm7.vnet.ibm.com`
- `ralvm7`

ホスト・プロセッサー (host processor). (1) ユーザー・アプリケーション・ネットワークの全体または一部を制御するプロセッサー。 (T)(2) ネットワークにおいて、データ通信アクセス方式が存在している処理装置。

ホスト・セッション (host session). パーソナル・コンピューターがホスト・システムと通信できるようにする論理接続。セッションは LU アドレス、LT 番号、またはセッション ID によって識別できる。*DOS セッション (DOS session)*を参照。*論理端末 (logical terminal)*も参照。

ホスト・システム (host system). パーソナル・コミュニケーションズにおいて、SDLC、LAN、ASYNCH、X.25、または DFT 接続によって 1 つまたは複数のパーソナル・コンピューターにリンクされたコンピューター。

I

IBM ディスク・オペレーティング・システム (IBM Disk Operating System (DOS)). MS-DOS に基づいたディスク・オペレーティング・システムであって、すべての IBM パーソナル・コンピューターと共に動作するもの。

IBM オペレーティング・システム/2 (IBM Operating System/2 (OS/2)). パーソナル・コンピューター用のオペレーティング・システムとして使用できる IBM ライセンス・プログラム。 OS/2 ライセンス・プログラムは、同時に複数のタスクを実行することができる。

IBM トークンリング・ネットワーク. ローカル・サイトで情報処理機器を相互に接続するための汎用ベースバンド・ローカル・エリア・ネットワーク。これはトークンリング・アクセス・プロトコルを使用し、4 または 16 Mbps のデータ転送速度をサポートする。それは IEEE 802.5 (トークンリング) およびIEEE 802.2 (論理リンク制御) の標準に準拠している。

アイコン (icon). イメージ、イメージ・バックグラウンド、およびラベルで作られる、オブジェクトのグラフィカル表現。

ID. (1) 識別子（Identifier）。(2) 識別（Identification）。

IDLC. 統合データ・リンク制御 (Integrated data link control) を参照。

非活動、非活動状態 (inactive). (1) 操作可能でない。(2) 1 つのノードまたは装置が他のノードまたは装置に接続されていないか、または接続の準備ができていないこと。(3) *活動、活動状態 (active)* と対比。

インバウンド (inbound). 通信において、ネットワークから受信するデータ。

組込みファイル (include file). 機能、プログラム、またはユーザーのグループによって使用される宣言を含んでいるテキスト・ファイル。 *ヘッダー・ファイル (header file)* と同義。

間接活動化 (indirect activation). VTAM において、資源階層における上位資源を指定した活動化コマンドに関連するSCOPE または ISTATUS 仕様の結果として、下位資源が活動化されること。*直接活動化* と対比。

情報 (I) 形式 (information (I) format). 情報転送のために使用されるフォーマット。

情報 (I) フレーム (information (I) frame). 番号を付けられた情報転送に使用されるI フォーマットの中のフレーム。

INITIATE. LU-LU セッションの確立を求めるために、論理装置 (LU) がシステム・サービス制御点 (SSCP) に送るネットワーク・サービス要求。

入出力 (input/output (I/O)). (1) 入力、出力、またはその両方を意味する。(2) データ入力、データ出力、またはその両方に関与する装置、プロセス、またはチャネルを意味する。

インスタンス (instance). AIX オペレーティング・システムにおいて、抽象的なオブジェクト・クラスを実体のあるものとして認識したもの。ウィジェットまたはガジェットのインスタンスは、特定のグラフィカル・オブジェクトを実行時に画面上で生成するために使用する外観および性質の詳細情報が入った、特定のデータ構造となる。

INT. 内部トレース・テーブル (Internal trace table)。

統合データ・リンク制御 (integrated data link control (IDLC)). 全二重ハイレベル・データ・リンク制御 (HDLC) プロトコル。 それは、CCITT Q.922 標準または拡張リンク・アクセス・プロシージャー(LAPE) の IBM 実施版である。 IDLC は、全二重広域ネットワーク (WAN) を介した 2 地点間ワークステーション接続をサポートしている。

対話式 (interactive). ユーザーとコンピューターとの間の情報交換に関する用語。

インターフェース (interface). (1) 場合に応じて機能特性、信号特性、または他の特性によって定義されるような 2 つの機能装置の間の共用境界。 この概念には、異なる機能を持つ 2 つの装置の接続を指定するという目的が含まれる。 (T) (2) システム、プログラム、または装置のリンクを行うハードウェアまたはソフトウェア、あるいはその両方。

中間セッション経路指定 (intermediate session routing (ISR)). APPN ネットワーク・ノード内の経路指定機能のタイプの 1 つで、これは、そのノードを通過するがエンド・ポイントはどこか別のところであるすべてのセッションについて、セッション・レベルのフロー制御と故障レポートを提供する。

I/O. 入出力 (input/output)。

IPR. 分離ペーシング応答 (Isolated pacing response)。

J

JISCII. 情報交換用日本工業規格コード。日本において、IBM パーソナル・コンピューターによって使用されるコード。それは、日本工業規格 (JIS) C 6226 「情報交換用漢字コード」(JIS 漢字セット) とユーザーの特殊文字を含んでいる。

L

LAN. ローカル・エリア・ネットワーク (Local area network)。

層 (layer). (1) ネットワーク・アーキテクチャーにおいて、概念的な見地からは完結しているサービス・グループであって、そのグループは階層的に配列された一群のグループの中の 1 つであり、またそのグループはネットワーク・アーキテクチャーに準拠しているすべてのシステムへ拡張されている。 (T)(2) 開放型システム間参照モデルにおいて、概念的に完結し階層的に配列されたサービス、機能、およびプロトコルの 7 つのグループの中の 1 つであって、すべての開放型システムへ拡張されるもの。 (T)(3) SNA において、関連のある機能をグループにしたもので、このグループは他のグループの中の機能とは論理的に別個のものである。 1 つの層の中での機能の実装は、他の層の機能に影響を与えずに変更することができる。

LEN. ローエントリー・ネットワーキング (low-entry networking)。

回線 (line）. (1) データ回線終端装置 (DCE) の外部のデータ回線にあって、DCE をデータ交換装置 (DSE) に接続するか、DCE を他の 1 つまたは複数の DCE に接続するか、または DSE を他の DSE に接続する部分。 (I) (2) *チャネル (channel)* および*回線 (circuit)* と同義。

回線制御規則 (line control discipline). *リンク・プロトコル (link protocol)* および *プロトコル (protocol)* の同義語。

伝送制御手順 (line discipline). *リンク・プロトコル(link protocol)* および *プロトコル (protocol)* の同義語。

リンク、連係 (link). (1) リンク接続機構（伝送媒体）と、その両端にある 2 つのリンク・ステーション。 分岐構成またはトークンリング構成では、複数のリンクが 1 つのリンク接続機構を共用することができる。(2) データ項目や、 1 つまたは複数のコンピューター・プログラムの部分を相互に接続すること。たとえば、リンケージ・エディターによるオブジェクト・プログラムのリンク、またはポインターによるデータ項目のリンク (T) (3) SNA において、*データ・リンク (data link)*と同義。

リンク接続 (link-attached). (1) データ・リンクにより制御装置に接続されている装置のことを指す。(2) *チャネル接続 (channel-attached)* と対比。(3) *リモート (remote)* と同義。

リンク接続機構 (link connection). (1) 1 つのリンク・ステーションと他のリンク・ステーションとの間で両方向の通信を提供する物理装置。たとえば、通信回線やデータ回線終端装置 (DCE) がある。(2) SNA においては、*データ回線 (data circuit)*と同義。

リンク接続セグメント (link connection segment). 構成の一部分で、この部分は、サービス・ポイント・コマンド・サービス(SPCS) 照会リンク構成要求リストの中で連続的にリストされている2つの資源の間に置かれている。

リンク・プロトコル (link protocol). (1) リンク・レベルでデータを送受信するための規則。(2) *回線制御規則(line control discipline)* および *伝送制御手順 (line discipline)* と同義。

リンク状況 (link status (LS)). ローカル・モデムおよびリモート・モデムが維持する情報。

ロードする (load). (1) コンピューター・プログラムの全体または一部を補助記憶装置からメモリーに入れ、コンピューターがそのプログラムを実行できるようにすること。(2) ディスケットをディスケット・ドライブに入れること。

ローカル (local). (1) 通信回線を使用しないで直接アクセスされる装置の状態。(2) *リモート (remote)* と対比。(3) *チャネル接続 (channel-attached)* の同義語。

ローカル・エリア・ネットワーク (local area network (LAN)). (1) 限られた地域内のユーザーの構内にあるコンピューター・ネットワーク。 ローカル・エリア・ネットワーク内での通信は、外部の規則により規制されない。しかし、LAN の境界を越えて通信を行う場合は、何らかの規制を受ける場合がある。 (T) (2) 一組の装置が相互に通信するように接続されているネットワークで、それよりも大きなネットワークへ接続できるもの。(3) *イーサネット (Ethernet)*および*トークンリング (token ring)*も参照。(4) *大都市圏ネットワーク (metropolitan area network (MAN))* および *広域ネットワーク (wide area network (WAN))* と対比。

ローカル・ディレクトリー・データベース (local directory database). ネットワーク内において特定のノードにあるものとして知られている一組の資源 (LU)。この資源には、ノード定義域にあるすべての資源のほか、キャッシュ・エントリーも含まれる。

ローカル LU (local LU). LAN 上に分散されてはいないが、ゲートウェイ・パーソナル・コンピューターによって制御される論理装置。通常、これはワークステーション、プリンター、または端末のような物理装置である。

ローカル・セッション識別 (local session identification (LSID)). SNA における FID3 (フォーマット識別タイプ3) 伝送ヘッダーの中にある 1 つのフィールドであって、セッションの種類 (SSCP-PU、SSCP-LU、または LU-LU) の表示と、周辺論理装置 (LU) または物理装置 (PU) のローカル・アドレスを含んでいるもの。

ログオン操作員 (logged-on operator). (1) 端末およびログオン・ユーザーを必要とする NetView 操作員ステーション・タスク。(2) *自動タスク (autotask)* と対比。

論理リンク (logical link). (1) それぞれ 2 つの隣接ノード内にある一対のリンク・ステーションと、それらの基礎となっているリンク接続機構で、その 2 つのノード間の単一リンク層接続を提供する。 複数の論理リンクが2つのノードに接続する同じ物理媒体の使用を共用していても、それぞれのリンクを区別できる。 例としては、ローカル・エリア・ネットワーク (LAN) 機能で使用される 802.2 論理リンク、および、2 つのノード間の同じ 2 地点間物理リンク接続上の LAP E 論理リンクなどがある。論理リンクという用語には、DTE から X.25 ネットワークへのアクセス・リンクの使用を共用する複数の X.25 論理チャネルも含まれる。(2) APPNTAM において、 1 つのノードでのリンクの単一方向表現。

論理レコード (logical record). 論理的な観点から見て１つのレコードとみなされる、関連データまたは語の集合。 (T)

論理端末 (logical terminal). (1) １つまたは複数の物理端末に関連付けられた名前の付いた宛先。(2) 特定の3270 または 5250 エミュレーション・セッションの定義。

論理装置 (logical unit (LU)). ネットワーク・アクセス可能単位のタイプの１つ。ユーザーはこれを使用して、ネットワーク資源にアクセスしたり互いに通信したりする。

作業論理単位 (logical unit of work (LUW)). 同期点操作の結果として行われ、１単位としてコミットまたはバックアウトされる、保護資源に対する変更。対象の保護資源は、会話によって結合しているいくつかの異なる LU に分散していてもよい。

論理エラー (logic error). VTAM において、無効な要求が原因で生じるエラー条件 (プログラム論理エラー)。

ログオン (logon). (1) ユーザーが端末セッションを開始する手順。(2) 定義域にアクセスしてユーザー ID を引き継ぐプロセス。

ログオン・モード (logon mode). VTAM において、論理装置との通信用としてログオン・モード・テーブル内に指定されているセッション・パラメーターのサブセット。セッション・パラメーター *(session parameters)* も参照。

