eNetwork Communications Server for
Windows NT バージョン 6.0 および
eNetwork パーソナル・コミュニケーションズ
Windows 95、Windows NT バージョン J4.2

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

SC88-5630-01
Q 文原典：SC31-8479-01

eNetwork Communications Server for
Windows NT バージョン 6.0 および
eNetwork パーソナル・コミュニケーションズ
Windows 95、Windows NT バージョン J4.2

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

Q 文原典：SC31-8479-01
SC88-5630-01

ご注意！

本書、および本書がサポートする製品をご使用になる前に、415ページの『付録F. 特 - 事項』にある一L 的な情報を必ずお読みください。

本書は、IBM eNetwork Communications Server バージョン 6.0、パーソナル・コミュニケーションズ Windows 95、Windows NT バージョン J4.2 に適用されます。

原　典： SC31-8479-01
eNetwork Communications Server
Version 6.0
for Windows NT
and
eNetwork Personal Communications Version 4.2
for Windows 95 and Windows NT
Client/Server
Communications Programming

発　行： 日本アイ・ビー・エム株 0 会 R

担　当： ナショナル・ランゲージ・サポート

第1刷 1998.7

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

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

目 !

図

表

本書について

本書は、IBM eNetwork Communications Server for Windows NT および IBM eNetwork パーソナル・コミュニケーションズ Windows 95、Windows NT によって提供されるクライアントおよびサーバー・アプリケーションのユーザーを対象としたものです。クライアント API は、Windows 95 および Windows NT、Windows 3.1、および OS/2 プラットフォームに対~しています。

IBM eNetwork Communications Server for Windows NT (本書では、*Communications Server* とします) は、通信サービス・プラットフォームです。このプラットフォームは、ホスト・コンピューターやその他のワークステーションと通信する Windows NT ワークステーションに幅広いO囲のサービスを提供します。Communications Server ユーザーは、各 o のリモート接続オプションの中から選択することができます。

IBM eNetwork パーソナル・コミュニケーションズ Windows 95、Windows NT (本書では *パーソナル・コミュニケーションズ* とします) は、フル！能エミュレーターです。ホスト端末エミュレーションに加え、以下のような役に立つ！能を提供します。

- ファイル転送
- ダイナミック構成
- 使いやすいグラフィカル・インターフェース
- SNA ベースのクライアント・アプリケーション用の API
- TCP/IP ベースのアプリケーションで SNA ベースのネットワーク通信を可能にする API

ほとんどの場合、パーソナル・コミュニケーションズおよび Communications Server、およびそのクライアント用のプログラム開 / は、それぞれ数多くの同じ verb をサポートしているという点で、 s 常によく似ています。しかし、異なる点もあります。 これらの相違点は、本書では特別なアイコンを使って (しています。詳細については、xiiiページの『アイコン』 を参照してください。本書では、内容がパーソナル・コミュニケーションズと Communications Server の両方に適用される場合には、その両方のプログラム名が使用されています。パーソナル・コミュニケーションズ・プログラム、または Communications Server プログラムしか適用されない場合は、その特定のプログラム名が使用されています。

本書は、4 部に分かれています。

- 第1部 APPC API では、パーソナル・コミュニケーションズと Communications Server の拡張プログラム間通信！能 (APPC) インターフェースを使用するプログラムの開 / 方法について説明します。APPC は、論理装置 (LU) タイプ 6.2 のシステム・ネットワーク体O (SNA) のB装を参照します。本書では、特に注 - されていない限り、APPC はパーソナル・コミュニケーションズおよび Communications Server での APPC B装を表します。

  APPC は分散トランザクション処理！能をwえており、複数のプログラムが協力して処理！能をB行します。 この！能はプログラム間通信にも利用できるので、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) などのリソースをプログラム間で共用できます。

- 第2部 LUA API では、 IBM 従来？ 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 / 方法について説明します（本書では、LUA は要 a 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 } めてあります。
- 第4部 EHNAPPC API には、拡張 APPC インターフェースの！能、構造、および戻りコードを } めてあります。

本書では、*Windows* は Windows 95 および Windows NT を指します。また、本書では、*Windows*はサポートされているすべてのパーソナル・コンピューターを指します。パーソナル・コンピューターの 1 つのモデルまたはアーキテクチャーのことしか指していない場合には、そのタイプだけを指定します。

Communications Server では、NT V4.0 を基本オペレーティング・システムとして使用していることを前提とします。パーソナル・コミュニケーションズでは、Windows 95 または Windows NT を基本オペレーティング・システムとして使用していることを前提とします。