ローエントリー・ネットワーキング (low-entry networking (LEN)). 基本対等プロトコルを使用してノード同士が直接接続して、論理装置間の複数並列セッションをサポートする能力。

ローエントリー・ネットワーキング (LEN) エンド・ノード (low-entry networking (LEN) end node). 隣接 APPN ネットワーク・ノードからネットワーク・サービスを受け取る LEN ノード。

ローエントリー・ネットワーキング (LEN) ノード (low-entry networking (LEN) node). 一連のエンド・ユーザー・サービスを提供し、対等プロトコルを使用して他のノードに直接接続し、そして、隣接APPNネットワーク・ノードから暗黙的に（つまり CP-CP セッションを直接使用しないで）ネットワーク・サービスを引き出すノード。

LS. リンク状況 (Link status)。

LSID. 論理セッション識別子 (local session identification)。

LU. 論理装置 (logical unit)。

LU アドレス (LU address). パーソナル・コミュニケーションズにおいて、SNA セッションでリモート・パートナーを表すためにホストによって割り当てられる 02 から 254 までの数。

LU-LU セッション (LU-LU session). SNA ネットワーク内の 2 つの論理装置 (LU) 間の論理接続で、一般に2人のユーザー間の通信を提供する。

LUS. 論理装置サービス (Logical unit services)。

LU タイプ(LU type). 特定のセッションについて各 LU がサポートする SNA プロトコルおよびオプションのサブセットを基準とした場合の LU の種別。基準には次のものがある。

- セッション活動化要求で使用できる必須値および任意選択値
- データ・ストリーム制御、機能管理ヘッダー(FMH)、要求単位パラメーター、およびセンス・データ値の用途
- 表示サービス・プロトコル（FMH の用途に関連したものなど）

LU タイプ０、１、２、３、４、6.1、6.2、および７が定義されている。

LUW. 論理作業単位 (Logical unit of work)。

LU 2. アプリケーション・プログラム用の LU の１つのタイプで、このタイプの LU は、SNA 3270 データ・ストリームを使用して、対話型環境で単一の表示ワークステーションと通信する。

LU 3. アプリケーション・プログラム用の LU の１つのタイプで、このタイプの LU は、SNA 3270 データ・ストリームを使用して、単一のプリンターと通信する。

LU 6.2. (1) 論理装置のタイプの１つ。分散処理環境でのプログラム間の一般的な通信をサポートする。 LU 6.2 には以下の特徴がある。(a) セッション・パートナー間の対等関係。(b) １つのセッションを複数トランザクション用に効率使用。(c) 広範な終端間エラー処理。(d) 製品の実装に組み込まれている構造化された verb から成る汎用アプリケーション・プログラミング・インターフェース (API)。(2) 分散データ処理環境において、SNA の一般データ・ストリームである構造化フィールド・データ・

ストリームか、ユーザー定義のデータ・ストリームを使って 2 つのアプリケーションの間のセッションをサポートするタイプの LU。

LU 6.2 verb. LU 6.2 アプリケーション・プログラミング・インターフェースにおいて 1 つの命令を表す構文単位。

M

マクロ命令 (macroinstruction). (1) ソース言語の中にあって、同じソース言語内の定義済みの命令順序で置き換えられる命令。この命令では、置換後のパラメーターの値も指定できる。 (T) (2) アセンブラー・プログラミングにおいて、マクロ定義と呼ばれる事前定義のステートメント・セットをアセンブラーに処理させるアセンブラー言語ステートメント。プログラム内のマクロ命令は、通常マクロ定義から作られるステートメントで置き換えられる。(3) *定義ステートメント (definition statement)* も参照。

保守サービス (maintenance services). SNA において、システム・サービス制御点(SSCP) および物理装置 (PU) で行われるネットワーク・サービスの一種。 保守サービスは、リンクとノードをテストするための機能、およびエラー情報を収集し記録するための機能を提供する。

MAN. 大都市圏ネットワーク (metropolitan area network)。

管理サービス (management services (MS)). (1) 制御点 (CP) および物理装置 (PU) の中のネットワーク・サービスのタイプの 1 つ。 管理サービスは、SNA ネットワークの管理を支援する目的で提供されているサービスで、問題管理、パフォーマンスおよび会計管理、構成管理、変更管理などがある。(2) システムおよびネットワークの管理を支援するサービス。対象分野には、問題管理、パフォーマンス管理、ビジネス管理、運用管理、構成管理、変更管理などがある。

マネージャー（manager）. (1) システム管理において、特定の対話に関してマネージャーの役割を果たすものとみなされるユーザー。(2) (a) オブジェクトに関する通知を受け取り、(b) オブジェクトを修正または照会するための管理操作を要求することによって、 1 つまたは複数の管理オブジェクトをモニターまたは制御するエンティティー。(3) マネージャーの役割を果たすものとみなされるシステム。

マップ式会話（mapped conversation）. 割振りトランザクション・プログラムが指定する LU 6.2 会話タイプ。マップ式会話を使用するトランザクション・プログラムは、基礎となっているデータ・ストリームに関係なく、任意の形式でメッセージを交換することができる。 システム定義またはユーザー定義のマッパーは、トランザクション・プログラムに関するデータ形式変更を行うことができる。 *会話 (conversation)* も参照。 *基本会話（basic conversation）* と対比。

マッピング（mapping）. 送信側がある形式で伝送したデータを、受信側が受け入れることのできるデータ形式に変換するプロセス。

突合せ (matching). MPTN アーキテクチャーにおいて、同じユーザー・プロトコルまたは同じトランスポート・プロトコルを使用する対等トランスポート・ユーザーまたは対等トランスポート提供者の間の関係を表す。

MB. メガバイト（megabyte）。

メガバイト (megabyte (MB)). (1) 主記憶装置、実記憶装置および仮想記憶装置、およびチャネル・ボリュームでは、220 または 1 048 576 バイト。(2) ディスク記憶装置の容量およびコミュニケーション・ボリュームでは、1 000 000バイト。

メモリー (memory). 処理装置またはその他の内部記憶装置内にあって、命令の実行に使用されるすべてのアドレス可能記憶域スペース。 (T)

メニュー (menu). (1) データ処理システムがユーザーに対して表示するオプションのリスト。このリストから、ユーザーは開始するアクションを選択できる。 (T) (2) テキスト処理において、テキスト・プロセッサーがユーザーに対して表示する選択項目のリスト。このリストから、ユーザーは開始するアクションを選択できる。 (T) (3) オブジェクトへ適用できる選択項目のリスト。メニューには、特定のコンテキストでは選択できない選択項目が含まれていることもある。 このような選択項目は弱いコントラストで表示される。 (T)

メッセージ (message). (1) 発信元から 1 つまたは複数の受信側に 1 つのエンティティーとして転送される、いくつかの文字と場合によっては制御コードの集合。 メッセージはエンベロープと内容の 2 つの部分から成っている。 (T) (2) 人またはプログラムから他の人またはプログラムへ送る通信。

メッセージ処理機能 (message processing facility (MPF)). メッセージ表示とメッセージ処理を制御する MVS 機能。

メッセージ待ち行列 (message queue). IMS/VS において、アプリケーション・プログラムによる処理の前または端末への送信の前に、待機のためにメッセージが入れられるデータ・セット。

メトリック (metric). インターネット通信において、同じ自律システムへの複数の出口点またはエントリー・ポイントを区別できるようにするために経路に関連付けられている値。最もメトリックの小さい経路が優先使用される。

大都市圏ネットワーク (metropolitan area network (MAN)). 複数のネットワークを相互接続して形成されるネットワークで、このネットワークは、構成単位の各ネットワークよりも高速で動作し、行政地区境界を越え、複数のアクセス方式を使用することができる。 (T) *ローカル・エリア・ネットワーク (local area network(LAN))*および*広域エリア・ネットワーク (wide area network (WAN))* と対比。

MID. マシン識別子 (Machine identifier)。

移行 (migration). プログラムの新しいバージョンまたはリリースを、旧バージョンまたはリリースと取り替える形で導入すること。

モード (mode). *モード名 (mode name)* を参照。

モデム（変復調装置）(modem (modulator/demodulator)). (1) 信号を変調および復調する機能装置。 モデムの主要機能の 1 つは、アナログ伝送機能を介してディジタル・データを伝送できるようにすることである。 (T) (A)(2) コンピューターからのディジタル・データを、通信回線上に伝送できるアナログ信号へ変換し、受け取られたアナログ信号をコンピューター用のデータへ変換する装置。

モード名 (mode name). セッションの起動側がセッションに必要な特性を指定するための名前。トランスポート・ネットワーク内の通信量のペーシング値、メッセージ長の限度、同期点と暗号のオプション、サービスのクラスなどがある。

モジュール (module). コンパイル、他の単位との結合、およびロードという点から見て、離散的で識別可能なプログラム単位。たとえば、アセンブラー、コンパイラー、リンケージ・エディター、またはエグゼクティブ・ルーチンへの入力、またはそれからの出力など。

モニター（する）(monitor). (1) データ処理システム内での選択された活動を、分析のために観察し記録する装置。標準からの大幅な逸脱を示したり、特定の機能単位の利用レベルを判別したりするなどの用途が考えられる。 (T) (2) システムの動作を観察、監視、制御、または検証するソフトウェアまたはハードウェア。 (A) (3) リング上でトークンの伝送を開始し、トークンの消失、フレーム循環、その他の問題に際してソフトエラー回復を行うために必要な機能。この能力はすべてのリング・ステーションにある。(4) NetView グラフィック監視機能において、NetView プログラムからの状況変更を受け入れることができるビューをオープンすること。このビューから直接、問題の判別と訂正ができる。*参照 (browse)* と対比。

マウス（mouse）. 最もよく使われる指示装置の 1 つ。1 つまたは複数のボタンがあり、ユーザーはこれを用いて、製品や操作環境と対話することができる。

MPF. メッセージ処理機能 (Message processing facility)を参照。

MS. 管理サービス (management services)。

MSG. コンソール・メッセージ (Console messages)。

N

NAU (NAU). (1) ネットワーク・アクセス可能単位 (network accessible unit)。(2) ネットワーク・アドレス可能装置(Network addressable unit)。

NC. ネットワーク制御 (Network control) を参照。

NCP/EP 定義機能 (NCP/EP definition facility (NDF)). システム・サポート・プログラム (SSP) の一部となっているプログラムで、組込みエミュレーション・プログラム (PEP)、ネットワーク制御プログラム (NCP)、またはエミュレーション・プログラム (EP) 用のロード・モジュールを生成するために使用される。

NDF. NCP/EP 定義機能 (NCP/EP definition facility) を参照。

否定応答 (negative response (NR)). SNA において、要求が正常に到着しなかったこと、または受信側により正常に処理されなかったことを示す応答。 *肯定応答 (positive response)* と対比。

折衝可能 BIND (negotiable BIND). SNA において、2 つのハーフセッションが、セッションを活動化する時点でそのセッションのパラメーターについて折衝することのできる能力。

折衝 (negotiation). ネットワークと 3710 ネットワーク制御装置との間で伝送されるパケット・サイズを決定するプロセス。

NETID. *ネットワーク識別子 (network identifier)* を参照。

NetView 間タスク (NetView-NetView task (NNT)). 定義域間 NetView 操作員セッションが実行されるタスク。*操作員ステーション・タスク (operator station task)* を参照。

ネットワーク (network). (1) ノードおよび接続分岐を配置したもの。 (T) (2) 情報交換のために接続されたデータ処理装置とソフトウェアとの構成。(3) ノードのグループおよびそれらのノードを相互に接続するリンク。

ネットワーク・アクセス可能単位 (Network accessible unit (NAU)). 論理装置 (LU)、物理装置 (PU)、制御点 (CP)、またはシステム・サービス制御点 (SSCP)。 これは、パス制御ネットワークによって送信される情報の発信元または宛先である。 *ネットワーク・アドレス可能単位 (network addressable unit)* と同義。

ネットワーク・アドレス（network address）. (1) ISO 7498-3 によれば、OSI 環境の中で、一組のネットワーク・サービス・アクセス点を明確に識別する名前。(2) サブエリアと要素フィールドから成るアドレスであって、リンク、リンク・ステーション、またはネットワーク・アドレス可能単位を識別するもの。サブエリアはネットワーク・アドレスを使用し、周辺ノードはローカル・アドレスを使用する。(3) SNA において、サブエリアおよび要素フィールドから成るアドレスで、このアドレスは、リンク、ゲートウェイ、またはネットワーク・アドレス可能単位 (NAU) を識別する。(4) サブエリア・ネットワークにおいて、サブエリア・フィールドと要素フィールドから成るアドレスで、リンク、リンク・ステーション、物理装置、論理装置、またはシステム・サービス制御点を識別する。 サブエリア・ノードはネットワーク・アドレスを使用し、周辺ノードは、ローカル・アドレスまたはローカル形式セッション識別子 (LFSID) を使用する。周辺ノードが接続されているサブエリア・ノード内の境界機能は、ローカル・アドレスまたは LFSID をネットワーク・アドレスに、またはその逆に形式変更する。*ネットワーク名 (network name)* と対比。

ネットワーク・アドレス可能単位 (Network addressable unit (NAU)). *ネットワーク・アクセス可能単位（network accessible unit)* の同義語。

ネットワーク・アーキテクチャー (network architecture). コンピューター・ネットワークの論理構造および動作原理。 (T)

注: ネットワークの動作原理には、サービス、機能、およびプロトコルの動作原理も含まれる。

ネットワーク過密 (network congestion). ネットワークの処理能力を超えたトラフィックによって引き起こされる、望ましくない過負荷状態。

ネットワーク制御 (network control (NC)). SNA において、明示経路および仮想経路の活動化と非活動化、そして周辺ノードの調整のためのロード・モジュールの送信などを目的として、物理装置 (PU) 間で交換される要求および応答に使用する要求応答単位 (RU) カテゴリー。*データ・フロー制御 (data flow control)* 、*機能管理データ (function management data)* 、および*セッション制御 (session control)* も参照。

ネットワーク・ディレクトリー・データベース (network directory database). *分散ディレクトリー・データベース (distributed directory database)* の同義語。

ネットワーク識別子 (network identifier). (1) TCP/IP において、ネットワークを定義するIP アドレス部分。 ネットワーク ID の長さは、ネットワーク・クラスのタイプ（A、B、またはC）によって異なる。(2) 特定のサブネットワークを固有のものとして識別する 1 ～ 8 バイトのユーザー選択名または 8 バイトの IBM 登録名。(3) MPTN アーキテクチャーにおいて、トランスポート提供者アドレスのアドレス修飾子。ノードのグループが属しているネットワークに従って、そのグループを識別する。

ネットワーク名 (network name). ユーザーが、特定のサブネットワーク内のネットワーク・アクセス可能単位、リンク、またはリンク・ステーションを参照するために使用できる記号識別子。 APPN ネットワークでは、ネットワーク名は経路指定の目的にも使用される。 *ネットワーク・アドレス (network address)* と対比。

ネットワーク・ノード定義域 (network node domain). 1 つの APPN ネットワーク・ノード制御点、それに接続しているリンク、その制御点がディレクトリー検索要求に対する応答を受け持つネットワーク資源（つまりその制御点のローカルLUおよび隣接LENエンド・ノード）、その制御点がディレクトリー検索要求および応答を交換する隣接 APPN エンド・ノード、および、その制御点自身のノードまたはそれが管理サービスを提供する隣接エンド・ノードに関連したその他の資源（ローカル記憶装置など）。

ネットワーク操作員 (network operator). (1) ネットワークの全体または一部の働きを制御する人。(2) 複数定

義域ネットワークにおいて、すべての定義域の制御に責任を負う人またはプログラム。(3) *定義域操作員 (domain operator)* も参照。

ネットワーク修飾名 (network-qualified name). SNA において、特定のネットワーク内で特定の資源（LU またはCP など）を固有のものとして識別する名前。 この名前は、ネットワーク識別子と資源名から成るもので、そのおのおのがそれぞれ 1 ～ 8 バイトの記号ストリングである。*完全修飾名 (fully qualified name)* と同義。

ネットワーク・サービス (network services). (1) ネットワーク・アクセス可能単位内あって、SSCP-SSCP、SSCP-PU、SSCP-LU、および CP-CP セッションを介したネットワークの動作を制御するサービス。(2) APPN ネットワーク・ノード制御点がその定義域に提供するセッション・サービス（ディレクトリーおよび経路選択機能）および管理サービス。

ノード (node). (1) ネットワークにおいて、 1 つまたは複数の機能単位がチャネルまたはデータ回線を接続するポイント。 (I) (2) ネットワークに接続されていて、データの送受信を行う任意の装置。(3) 1 つのリンクの端点、またはネットワーク内の複数のリンクに共通する接合点。 ノードには、プロセッサー、通信制御装置、クラスター制御装置、端末などがある。ノードによって、経路指定などの機能が異なる場合がある。

ノード名 (node name). VTAM において、ネットワーク定義時に特定の大ノードまたは小ノードに割り当てられる記号名。

非ブロッキング・モード (nonblocking mode). (1) インターフェースを介してサービスを要求する方法の 1 つ。要求がただちに完了できない場合、要求元のプロセスは続行可能であり、延期はされない。(2) *ブロッキング・モード (blocking mode)* と対比。

無応答 (no response). SNA において、要求の受信側に、要求が正常に受信され処理されたかどうかに関係なく応答を一切戻さないように指示するために、要求ヘッダーの応答形式要求フィールドで要求するプロトコル。*確定応答 (definite response)* および *例外応答 (exception response)* と対比。

通常フロー (normal flow). SNA において、伝送ヘッダー (TH) の中で指定するデータ・フローの 1 つで、主としてエンド・ユーザー・データの伝達に使用される。通常フローにおける要求の流れの速度は、セッション・レベル・ペーシングにより規定できる。通常フローも急送フローも、 1 次から 2 次へ、および 2 次から 1 次へと両方向に流れる。*急送フロー (expedited flow)* と対比。

通知 (notification). 発生したイベントについて、偶発的かつ自動的に生成されるレポート。

NOTIFY. システム・サービス制御点 (SSCP) が、論理装置 (LU) が要求したプロシージャーの状況を知らせるために、その LU に送るネットワーク・サービス要求。

O

OAF. 起点アドレス・フィールド (origin address field)。

OAF. 起点アドレス・フィールド・プライム記号 (Origin address field prime)。

オブジェクト (object). (1) オブジェクト指向の設計およびプログラミングにおいて、データおよびそのデータに関連した操作から成る概念。*クラス (class)* も参照。(2) タスクを実行するためにユーザーが 1 単位として取り扱うことができる項目。オブジェクトは、テキスト、アイコン、またはその両方の形式で表示できる。

ODAI. 起点宛先割当て標識。FID2 伝送ヘッダー中にあり、アドレス空間を分割するために使用されるビット。この分割方式では、あるノード内のアドレス空間マネージャー (ASM) はある設定の ODAI で可能な OAF′、DAF′ のすべての組合せを使用でき、隣接ノードの ASM は補完設定の ODAI で可能な OAF′、DAF′ のすべての組合せを使用できる。

オフ (Off). 最新表示選択項目からのカスケード・メニューに現れる選択項目の 1 つ。これは最新表示機能をオフに設定する。

オフライン (offline). (1) コンピューターの主な操作から独立して、またはそれと並行して実行される機能単位の操作に関する表現。 (T)(2) コンピューターから制御されないだけでなく、また、コンピューターと通信していない状態。 *オンライン (online)* と対比。

オフセット (offset). レコード、区域、または制御ブロックの中の任意の開始点から、ある他の点までを測定した場合の測定単位数。

OIA. 操作員情報域 (operator information area)。

OK. ウィンドウ内の情報を受け入れてそのウィンドウをクローズするための押しボタン。ウィンドウ内に変更された情報が含まれている場合は、ウィンドウをクローズするまでそれらの変更が適用される。

オン (On). 最新表示選択項目からのカスケード・メニューに現れる選択項目の１つ。ウィンドウ内のビューがただちに最新表示される。

片方向ブラケット (one-way bracket). このブラケット内のデータは、開始ブラケットと条件付き終了ブラケットを備えた例外応答要求付きの単一連鎖として、NAU から別の NAU に送られる。CP-CP セッションで片方向ブラケットを使用する場合は、ブラケットは常に競合勝者セッションで送られる。

オンライン (online). (1) コンピューターの直接制御下にある機能単位の操作に関する用語。 (T)(2) ユーザーがコンピューターと対話できる状態を示す用語。 (A)(3) ユーザーが端末を介してコンピューターへアクセスできる状態を示す用語。 (A)(4) コンピューターによって制御されるか、コンピューターと通信している状態。(5) *オフライン (offline)* と対比。

オープン（する）(open). (1) 電気回路におけるブレーク。(2) アダプターを使用可能状態にすること。

オープン (open). ユーザーがオープンしたいオブジェクトを選択するためのウィンドウを表示する選択項目。

オペレーティング・システム (operating system (OS)). プログラムの実行を制御するソフトウェアであって、資源の割振り、スケジューリング、入出力制御、およびデータ管理などのサービスを提供する場合がある。オペレーティング・システムは原則としてソフトウェアであるが、部分的にハードウェア実装も可能である。 (T)

操作（operation）. オブジェクト指向の設計またはプログラミングにおいて、オブジェクトの境界で要求できるサービス。操作には、オブジェクトの変更や、オブジェクトに関する情報の表示などがある。

演算子、操作員（operator）. (1) 言語ステートメントにおいて、オペランドに対して行うアクションを示す字句エンティティー。定義ステートメント *(definition statement)* も参照。(2) MVS、NetView プログラム、IMS など、特定のソフトウェアが制御する活動を管理する責任を負う人またはプログラム。(3) 装置を操作する人。(4) システムの実行を維持する人。(5) *自動タスク (autotask)* 、*ログオン操作員 (logged-on operator)* 、*ネットワーク操作員 (network operator)* 、および*操作員ステーション・タスク (operator station task)* も参照。

操作員情報域 (operator information area (OIA)). 表示域の下部にある区域であって、その区域に、端末またはシステムの状況情報が表示される。

操作員ステーション・タスク (operator station task (OST)). ネットワーク操作員とのオンライン・セッションを確立し維持する NetView タスク。NetView プログラムにログオンする各ネットワーク操作員について、操作員ステーション・タスクが１つずつある。*NetView 間タスク (NetView-NetView task)* を参照。

オプション・セット (option set). (1) 特定のアーキテクチャーを実装する製品がサポートしている機能の集合。１つの製品が、多数のオプション・セットをサポートすることもあり、まったくサポートしないこともある。サポートされる各オプション・セット内のすべての機能がサポートされる。(2) *基本セット (base set)* と対比。

発信元 (origin). メッセージまたはその他のデータを発行する外部論理装置 (LU) またはアプリケーション・プログラム。*宛先 (destination)* も参照。

起点アドレス・フィールド (origin address field (OAF)). SNA における FID0 または FID1 伝送ヘッダーの中にある１つのフィールドで、そのフィールドの中には、発信ネットワーク・アクセス可能単位 (NAU) のアドレスがある。 *宛先アドレス・フィールド (destination address field (DAF))* と対比。 *形式識別 (format identification (FID))フィールド* および*ローカル・セッション 識別 (local session identification (LSID))* も参照。

OS. オペレーティング・システム (Operating System) を参照。

アウトバウンド (outbound). 通信において、ネットワークに伝送されるデータ。

出力 (output). 出力プロセスに関与する装置、プロセス、またはチャネル、あるいはそれらに関連したデータまたは状況を表す。コンテキスト上明らかな場合は、「出力データ」、「出力シグナル」、「出力プロセス」の代わりに単に「出力」が使用されることがある。

オーバーレー(overlay). 行、陰影、テキスト、ボックス、またはロゴなど、印刷時にページ上の可変データと組み合わせて使用できる事前定義データの集合。

P

ペーシング. (1) 受信側の構成要素が、オーバーランや輻輳（ふくそう）を防ぐために送信側構成要素の伝送速度を制御する方法。(2) *受信ペーシング (receive pacing)* 、*送信ペーシング (send pacing)* 、セッションレベル・ペーシ

ング (session-level pacing) 、および*仮想経路 (VR) ペーシング (virtual route (VR) pacing)* を参照。(3) *フロー制御 (flow control)* も参照。

ペーシング・グループ (pacing group). *ペーシング (pacing window)* の同義語。

ペーシング応答 (pacing response). SNA において、受信コンポーネントが他のペーシング・グループを受け入れる準備ができたことを知らせる標識。 この標識は、セッション・レベル・ペーシングの場合は応答ヘッダー (RH) に入れ、仮想経路ペーシングの場合は伝送ヘッダー (TH) に入れて送られる。

ペーシング (pacing window). (1) 仮想経路ペーシング応答を受信する前に仮想経路上を伝送できるパス情報単位 (PIU)。仮想経路の受信側が、経路上の後続の PIU を受信できる状態にあることを示す。(2) セッション・レベル・ペーシング応答を受信する前に、セッションの通常フローで一方向に伝送できる要求。受信側が、次の要求グループを受け入れることのできる状態にあることを示す。(3) *ペーシング・グループ (pacing group)* の同義語。

パケット (packet). (1) データ通信における 2 進数の列。この列はデータと制御信号を含み、合成された全体として伝送され交換される。 データ、制御信号、また場合によってはエラー制御情報が特定のフォーマットで配列されている。 (I) (2) アドレス情報および順序情報を含む明確な情報単位。この情報単位は同一および他の伝送順序のパケットとインターリーブして送ることができる。X.25 プロトコルのようなパケット交換プロトコルによって送られたデータは、PSDN 上では必ずしも送られた順序で到着するわけではなく、また同じ経路指定によって到着するわけでもない。

パケット組立て/分解機能 (packet assembler/disassembler (PAD)). (1) パケット交換の機能を備えていないデータ端末装置 (DTE) がパケット交換ネットワークにアクセスできるようにする機能単位。 (T) (A) (2) データを伝送用パケットにコード化し、受け取られたパケットを元のデータ・フォーマットへ組み立てるプログラム。*パケット (packet)*、*パケット交換データ・ネットワーク (packet-switching data network)*も参照。

パケット交換 (packet switching). アドレスされたパケットを用いてデータを経路指定および伝送し、1 つのチャネルがパケットの伝送中にのみ占有されるようにするプロセス。 伝送が完了すると、チャネルは他のパケットの転送に使用できるようになる。 (I)

パケット交換データ・ネットワーク (packet switching data network (PSDN)). (1) データの伝送手段としてパケット交換を使用するネットワーク。(2) 情報のパケットを使用して通信するホスト・システムとネットワーク・ステーションとから構成されるデータ伝送システム。データは、情報のフローを分散しネットワークのデータ伝送能力を最大にするために、インターリーブされたバケットとして送られる。X.25 はパケット交換データ・ネットワーク通信プロトコルである。*X.25 ネットワーク (X.25 network)*を参照。

PAD. パケット組立て/分解機能 (Packet assembler/disassembler) を参照。

page. (1) 仮想記憶装置において、仮想アドレスを有し実記憶装置と補助記憶装置との間で 1 つの単位として転送される固定長ブロック。 (I) (A) (2) 表示装置の画面上で同時に表示される情報。(3) 画面上で表示された情報を、同じファイルの先行する情報または後続の情報と置換すること。

並行、並列 (parallel). (1) すべてのイベントが同じ時間間隔の中で起こり、各イベントが、別々ではあるが類似している機能単位によって処理されるプロセスに関する用語。例として、コンピューター・ワードのビットが内蔵バスの線に沿って転送される並列転送がある。 (T) (2) 複数の装置の同時的な動作または単一の装置における複数の活動の同時的実行に関する用語。 (A) (3) 複数の装置またはチャネルにおける複数の関連した活動の同時的発生に関する用語。 (A) (4) 複数のプロセスの同時性に関する用語。 (A) (5) 全体の中の個々の部分、たとえば文字の中の各ビット、ワードの中の各文字を、各部分のために別個の手段を使用して同時に処理することに関する用語。 (A) (6) *逐次*、*直列 (serial)* と対比。

並列セッション (parallel sessions). 使用しているネットワーク・アドレスまたはローカル形式セッション識別子ペアが異なる 2 つの同一ネットワーク・アクセス可能単位 (NAU) の間で、同時に活動状態にある複数のセッション。 各セッションがそれぞれ独自のセッション・パラメーターを持つことができる。

パラメーター (parameter). (1) 指定されたアプリケーションに定数値を与える変数。また、そのアプリケーションを表示するもの。 (I) (A) (2) 基本 CUA アーキテクチャーにおいて、コマンドと共に使用されてコマンドの結果に影響を与える変数。(3) メニュー中の項目であって、ユーザーがその項目に対して値を指定するか、メニューが解釈されるときにシステムがその項目に対して値を提供するもの。(4) ユーザーまたはプログラムによっ

て他のプログラムまたはプロシージャーへ渡されるデータ。すなわち、言語ステートメント中のオペランド、メニュー中の項目、または共用のデータ構造体。

親 (parent). フォーク・プリミティブを使用して子プロセスを生成したプロセス。子 *(child)* と対比。

親プロセス (parent process). AIX および OS/2 オペレーティング・システムにおいて、他のプロセスを生成するプロセス。*子プロセス (child process)* と対比。

パスワード (password). (1) 認証のために使用される値、または特定の特権をもっている人々の集合の中でメンバーシップを確立するために使用される値。(2) コンピューター・システムおよびユーザーに知られている文字の独特のストリング。ユーザーは、システムおよびその中の情報へのアクセスを獲得するためには、その文字ストリングを指定する必要がある。(3) コンピューター・セキュリティーにおいて、コンピューター・システムとユーザーに知られた文字ストリング。ユーザーは、システムおよびその中に記憶されたデータへの無制限アクセスまたは制限アクセスを行うために、その文字ストリングを指定する必要がある。

パス (path). (1) ネットワークにおいて、2 つのノードの間の経路。パスには複数のブランチが含まれることもある。 (T) (2) 2 つのネットワーク・アクセス可能単位の間で交換する情報が通過する一連のトランスポート・ネットワーク構成要素（パス制御およびデータ・リンク制御）。*明示経路 (explicit route (ER))* 、*経路拡張機能 (route extension (REX))* 、および*仮想経路 (virtual route (VR))* も参照。

パス制御 (path control (PC)). ネットワーク内のネットワーク・アクセス可能単位間でメッセージを経路指定し、それらの単位間のパスを提供する機能。 この機能は、伝送制御からの基本情報単位 (BIU) を、多くの場合セグメント化によってパス情報単位 (PIU) に変換し、 1 つまたは複数の PIU を含む基本伝送単位をデータ・リンク制御との間で交換する。 パス制御はノード・タイプによって異なる。すなわち、あるノード(たとえば、APPN ノード) は経路指定のためにローカルで生成されたセッション識別子を使用し、他のノード (サブエリア・ノード) は経路指定のためにネットワーク・アドレスを使用する。

パス情報単位 (path information unit (PIU)). 伝送ヘッダー (TH) のみを含むメッセージ単位、または TH の次に基本情報単位 (BIU) または BIU セグメントが続いているメッセージ単位。

PC. (1) パーソナル・コンピューター (Personal computer (PC))。(2) パス制御 (path control)。(3) パーソナル・コミュニケーションズ (Personal Communications (PC))。

対等、対等機能 (peer). ネットワーク・アーキテクチャーにおいて、他のエンティティーと同じ層にある機能単位。 (T)

パフォーマンス・エラー (performance error). *一時エラー (temporary error)* の同義語。

周辺 LU (peripheral LU). SNA において、周辺ノード中の論理装置。 *サブエリア LU (subarea LU)* と対比。

周辺ノード（peripheral node）. 経路指定にローカル・アドレスを使用し、したがってネットワーク・アドレスの変更の影響を受けないノード。周辺ノードは、隣接サブエリア・ノードからの境界機能の援助を必要とする。周辺ノードは、サブエリア境界ノードに接続されているタイプ 1 、2.0、または 2.1 のノードである。

周辺 PU (peripheral PU). SNA において、周辺ノード中の物理装置。*サブエリア PU (subarea PU)* と対比。

永続エラー (permanent error). エラー回復プログラムでは解決できないエラー。*一時エラー (temporary error)* と対比。

相手固定接続 (permanent virtual circuit (PVC)). (1) X.25 およびフレーム・リレー通信において、各データ端末装置 (DTE) において論理チャネルが永続的に割り当てられている仮想回線。 呼出し確立プロトコルは必要でない。*相手選択接続、交換仮想回線 (switched virtual circuit (SVC))* と対比。(2) 2 つのフレーム・リレー終端装置ステーション間の直接的な、または 1 つまたは複数のフレーム・リレー・フレーム・ハンドラーを介した論理接続。PVCは 1 つまたは複数の PVC セグメントから成っている。(3) 専用回線電話リンクに割り当てられたセッション。

持続検査 (persistent verification). VTAM において、2 つの論理装置が、セッションで最初の会話を行うために相互の識別名を検査できるようにし、そのセッションを継続するために、2 つの論理装置が、以後の会話が正しいものと想定できるようするセキュリティー機能。

パーソナル・コミュニケーションズ製品のファミリー. 3270 および 5250 端末をエミュレートし、かつ OS/2、DOS、および Windows のようなオペレーティング・システムの上で実行される IBM ライセンス・プログラム。

パーソナル・コンピューター (personal computer (PC)). (1) 主として個人が独立して使用するためのマイクロコンピューター。 (T)(2) 通常、システム装置、表示モニター、キーボード、ディスケット・ドライブ、内蔵ハード・ディスク記憶装置、およびオプションのプリンターから構成されるデスクトップ型、床置き式、または携帯用マイクロコンピューター。PC は、主として単独使用を目的として設計されているが、メインフレームまたはネットワークにも接続できる。

物理回線 (physical circuit). 多重化なしに確立される回線。 *データ回線 (data circuit)* も参照。 *仮想回線 (virtual circuit)* と対比。

物理装置 (physical unit (PU)). (1) SSCP-PU セッションを介した SSCP からの要求に応じて、ノードに関連した資源（接続しているリンクおよび隣接リンク・ステーションなど）を管理しモニターする構成要素。 SSCP は、接続しているリンクなどのノード資源を、PU を介して間接的に管理するために、物理装置とのセッションを活動化する。 この用語は、タイプ 2.0、タイプ 4、およびタイプ 5 のノードのみに適用される。(2) *周辺 PU (peripheral PU)* およびサブエリア *PU (subarea PU)* も参照。

物理装置 (PU) サービス (physical unit (PU) services). SNA において、SSCP-PU セッション用の構成サービスおよび保守サービスを提供する、物理装置 (PU) 内の構成要素。

PIP. プログラム初期設定パラメーター (program initialization parameters)。

PLU. 1 次論理装置 (primary logical unit)。

ポインター (pointer). (1) データ要素の位置を示す他のデータ要素。 (T) (2) データ項目の位置を示す識別子。 (A)

ポップする （pop). プッシュダウン・リストの最上部から項目を削除すること。 *プッシュする (push)* と対比。

POP. ポスト・オフィス・プロトコル (Post Office Protocol) を算用。

肯定応答 (positive response). SNA において、要求が受け取られ処理されたことを示す応答。 *否定応答 (negative response)* と対比。

ポスト（post）. (1) レコードに一単位の情報を入力すること。(2) 事象の発生を通知すること。(3) レコードを最新状態に保つため、そのレコードに情報を付け加えること。

ポスト・オフィス・プロトコル (Post Office Protocol (POP)). ネットワーク・メールの交換およびメールボックスへのアクセスに使用するプロトコル。

Prepare. コミット処理の一環として送られ、パートナーが 2 フェーズ・コミット・プロセスの第 1 フェーズを開始したことを示す表示サービス・ヘッダー。

表示空間 ID (presentation space ID (PSID)). コミュニケーション・マネージャー/2 において、*短縮名 (short name)*の同義語。

1 次側ハーフセッション (primary half-session). SNA において、セッション活動化要求を送るハーフセッション。 *1 次論理装置 (primary logical unit (PLU))* も参照。 *2 次ハーフセッション (secondaryhalf-session)* と対比。

1 次論理装置 (primary logical unit (PLU)). SNA において、BIND を送ってパートナー LU とのセッションを活動化する論理装置 (LU)。 *2 次論理装置 (secondary half-session (SLU))* と対比。

問題判別 (problem determination). 問題の原因を決定するプロセス。たとえば、原因がプログラム構成要素であるか、マシン障害であるか、電気通信設備の故障であるか、ユーザーまたは契約会社が導入したプログラムまたは機器であるか、電源切断のような環境の障害であるか、またはユーザー・エラーであるかなどを決定する。

プロシージャー、手順 (procedure). (1) プログラム言語において、プロシージャー呼出しにより呼び出して実行する、形式パラメーター付きまたは形式パラメーターなしのブロック。 (I) (2) 問題の解決のために行う一連のアクションの記述。 (A)

処理する、プロセス（process）. (1) プロセスにおいてデータ操作を実行すること。 (I) (A) (2) データ処理において、プログラムの全体または一部分の実行中に起こる複数のイベントの推移。 (T) (3) 複数のイベントの推移であって、各イベントがその推移の目的または効果によって定義され, また所与の条件の下で達成されるもの。(4) データに対する 1 つの操作、または操作の組合わせ。(5) 実行されている機能、または実行を待機している機能。

プロセス識別番号 (process identification number (process ID)). オペレーティング・システムがプロセスに割り当てる固有の番号。 この番号は、プロセス間の通信のために内部的に使用される。

プロセッサー (processor). コンピューターにおいて、命令を解釈し実行する機能単位。プロセッサーは、少なくとも 1 つの命令制御装置および算術論理装置から成っている。 (T)

プロダクトセット識別 (product-set identification (PSID)). (1) SNA において、ネットワーク構成要素を実施しているハードウェア製品およびソフトウェア製品を識別する技法。(2) 定義内に記述された情報をトランスポートする管理サービス共通サブベクトル (1)。

プロファイル (profile). 単独のユーザー、ユーザーのグループ、または単独または複数のコンピューター資源の顕著な特性を記述するデータ。

プログラム (program). (1) コンピューターの処理に適した命令のシーケンス。 処理には、プログラムを実行することのほかに、アセンブラー、コンパイラー、インタープリター、または変換プログラムを使用して、プログラムを実行できるようにするための準備も含まれる。 (I) (2) プログラミング言語において、相互に関連付けられた 1 つまたは複数のモジュールの論理集合。 (I) (3) コンピューター・プログラムを設計、作成、およびテストすること。 (I) (A)

プログラム初期設定パラメーター (program initialization parameters (PIP)). ターゲット・プログラムへ入力として渡される最初のパラメーター値。または、プロセス環境をセットアップするために使用される最初のパラメーター値。

プロトコル（protocol）. (1) 通信を達成するうえで機能単位の行動を決定する意味的および構文的規則の集合。 (I) (2) 開放型システム間相互接続アーキテクチャーにおいて、通信機能を達成する場合に同じ層のエンティティーの行動を決定する意味的および構文的規則の集合。 (T) (3) SNA において、ネットワークの管理、データの伝送、およびネットワーク構成要素の状態の同期化に使用される要求と応答の意味と順序付け規則。*回線制御規則 (line control discipline)* および *伝送制御手順 (line discipline)* と同義。 *ブラケット・プロトコル (bracket protocol)* および*リンク・プロトコル (link protocol)* を参照。

PSID. 表示空間 ID (Presentation space ID) を参照。

PU. 物理装置 (physical unit)。

プッシュする (push). プッシュダウン・リストの最上部に項目を付け加えること。 ポップする *(pop)* と対比。

プッシュダウン・リスト (pushdown list). (1) 次に検索されるデータ要素が、最も最近記憶されたものになるようにするために、作成され維持されるリスト。 (T) (2) スタック *(stack)* と同義。

PUT. プログラム更新テープ (Program update tape)。

PVC. 相手固定回線接続 (permanent virtual circuit)。

Q

待ち行列、待ち行列化 (queue). (1) 検索されるべき次のデータ要素が最初に記憶されるように構成され保守されるリスト。 (T) (2) 処理を待機している項目の行またはリスト。たとえば、実行されるべき作業、表示されるべきメッセージ。(3) 待ち行列として配列すること、または、待ち行列を形成すること。

静止する (quiesce). 操作が正常に完了するようにしてプロセスを終了すること。

静止プロトコル (quiesce protocol). VTAM において、一度に 1 方向の通信を行う方式。1 次論理装置(PLU)または2 次論理装置 (SLU) のどちらか一方が、通常フロー要求を送る排他的権利を持ち、もう一方のノードはその種の要求を送らない。送信側が受信するときは、相手ノードを静止状態から解放する。

R

RAM. ランダム・アクセス・メモリー (random access memory)。 (A)

RAM セマフォー (RAM semaphore). OS/2 オペレーティング・システムにおいて、 1 つのプロセスの異なるスレッドを逐次化するために使用するセマフォー。 システム・セマフォー *(system semaphore)* と対比。

ランダム・アクセス・メモリー (random access memory (RAM)). 非順次方式でデータが入力され検索される記憶装置。

受信ペーシング (receive pacing). SNA において、構成要素が受け取っているメッセージ単位のペーシング。*送信ペーシング (send pacing)* と対比。

レコード (record). 1 つの単位として扱われるデータのセット。 (T)

再入可能 (reentrant). プログラムまたはルーチンの同じコピーが複数のタスクによって同時に使用できるようにするプログラムまたはルーチンの属性。

最新表示 (Refresh). ウィンドウ内のデータに加えた変更を、ただちに表示するのか、まったく表示しないのか、または後で表示するのかを制御するための他の選択項目 (*On* および *Off*) に、ユーザーがアクセスするためのカスケード選択項目。

解放（する）、リリース (release). (1) 新製品の配布、または既存製品に関する新機能およびAPAR修正の配布。通常は、新リリースの発行後一定期間が経過したあとは、旧リリースに関するプログラミング・サポートは廃止される。製品の最初のバージョンは、リリース 1 修正レベル 0 として発表される。(2) VTAM において、資源（通信制御装置または物理装置）の制御権を放棄すること。*資源引継ぎ (resource takeover)* も参照。*獲得（する）(acquire)* と対比。

リモート (remote). (1) 通信回線によってアクセスされるシステム、プログラム、または装置に関する用語。(2)*リンク接続 (link-attached)* の同義語。(3) *ローカル(local)* と対比。

リモート・ホスト (remote host). ネットワーク内で、特定の操作員が作業を行っているホストを除くすべてのホスト。*外部ホスト (foreign host)* と同義。

要求（request）. 特定のアクションまたはプロトコルの開始を知らせるメッセージ。たとえば、Initiate-Self は LU-LU セッションの活動化の要求。

リクエスター (requester). サーバーを介して共用ネットワーク資源にアクセスするコンピューター。*クライアント (client)* の同義語。

要求ヘッダー (request header (RH)). 要求単位 (RU) に先行する制御情報。 *要求応答ヘッダー (request/response header (RH))* も参照。

要求応答ヘッダー (request/response header (RH)). 特定の RU に関連した制御情報。 RH は要求応答単位 (RH) の先頭にあり、RU（要求単位または応答単位）のタイプを指定する。

要求応答単位（RU）(request/response unit (RU)). 要求単位または応答単位の一般用語。 *要求単位 (request unit (RU))* および *応答単位 (response unit (RU))* を参照。

送信要求 (request to send (RTS)). データ通信において、データ端末が作動可能状態にあるときにデータの送信を可能にするために、データ端末装置 (DTE) が、データ回線終端装置 (DCE) からの機能を要求するために立ち上げるシグナル。 *送信可 (clear to send (CTS))* と対比。

要求単位 (request unit (RU)). 制御情報、エンド・ユーザー・データ、またはその両方を含むメッセージ単位。

リセットする（reset）. 仮想回線では、データ・フロー制御の再初期設定。 リセットすると、転送中のすべてのデータが除去される。

資源 (resource). ジョブまたはタスクのために必要なコンピューティング・システムまたはオペレーティング・システムの機能であって、主記憶装置、入出力装置、処理装置、データ・セット、および制御プログラムまたは処理プログラムなどが含まれる。

資源引継ぎ (resource takeover). VTAM において、接続が切れたり、接続上の既存の LU-LU セッションが中断されたりすることなく定義域から他の定義域へと資源の制御を転送するために、ネットワーク操作員が開始するアクション。*獲得（する）(acquire)* および *解放（する）(release)* も参照。

資源タイプ (resource types). NetView プログラムにおいて、パネルの編成を記述する概念。資源タイプは、中央処理装置、チャネル、制御装置、および入出力装置が１つのカテゴリーとして定義され、通信制御装置、アダプター、リンク、クラスター制御装置、および端末が別のカテゴリーとして定義される。資源タイプとデータ・タイプおよび表示タイプとの組合せにより、表示編成が記述される。*データ・タイプ (data types)* および *表示タイプ (display types)* も参照。

応答（response）. (1) データ通信において、応答フレームの制御フィールドで表現された応答。 １次ステーションまたは複合ステーションに、2 次ステーションまたは他の複合ステーションが 1 つまたは複数のコマンドに対して行ったアクションを知らせる。(2) コマンド *(command)*も参照。

応答ヘッダー （response header (RH)）. (1) 応答が肯定であるか否定であるかを示すヘッダー。このヘッダーの後には任意選択的に応答単位 (RU) が続く場合があり、またヘッダーにペーシング応答が含まれている場合がある。(2) *否定応答 (negative response)*、*ペーシング応答 (pacing response)*、および *肯定応答 (positive response)*も参照。

応答標識 (response indicator). AS/400 システムにおいて、システムからプログラムへ入力レコードと共に渡される 1 文字のフィールドであって、データ・レコードまたはワークステーションのユーザーによってとられたアクションに関する情報を提供するもの。

応答単位 (response unit (RU)). 要求単位に肯定応答を与えるメッセージ単位。 要求単位で受信した接頭部情報が含まれている場合もある。 正の場合、応答単位には、(BIND SESSION に応答するためのセッション・パラメーターなどのような) 追加情報が含まれることがある。負の場合は、応答単位には、例外条件を定義するセンス・データが入っている。

再同期 (resync). 同期点処理中にセッション、トランザクション・プログラム、または LU の障害が発生したときに、同期点サービスが行う回復処理。再同期の目的は、保護された資源を整合性のある状態に戻すことにある。

戻りコード (return code). (1) 後続の、命令の実行に影響を与えるために使用されるコード。 (A) (2) プログラムによって要求された操作の結果を示すため、そのプログラムに戻される値。

REX. 経路の拡張 (route extension) を参照。

RH. 要求/応答ヘッダー (Request/responce header) を参照。

リング（ring）. *リング・ネットワーク（ring network）*を参照。

リング・ネットワーク（ring network）. ネットワーク構成において、装置が単一方向の伝送リンクによって接続され、閉鎖経路を形成しているネットワーク構成。

経路 (route). (1) 起点ノードから宛先ノードへのパスを形成し、両ノード間で交換されるトラフィックが通過する、一連のノードおよび伝送グループ (TG) の並び。(2) ネットワーク・トラフィックが起点から宛先に達するため使用するパス。

ルート d（routed）. 「ルートディー」と発音する。*経路デーモン (route daemon)* を参照。

経路デーモン (route daemon). ローカル・エリア・ネットワーク内の各マシンに経路情報を伝達するために 4BSD UNIX のもとで実行されるプログラム。 *ルート d (routed)* (「ルートディー」と発音する) も参照。

経路拡張機能 (route extension (REX)). SNA におけるパス制御ネットワーク構成要素。この構成要素は周辺リンクを含み、またサブエリア・ノードと、隣接した周辺ノードの中のネットワーク・アドレス可能単位 (NAU) との間のパス部分を形成している。 *明示経路 (explicit route (ER))* 、*パス (path)* 、および*仮想経路 (virtual route (VR))* も参照。

ルーター (router). (1) ネットワーク・トラフィック・フローのパスを決定するコンピューター。いくつかのパスから選択するときの基準となるのは、特定のプロトコルから得られる情報、最短または最良のパスを識別するためのアルゴリズム、およびメトリックやプロトコル固有宛先アドレスなどのような、その他の基準である。(2) 類似してはいるが異なるアーキテクチャーを使用する2つの LAN セグメントを、リファレンス・モデル・ネットワーク層で接続する接続装置。(3) OSI 用語では、あるエンティティーに到達するためのパスを決定する機能。(4) TCP/IP では、*ゲートウェイ (gateway)* と同義。(5) *ブリッジ (bridge)* と対比。

ルーチン (routine). 汎用または使用頻度の高いプログラムまたはプログラムの一部。 (T)

経路指定 (routing). (1) ネットワークを介してメッセージを伝送するために使用されるパスを決定するプロセス。 (T) (2) メッセージが宛先に到達するために使用するパスを割り当てること。(3) SNA において、メッセージ単位の中にあるパラメーターに従って、ネットワークを通る特定のパスに沿ってそのメッセージ単位を渡すこと。パラメーターの例としては、伝送ヘッダーの中にある宛先ネットワーク・アドレスがある。

RQD. 接続解除要求 (Request discontact) を参照。

RTS. 送信要求 (Request to send)。

RU. 要求応答単位 (request/response unit)。

RU 連鎖 (RU chain). SNA において、特定の通常または急送データ・フローの上を連続的に伝送される、関連した要求/応答単位 (RU) の集合。 要求 RU 連鎖は回復単位である。すなわち、連鎖中のRU の 1 つが処理不可能であれば、全体の連鎖が破棄される。各 RU はそれぞれ 1 つの連鎖だけに所属し、RU の始めと終りは、RU 連鎖内の要求応答ヘッダーの制御ビットにより指示される。各 RU は、先頭連鎖 (FIC)、最終連鎖 (LIC)、中間連鎖 (MIC)、または単独連鎖 (OIC) のいずれかとして指定できる。 応答単位および急送フロー要求単位は、常に「単独連鎖」として送信される。

S

SBCS. 1 バイト文字セット (Single-byte character set) を参照。

SC. セッション制御 (session control)。

SDLC. 同期データ・リンク制御 (Synchronous Data Link Control)。

SDT. データ・トラフィック開始 (Start data traffic)。

2 次側ハーフセッション (secondary half-session). SNA において、セッション活動化要求を受け取るハーフセッション。 *2 次論理装置 (secondary logical unit (SLU))* も参照。 *1 次側ハーフセッション (primary half-session)* と対比。

2 次論理装置 (secondary logical unit (SLU)). SNA において、特定の LU-LU セッションについて第 2 のハーフセッションを含む論理装置 (LU)。 1 つの LU に、いくつかの異なる活動 LU-LU セッション用の 2 次側ハーフセッションおよび 1 次側ハーフセッションを含めることができる。 *1 次論理装置 (primary logical unit (PLU))* と対比。

2 次論理装置 (SLU) キー (secondary logical unit (SLU) key). 2 次側ハーフセッションにセッション暗号キーを伝送するときに、そのキーを保護するために使用されるキー暗号キー。

セグメント (segment). (1) 複数の構成要素または装置の間のケーブル部分。 セグメントは、単一のパッチ・ケーブル、相互に接続されたいくつかのパッチ・ケーブル、または、相互に接続された構築ケーブルおよびパッチ・ケーブルの組合せから成る。(2) インターネット通信において、異なるマシンの TCP 機能間での転送の単位。各セグメントには、制御フィールドとデータ・フィールドが含まれている。受信したデータの妥当性検査のために、現行バイト・ストリーム位置および実データ・バイトが、チェックサムとともに識別される。(3) *BIU セグメント (BIU segment)* の同義語。(4) *リンク接続セグメント (link connectionsegment)*も参照。

セグメンテーション (segmentation). 隣接ノードの小さいバッファー・サイズに適合させるために、パス制御 (PC) が基本情報単位 (BIU) を BIU セグメントと呼ばれる小さい単位に分割するプロセス。 セグメント化とセグメント・アセンブリーのどちらも、オプションの PC 機能である。 この一方または両方に対するサポートは、BIND 要求および応答に指示される。

セグメント化 (segmenting). OSI において、ある層が、サポートする層からの 1 つのプロトコル・データ単位 (PDU) を複数の PDU にマップする機能。

選択する (select). 後で選択動作を適用するために 1 つまたは複数のオブジェクトを明確に識別すること。

選択（selection）. 後続の選択項目を適用する 1 つまたは複数のオブジェクトを明示的に識別するプロセス。

セマフォー (semaphore). ファイルへのアクセスを制御するために使用される標識。たとえば、マルチユーザー・アプリケーションでは、ファイルへの同時アクセスを防止するためのフラグ。

送信ペーシング (send pacing). SNA において、構成要素が送信しているメッセージ単位のペーシング。 *受信ペーシング (receive pacing)* と対比。

順序番号 (sequence number). (1) 通信において、伝送フローおよびデータの受信を制御するために特定のフレームまたはパケットに割り当てられる番号。(2) 2 つのノード間で交換される各メッセージに VTAM が割り当てる数値。アプリケーション・プログラムが順序番号設定およびテスト (STSN) 標識によりリセットしない限り、連続して伝送される各メッセージごとに 1 ずつ増加する値（アプリケーション・プログラムが論理装置に送るメッセージ用に 1 つ、論理装置がアプリケーション・プログラムに送るメッセージ用に 1 つある）。

逐次、直列 (serial). (1) すべてのイベントが順次に起こるようなプロセスに関する用語。たとえば、V24 CCITT プロトコルに従って文字ビットを順次に伝送する。 (T) (2) 1 つの装置またはチャネルで、関連した複数の活動が連続して順次に起こることに関する用語。 (A) (3) 全体の中の個々の部分 (たとえば、文字を構成するビットや語を構成する文字) を、連続する各部分について同じ機能を使用して、順次に処理することに関する用語。 (A) (4) *並行、並列 (parallel)* と対比。

サーバー (server). (1) ネットワークを介してワークステーションへ共通サービスを提供する機能単位。たとえば、ファイル・サーバー、プリント・サーバー、メール・サーバーなどがある。 (T) (2) ネットワークにおいて、他のステーションへ機構または機能を提供するデータ・ステーション。たとえば、ファイル・サーバー、プリント・サーバー、メール・サーバーなどがある。 (A) (3) AIX オペレーティング・システムにおいて、通常、バックグラウンドで実行され、システム・プログラム制御手段によって制御されるアプリケーション・プログラム。(4) 拡張 X-Windows ツールキットにおいて、基本ウィンドウ・

メカニズムを提供するプログラム。クライアントからのプロセス間通信 (IPC) 接続を操作し、画面上でグラフィックス要求の多重化を解除し、そして入力を再度多重化してクライアントに戻す。

サービス・ポイント (service point (SP)). 1つのエントリー・ポイントとして、それ自体の直接制御下にない資源のためにネットワーク管理を提供するアプリケーションをサポートするエントリー・ポイント。 各資源は、別のエントリー・ポイントの直接制御下にあるか、またはどのエントリー・ポイントの直接制御下にもない。 この種の資源にアクセスするサービス・ポイントは、SNAセッションを使用する必要はない（この点はフォーカル・ポイントとは異なる）。 何らかのネットワーク管理機能についてエントリー・ポイント・サポートがまだ使用可能になっていないときに、サービス・ポイントが必要になる。

セッション. (1) ネットワーク・アーキテクチャーでは、複数の機能単位の間でデータ通信を行うためのすべての活動であって、接続の確立、維持、および解放の間に起こるもの。 (T) (2) 2 つのネットワーク・アクセス可能単位（NAU）の間の論理接続。セッションは、活動化し、各種プロトコルを提供するように調整し、要求に基づいて非活動化することができる。 各セッションは、セッション中に交換される各伝送に付随する伝送ヘッダー (TH) の中で、固有のものとして識別される。(3) サーバーとリクエスターとの間の接続であって、共有資源への要求が成功したときに開始されるもの。*ホスト・セッション (host session)*および*DOS セッション (DOS session)*も参照。

セッション活動化要求 (session activation request). SNA において、2 つのネットワーク・アクセス可能単位 (NAU) の間でセッションを活動化し、セッション活動の間に種々のプロトコルを制御するセッション・パラメーターを指定する要求。たとえば、BIND および ACTPU がある。 *セッション非活動化要求 (session deactivation request)* と対比。

セッション・コネクター (session connector). APPN ネットワーク・ノード内のセッション層構成要素。または、1セッションの 2 つのステージを接続するサブエリア・ノード境界またはゲートウェイ機能の中のセッション層構成要素。 セッション・コネクターは、セッション・レベルの中間経路指定で 1 つのアドレス空間を別のアドレス空間に交換し、必要に応じてセッション・メッセージ単位をセグメント化し、そして、(ゲートウェイ機能セッション・コネクターの場合を除き）各方向のセッション・トラフィックを適正にペーシングする。*ハーフセッション (half-session)* も参照。

セッション制御 (session control (SC)). SNA において次のいずれか 1 つ。

- 伝送制御の構成要素の 1 つ。セッション制御は、回復不能エラーが発生した後でセッション内を流れるデータを除去したり、その種のエラーの後でデータ・フローを再同期したり、また暗号検査を行ったりするために使用される。
- セッションのセッション制御構成要素間で交換する要求および応答用として、また、セッションの活動化および非活動化の要求および応答用として使用される要求単位 (RU) カテゴリー。

セッション・データ (session data). NetView プログラムが収集する、セッションに関するデータ。セッション認識データ、セッション・トレース・データ、およびセッション応答時間データから成る。

セッション非活動化要求 (session deactivation request). SNA において、2 つのネットワーク・アクセス可能単位(NAU)の間でセッションを非活動化する要求。たとえば、UNBIND およびDACTPU などがある。 *汎用 UNBIND (generic unbind)* と同義。 *セッション活動化要求 (session activation request)* と対比。

セッション ID (session ID). パーソナル・コミュニケーションズによって各セッションまたは画面へ割り当てられるアルファベットの ID (*a* から *h*)。これはすべてのタイプのホスト・セッションへ適用され、ファイル転送で使用される。*短縮名 (short name)*も参照。

セッション・レベル LU-LU 検査 (session-level LU-LU verification). セッションの確立時に、各論理装置のアイデンティティーを検査するために使用される LU 6.2 セキュリティー・サービス。

セッション・レベル・ペーシング (session-level pacing). (1) 受信側ハーフセッションまたはセッション・コネクターが、通常フローでのデータ転送速度（要求単位を受信する速度）を制御できるようにするフロー制御の技法。 この技法は、送信側での要求生成の速度が受信側の処理能力を上回る場合に、未処理の要求により受信側に過負荷が発生するのを防ぐために使用される。(2) *最適セッション・レベル・ペーシング (adaptive session-level pacing)*、*固定セッション・レベル歩調合せ (fixed session-level pacing)*、および*仮想経路ペーシング (virtual route (VR))* も参照。

セッション限度 (session limit). 特定の論理装置 (LU) が同時にサポートできる活動 LU-LU セッションの最大数。

セッション・パラメーター (session parameters). SNA において、2 つのネットワーク・アクセス可能単位間のセッションについて、プロトコル（ブラケット・プロトコルおよびペーシングなど）を指定または制限するパラメーター。*ログオン・モード (logon mode)* も参照。

セッション・サービス (session services (SS)). 制御点 (CP) および論理装置 (LU) 内のネットワーク・サービスのタイプの 1 つ。 この種のサービスは、LU またはネットワーク操作員を対象として、論理装置間のセッションの開始または終了に関する制御点 (ENCP、NNCP、または SSCP) 補助を要求する機能を提供する。 セッション終了に関する補助は、SSCP 従属 LU の場合に限り必要とされる。 *構成サービス (configuration services)* 、*保守サービス (maintenance services)* 、および*管理サービス (management services)* を参照。

共用 (shared). 1 つの資源を同時に複数の用途に使用できることを表す。

シェル（shell）. (1) コンピューターのユーザーとオペレーティング・システム間のソフトウェア・インターフェース。シェル・プログラムは、キーボード、指示装置、タッチ検知スクリーンなどの装置でのコマンドおよびユーザー対話を解釈し、それをオペレーティング・システムに伝える。シェルは、オペレーティング・システムの要件に対するユーザーの配慮をなくすことで、ユーザー対話を簡素化する。コンピューターは、各種レベルのユーザー対話用のいくつかの層またはシェルを備えていることがある。(2) AIX オペレーティング・システムでは、ユーザーとオペレーティング・システムとの間のインターフェースとして働くコマンド・インタープリター。シェルの内部にはネストされた別のシェルを含むことができる。この場合、外側のシェルが親シェルで、内側のシェルが子になる。

シフトイン文字 (shift-in character (SI)). シフトアウト文字で始まった文字列を終わらせて、標準文字セットのグラフィック文字に効力を与えるコード拡張文字。 (I) (A)

シフトアウト文字 (shift-out character (SO)). 標準文字セットのグラフィック文字の代わりに、合意に達しているかまたはコード拡張手順により指定してある代替グラフィック文字セットを使用するためのコード拡張文字。 (I) (A)

短縮名 (short name). (1) パーソナル・コミュニケーションズにおいて、操作員情報域の列 7 に表示される文字であって、セッション ID を示すもの。*セッション ID (session ID)*および*操作員情報域(operator information area)*も参照。(2) コミュニケーション・マネージャー/2 において、表示空間またはエミュレーション・セッションの 1 文字の名前 (A から Z)。 *表示空間 ID (presentation space ID (PSID))* および*短セッション ID (short-session ID)*と同義。

短セッション ID (short-session ID). コミュニケーション・マネージャー/2 において、*短縮名 (short name)*の同義語。

シャットダウン (shutdown). 定義された手順に従い、システムまたはサブシステムの動作を終了させるプロセス。

SI. シフトイン文字 (shift-in character) を参照。 (I) (A)

1 バイト文字セット (single-byte character set(SBCS)). 各文字が、1 バイト・コードで表される文字セット。*2 バイト文字セット (double-byte character set (DBCS))* と対比。

SLU. 2 次論理装置 (secondary logical unit)。

SNA. システム・ネットワーク体系 (Systems Network Architecture)。

SNA ネットワーク (SNA network). システム・ネットワーク体系の形式とプロトコルに合わせたユーザー・アプリケーション・ネットワークの部分のこと。 ユーザー間での信頼性の高いデータ転送を可能にし、各種ネットワーク構成の資源を制御するためのプロトコルを提供する。 SNA ネットワークは、ネットワーク・アクセス可能単位 (NAU)、境界機能、ゲートウェイ機能、および中間セッション経路指定機能の各構成要素と、トランスポート・ネットワークから成っている。

SNAP. サブネットワーク・アクセス・プロトコル (Subnetwork Access Protocol) を参照。

SO. シフトアウト文字 (shift-out character)。 (I) (A)

SP. サービス・ポイント(Service point) を参照。

SSCP. システム・サービス制御点 (system services control point)。

SSCP 従属 LU (SSCP-dependent LU). LU-LU セッションの開始のためにシステム・サービス制御点 (SSCP) からの援助を必要とする LU。 SSCP-LU セッションを必要とする。

SSCP-LU セッション（SSCP-LU session）. SNA では、システム・サービス制御点（SSCP）と論理装置（LU）と

の間のセッション。 このセッションで、LU は LU-LU セッションを開始するための援助を SSCP に要求できる。

SSCP-PU セッション（SSCP-PU session）. SNA において、システム・サービス制御点(SSCP) と物理装置 (PU) との間のセッション。SSCP-PU セッションによって、SSCP は、ネットワーク構成を制御するために個々のノードへ要求を送り、またノードから状況情報を受け取ることができる。

スタック (stack). *プッシュダウン・リスト (pushdown list)*の同義語。

ステージ (stage). NetView パイプライン内のメッセージを処理するプログラム。ステージは、互いに逐次的にメッセージを送る。

ステートメント (statement). 演算子または他のステートメント識別子と、それに続く 1 つまたは複数のオペランドから成る言語構文単位。*定義ステートメント (definition statement)* を参照。

静的 (static). (1) プログラミング言語において、プログラムを実行する前に確立できる特性に関する用語。たとえば、固定長変数の長さは静的である。 (I) (2) 定義済みの時刻または固定された時刻に行われる操作に関する用語。(3) *動的 (dynamic)* と対比。

状況 (status). 通常状況コードで表される、ハードウェアまたはソフトウェアの条件または状態。

記憶装置、格納 (storage). (1) データを入れておき、データを検索することのできる機能単位。 (T) (2) データを記憶装置に入れるアクション。 (I) (A) (3) 記憶装置 (storage device)。 (A)

ストリーム (stream). (1) 1 つの装置から他の装置へデータを送ること。(2) データ・ストリーム *(data stream)*を参照。

STSN. 設定およびテスト順序番号 (set and test sequence numbers)。

スタブ (stub). (1) クライアントとサーバーの間でリモート・プロシージャー呼出し (RPC) および応答を転送するプログラム・モジュール。スタブは、整列、整列解除、およびデータ形式変換を行う。クライアントとサーバーはどちらもスタブを備えている。ネットワーク・インターフェース定義言語 (NIDL) コンパイラーは、インターフェース定義からクライアントおよびサーバー・スタブ・コードを生成する。(2) 拡張Xウィンドウ・ツールキット用のプロトコル要求を生成するために、プロトコルへの拡張機能として使用されるフック機能。(3) アプリケーション・コードに連係編集されている小さいモジュールで、関連コードの大きいボディを見つけてそれに制御を転送する。

サブエリア (subarea). SNA ネットワークの一部分であって、サブエリア・ノード、接続された周辺ノード、および関連の資源から構成されている。 サブエリア・ノード内では、すべてのネットワーク・アクセス可能単位 (NAU)、リンク、およびそのサブエリア・ノード内でアドレス可能な隣接リンク・ステーション（接続している周辺ノードまたはサブエリア・ノード内の）が、共通のサブエリア・アドレスを共用し、それぞれ異なる要素アドレスを持っている。

サブエリア LU (subarea LU). SNA において、サブエリアの中にある論理装置。 *周辺 LU (peripheral LU)* と対比。

サブエリア・ネットワーク（subarea network）. 相互接続されたいくつかのサブエリア、それらに直接接続している周辺ノード、および、それらに付随する伝送グループ。

サブエリア・ノード (subarea node (SN)). 経路指定用にネットワーク・アドレスを使用し、ネットワークの構成を反映する経路指定テーブルを維持するノード。 サブエリア・ノードは、複数のサブエリア・ネットワークを接続するゲートウェイ機能、中間経路指定機能、および、周辺ノード用の境界機能サポートを提供する。 サブエリア・ノードとなることができるのは、タイプ 4 およびタイプ 5 のノードである。

サブエリア PU (subarea PU). SNA において、サブエリアの中にある物理装置。 *周辺 PU (peripheral PU)* と対比。

サブディレクトリー (subdirectory). ファイル・システム階層において、1 つのディレクトリーの中に含まれた他のディレクトリー。

サブネットワーク・アクセス・プロトコル (Subnetwork Access Protocol (SNAP)). LAN において、パケットが属する非 IEEE 標準プロトコル・ファミリーを識別する、5 バイトのプロトコル判別子。 SNAP 値は、`$AA` をサービス・アクセス・ポイント (SAP) として使用するプロトコルを識別するために使用される。

サブシステム (subsystem). 2 次的または従属的なシステム。通常、このシステムは制御を行っているシステムとは独立して(すなわち非同期的に) 動作することができる。 (T)

サブシステム管理 (subsystem management). コミュニケーション・マネージャー/2 において、診断および問題解決を目的とする拡張機能のグループ。これらの機能は、主としてシステム管理者およびアプリケーション・プログラマーが使用する。

SVC. 相手選択接続、仮想交換回線 (switched virtual circuit (SVC))。

相手選択接続、仮想交換回線 (switched virtual circuit (SVC)). 必要なときに動的に確立される X.25 回線。交換回線の X.25 版。 *相手固定接続 (permanent virtual circuit (PVC))* と対比。

同期点 (synchronization point). *同期点 (sync point)* の同義語。

同期 (synchronous). (1) 共通タイミング信号のように、特定のイベントの発生に依存する複数のプロセスに関する用語。 (T) (2) 定期的または予測可能な関係で発生することに関する用語。

同期データ・リンク制御 (Synchronous Data Link Control (SDLC)). 米国規格協会 (ANSI) の拡張データ通信制御手順(ADCCP)および国際標準化機構のハイレベル・データ・リンク制御 (HDLC) のサブセットに準拠した規則であって、リンク接続を介して行われる同期、コード透過、ビット順のデータ伝送を管理するもの。 伝送交換は、交換リンクまたは非交換リンクを介して二重または半二重で行う。 リンク接続の構成は、2 地点間、マルチポイント、またはループ接続になる。 (I)

同期操作 （synchronous operation）. VTAM での通信またはその他の操作の方式の 1 つ。同期操作では、VTAM は、操作の要求を受け取ってから、その操作が完了するまで、プログラムに制御を戻さない。*非同期操作 (asynchronous operation)* と対比。

同期要求 (synchronous request). VTAM において、同期操作を求める要求。*非同期要求 (asynchronous request)* と対比。

同期点 (sync point). トランザクションの処理において、そのトランザクションの 1 つまたは複数の保護された資源に対する更新または変更が、論理的に完了してエラーなしの状態となる中間点または終点。 *同期点 (synchronization point)* と同義。

システム（system）. データ処理において、特定の機能の集合を実行するように組織化された人、マシン、および方法より成るもの。 (I) (A)

システム・セマフォー (system semaphore). OS/2 オペレーティング・システムにおいて、複数のプロセスのスレッド間の制御を提供するシグナル・メカニズムで、メモリーを共用しない複数のプロセスにより使用できる。 *RAM セマフォー (RAM semaphore)* と対比。

システム・サービス制御点 (system services control point (SSCP)). サブエリア・ネットワーク内の構成要素の 1 つで、構成を管理し、ネットワーク操作員および問題判別要求を調整し、ネットワーク・ユーザーにディレクトリー・サービスおよびその他のセッション・サービスを提供する。 複数の SSCP が対等機能として互いに協調して働き、ネットワークを制御の定義域に分割することができる。この場合、各 SSCP は、それぞれの担当の定義域内の物理装置および論理装置に対して階層的な制御関係を持つことになる。

システム・サービス制御点 (SSCP) 定義域 (system services control point (SSCP) domain). システム・サービス制御点、物理装置 (PU)、論理装置(LU)、リンク、リンク・ステーション、および SSCP が活動化および非活動化によって制御することのできるすべての資源。

システム・ネットワーク体系 (Systems Network Architecture (SNA)). ネットワークを介して情報単位を伝送し、またネットワークの構成および動作を制御するための、論理構造、プロトコル、および操作順序のこと。SNA の構造は階層化されているので、情報の末端の起点および宛先（つまりユーザー）は、情報交換に使用する特定の SNA ネットワーク・サービスおよび機能に依存することも、またその影響を受けることもない。

T

テーブル (table). NetDA/2 がネットワークの設計のために使用するデータのリポジトリー。各テーブルにはネットワークに関連した情報が入っている。

タスク (task). マルチプログラミングまたは多重処理環境における命令のシーケンスであって、制御プログラムが、コンピューターによって実行されるべき 1 つの作業要素として取り扱うもの。 (I) (A)

タスク関連ユーザー出口 (Task-Related User Exit (TRUE)). CICS の外部にある資源管理プログラムを呼び出すために使用する CICS モジュール。TRUE は CICS ソケットの構成要素の 1 つ。

TC. 伝送制御 (transmission control)。

一時エラー (temporary error). エラー回復プログラムによって解決できる資源障害。*パフォーマンス・エラー (performance error)* と同義。*永続エラー (permanent error)* と対比。

TERMINATE. SNA において、指定された LU-LU セッションを終わらせる手順をシステム・サービス制御点 (SSCP) に開始させるために、論理装置 (LU) によってその SSCP へ送られる要求単位。

TH. 伝送ヘッダー (transmission header)。

スレッド (thread). OS/2 オペレーティング・システムにおいて、1 つのプロセス内で行う操作の最小単位。

タイムアウト（timeout）. (1) 指定されたイベントの開始時に始まった時間間隔の終わりに発生する他のイベント。 (I)(2) 特定の操作を行うために割り当てられた時間間隔。たとえば、ポーリングまたはアドレス指定が行われてから、この時間内に応答を行わないと、システムが中断されて再始動が必要になる場合など。

TLPB. トランスポート層プロトコル境界 (Transport-layer protocol boundary) を参照。

トークン（token）. (1) ローカル・エリア・ネットワークにおいて、伝送媒体を一時的に制御しているステーションを示すために、1 つのデータ・ステーションから他のデータ・ステーションへ次々に渡される、権限を示す記号。 どのデータ・ステーションにも、トークンを獲得し、それを使用して媒体を制御する機会がある。 トークンは、伝送の許可を意味する特別なメッセージつまりビット・パターンである。 (T) (2) LAN において、伝送媒体を介して 1 つの装置から別の装置に渡されるビット順序。 トークンにデータが付加されている場合は、全体として 1 つのフレームとなる。

トークンリング（token ring）. (1) IEEE 802.5 によると、媒体接続ステーション間でのトークン（特殊パケットまたはフレーム）の受渡しにより媒体アクセスを制御するネットワーク・テクノロジー。(2) 接続しているリング・ステーション（ノード）から別のノードへトークンを渡すリング・テクノロジーを使用する FDDI または IEEE 802.5 ネットワーク。(3) *ローカル・エリア・ネットワーク (local area network (LAN))* も参照。

トークンリング・ネットワーク (token-ring network). (1) データ・ステーションの間で単一方向のデータ伝送を行って、トークン渡しの手順により、伝送されたデータが送信ステーションへ戻されるようにするリング・ネットワーク。 (T)(2) トークンが回線中をノードからノードへと渡されるリング・トポロジーを使用するネットワーク。送信可能な状態にあるノードは、トークンをキャプチャーし、伝送するデータを挿入することができる。

TP. トランザクション・プログラム (Transaction Program) を参照。

トランザクション・プログラム (transaction program (TP)). SNA ネットワークでトランザクションを処理するプログラム。 次の 2 種類のトランザクション・プログラムがある。アプリケーション・トランザクション・プログラムおよびサービス・トランザクション・プログラム。*会話 (conversation)* も参照。

変換テーブル (translation table). 文字を代替の文字と置換するために使用されるテーブル。このテーブルは、たとえば、仮想アドレスを表している文字を、実アドレスを表す文字へ変換したり、イベントを表している文字を、プロシージャー呼出しを表す文字へ変換したり、ある国の文字セットの文字を他の国の言語文字へ変換したり、再配置アドレスを表している文字を、絶対アドレスを表す文字へ変換する場合に使用される。

伝送制御 (TC) 層 (transmission control (TC) layer). ハーフセッションまたはセッション・コネクター内部にあり、セッション・レベル・データ・トラフィックの同期とペーシングを行い、要求のセッション順序番号を検査し、エンド・ユーザー・データを暗号化および解読する層。*ハーフセッション (half-session)* も参照。

伝送ヘッダー (transmission header (TH)). メッセージ単位を経路指定し、またネットワーク内でメッセージ単位の流れを制御するために、パス制御によって作成され使用される制御情報。この制御情報には、基本情報単位 (BIU) または BIU セグメントが続いている場合がある。*パス情報単位 (path information)* も参照。

トランスポート層プロトコル境界 (transport-layer protocol boundary (TLPB)). MPTN アーキテクチャーにおいて、プロトコルに依存しない方式で複数のトランスポート・プロトコルにアクセスできるようにするプロトコル境界。

トラップ (trap). 簡易ネットワーク管理プロトコル (SNMP) において、管理ノード（エージェント機能）が、例外条件の報告のために管理ステーションに送るメッセージ。

TRUE. タスク関連ユーザー出口 (Task-Related User Exit) を参照。

チュートリアル (tutorial). 学習形式で提供される情報。

U

UNBIND. SNA において、2 つの論理装置 (LU) の間のセッションを非活動化するための要求。 *セッション非活動化要求 (session deactivation request)* も参照。 *BIND* と対比。

非解釈名 (uninterpreted name). SNA において、システム・サービス制御点 (SSCP) が論理装置 (LU) のネットワーク名に変換することのできる文字ストリング。一般に、非解釈名は、2 次論理装置 (SLU) からのログオンまたは開始要求の際に、どの 1 次論理装置 (PLU) とのセッションを要求するかを指示するために使用される。

UPM. ユーザー・プロファイル管理 (User Profile Management) を参照。

ユーザー (user). (1) 情報処理システムとの間でコマンドおよびメッセージを送受信する人または物。 (T) (2) 計算機システムのサービスを必要とする人。

ユーザー識別子 (user identifier (UID)). ネットワーク上またはシステム上のユーザーを識別する独特の名前。

ユーザー・プロファイル (user profile). コンピューター・セキュリティーにおいて、特定のユーザーの記述。ユーザー ID、ユーザー名、パスワード、アクセス権限、および、ログオン時に取得するその他の属性などの情報が含まれている。

ユーザー・プロファイル管理 (User Profile Management (UPM)). OS/2 オペレーティング・システムにおいて、ユーザー ID の妥当性検査およびユーザーおよびグループの管理のための機能。 コミュニケーション・マネージャー/2は UPM を使用する。 UPM の導入形態は個々のワークステーションにとってローカルのものであり、そのワークステーションに常駐している制御対象のデータまたはプログラムへのユーザー・アクセスの導入および妥当性検査を行う。 UPM は、システム・ユーザーを識別し認証するためのログオン・ログオフ・メカニズムも備えている。

UTC. 世界協定時 (Coordinated universal time)。

V

妥当性検査 (validation). データが正しいかどうか、また適用可能な標準、規則、および規約にデータが準拠しているかどうかをチェックすること。 (A)

値 (value). (1) 属性の特定の発現。たとえば、属性「色」について「青」と表示されること。 (T) (2)定数、変数、パラメーター、または記号に割り当てられた量。

変数 (variable). (1) プログラミング言語において、異なった値を一度に 1 つだけ取ることができる言語オブジェクト。通常は、変数の値は特定のデータ・タイプに限定されている。 (I) (2) 複数の値より成る所与の集合から、任意の値を取ることができる量。 (A) (3) プログラムが実行されている間にデータ項目の値が変化する場合に、データ項目を表すために使用される名前。

verb. *LU 6.2 verb* を参照。

バージョン (version). 別個にライセンスされるプログラムであって、通常、著しく新規なコードまたは新しい機能を含んでいるもの。

仮想回線 (virtual circuit). (1) パケット交換において、ネットワークによって提供され、ユーザーに対して現実の接続であるかのような外観を呈する機構または機能。 (T) *データ回線 (data circuit)*も参照。 *物理回線 (physicalcircuit)* と対比。(2) 2 つの DTE 間で確立される論理接続。

仮想経路 (virtual route (VR)). (1) SNA において、次のいずれか。(a) 2 つのサブエリア・ノードの間にある論理接続であって、物理的には特定の明示経路として実現されるもの。(b) ノード間セッションのために全面的に 1 つのサブエリア内に含まれている論理接続。 異なるサブエリア・ノード間の仮想経路は、基礎となっている明示経路に何らかの伝送優先順位を課し、仮想経路ペーシングを使用してフロー制御を行い、パス情報単位 (PIU) の順序番号付けによりデータ保全性を確保する。(2) *明示経路 (explicit route (ER))*と対比。 *パス (path)* および *経路拡張機能 (route extension (REX))* も参照。

仮想経路 (VR) ペーシング (virtual route (VR) pacing). SNA において、パス情報単位 (PIU) が仮想経路上を流れる速度を制御するため、仮想経路の両端でパス制御の仮想経路制御構成要素によって使用される、フロー制御の手法。 VR ペーシングは、経路上の任意のノードでのトラフィック輻輳（ふくそう）の程度に応じて調整できる。 *セッション・レベル・ペーシング (session-level pacing)* も参照。

仮想記憶通信アクセス方式 (Virtual Telecommunications Access Method (VTAM)). SNA ネットワークで通信とデータの流れを制御する IBM ライセンス・プログラム。単一定義域、複数定義域、および相互接続ネットワーク機能を提供する。

VR. 仮想経路。

VTAM. (1) 仮想記憶通信アクセス方式 (Virtual Telecommunications Access Method)。(2) *ACF/VTAM* と同義。

W

WAN. 広域ネットワーク (wide area network)。

広域ネットワーク (wide area network (WAN)). (1) ローカル・エリア・ネットワークまたは大都市圏ネットワークの対象範囲より広い地域を対象として通信を提供し、公共通信施設を利用または提供できるネットワーク。 (T) (2) 数百または数千マイルの区域にサービスするように設計されたデータ通信ネットワーク。たとえば、公衆および私用のパケット交換ネットワーク、および国内電話網など。(3) *ローカル・エリア・ネットワーク (local area network (L)AN)* および *大都市圏ネットワーク (metropolitan area network (MAN))* と対比。

ウィンドウ (window). (1) 表示画面の一部分であり、特定のアプリケーションに関する表示像を表示できる画面部分。 いくつかの異なるアプリケーションを、それぞれ異なるウィンドウに同時に表示できる。(2) 目に見える境界がある区域であり、そこにオブジェクトのビューが表示されるか、それを用いてユーザーがコンピューター・システムと対話を行うもの。(3) データ通信におけるデータ・パケットの数であり、データ端末装置 (DTE) またはデータ回線終端装置 (DCE) は、その数だけのデータ・パケットを、他のデータ・パケットを送る許可を待つ前に、論理チャネルを通して送ることができる。 ウィンドウは、パケットのペーシング、つまりフロー制御の主要メカニズムである。(4) ペーシング *(pacing window)* を参照。

WinSock アプリケーション・プログラミング・インターフェース (WinSock application programming interface (API)). Windows ファミリーのオペレーティング・システム用に開発されたソケット・スタイルのトランスポート・インターフェース。

ワークステーション (workstation). (1) ユーザーが作業を行う機能装置。 ワークステーションは何らかの処理機能を持つものが多い。 (T) (2) ユーザーに作業を行わせるプログラム式または非プログラム式装置。(3) 端末またはマイクロコンピューターであって、通常、メインフレームまたはネットワークへ接続され、ユーザーがそこでアプリケーションを実行できるもの。

X

X.25. (1) データ端末装置とパケット交換データ・ネットワークとの間のインターフェースに関する、国際電信電話諮問委員会 (CCITT) の勧告。(2) *パケット交換 (packet switching)*も参照。

X.25 ネットワーク (X.25 network). ITU-TS 勧告 X.25 に従ってリンクされたパケット交換データ・ネットワーク。

Z

Z 時 (Z time). *ズールー時 (Zulu time)* の略称。*世界協定時 (coordinated universal time (UTC))* と同義。

ズールー時 (Zulu time (Z)). *世界協定時 (coordinated universal time (UTC))* と同義。

索引

日本語, 英字, 数字, 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]

[カ行]

［サ行］

［タ行］

［ハ行］

［マ行］

［ヤ行］

［ラ行］

［数字］

A

B

C

D

F

G

I

L

N

R

S

T

U

V

W

Printed in Japan

SC88-7727-00