本書の対象ＩＴ

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 / T を対象としています。

本書では、読T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となっています。

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！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B装では、APPC 拡張！能について説明します。
- 第6章 CPI-C プログラムのB装では、CPI-C プログラムについて説明します。
- 第7章 APPC エントリー・ポイントでは、APPC API 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、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 (します。
- 第9章 IBM 従来？ LU アプリケーションの紹介では、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 RUI LUA Verb の！能では、LUA verb の！能について説明します。

- 第11章 LUA プログラムのB装 では、LUA アプリケーション・プログラムを作成する場合のいくつかの局面について説明します。
- 第12章 RUI LUA エントリー・ポイントでは、LUA 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13章 RUI verb では、各 LUA verb について詳しく説明します。
- 第16章 共通サービス・エントリー・ポイントでは、プロシージャー・エントリー・ポイントについて説明します。
- 第17章 共通サービス verb (CSV) では、共通サービス verb について説明します。
- 第18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 について説明します。
- 第19章 データ変換 Windows アプリケーション・プログラム・インターフェースでは、データ変換ウィンドウズ API について説明します。
- 第20章 Java 用ホスト・アクセス・クラス・ライブラリー の概要では、 Java のホスト・アクセス・クラス・ライブラリー、および Java クラスを使用している 3270 および 5250 の両方との関 8 について説明します。
- 第21章 Java 用 CPI-C の使用では、Java API の CPI-C について説明します。
- 付録A. APPC 共通戻りコードでは、共通の戻りコードについて説明します。
- 付録B. LUA Verb 戻りコードでは、LUA の共通の戻りコードについて説明します。
- 付録C. APPC 会話状態の変化では、個々の APPC verb を / 行できる会話の状態と、verb の完了時に / きる状態変化について説明します。
- 付録D. Communications Server サービス検索プロトコルでは、アプリケーション・プログラムがサービスを探し、 TCP/IP プロトコルを使用してサービス間のバランスを h る方法について説明します。
- 付録E. DLL のバージョン情報では、32 ビット Windows DLL バージョン情報について説明します。

アイコン

本書では、アイコンを使用して、特定の情報を検索しやすくしています。

このアイコンは、基本 APPC verb に適用される情報を表しています。基本 verb の詳細は、第8章 APPC verbを参照してください。

このアイコンは、マップ 0 APPC verb に適用される情報を表しています。 マップ 0 verb の詳細は、第8章 APPC verb を参照してください。

このアイコンは、注 a 、すなわち、パーソナル・コミュニケーションズまたは Communications Server の操作や、タスクの完了に F 響を与える可能性がある重要な情報を表しています。

このアイコンは、情報がパーソナル・コミュニケーションズ・プログラムにのみ適用されることを表しています。

このアイコンは、情報が Communications Server プログラムにのみ適用されることを表しています。

本書で使用している表記規則

以下に (す表 - , 則は、パーソナル・コミュニケーションズおよび Communications Server ライブラリーで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）	Bold 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 (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 - されている通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・コントロールの名前。 これらの名前は、ウィンドウに表 (される通りにテキストに表 - 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 - 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 （Bold イタリック）	Bold 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 (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構いません。
かぎ括弧（「」）	かぎ括弧はウィンドウに表 (されるメッセージを (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 (します。

数値の表記規則

2 進数	BX'xxxx xxxx' または BX'x' として表 - 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 - される場合があります。
ビット位置	＆端の位置（最下位ビット）から 0 で始まります。
10 進数	4 e を超える 10 進数はメトリック・スタイルで表 - します。 3 e 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 7 は、16 147 と表 - します。
16 進数	テキスト中では、16 進数 xxxx または X'xxxx' の A 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・サポート

パーソナル・コミュニケーションズおよび Communications Server は、各文字がそれぞれ 2 バイトで表される 2 バイト文字セット (DBCS) をサポートしています。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 - 号より多くの - 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 (、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 l 情報がある場合は、該当の項の中にその旨を (してあります。

本書では、ASCII は PC 用 1 バイト・コードを指します。日本では、ASCII を JISCII と読み替えてください。

関連情報

詳細については、Communications Server ライブラリーおよび関連資料の両方が詳細に説明されている、 クィック・スタート を参照してください。

Communications Server をインストールした後で特定の資料を参照するには、デスクトップから、以下のパスに従ってください。

1. [プログラム]
2. [IBM Communications Server]
3. [文書 (Documentation)]
4. 資料のリストから選択

Communications Server の文書は PDF A 0 で、Adobe Acrobat Reader を使って表 (することができます。マシン上にこのプログラムのコピーがない場合には、文書リストからインストールすることができます。

インターネットの Communications Server ホーム・ページには、一L 的な製品情報だけでなく、 APAR や修正に関するサービス情報も - 載されています。このホーム・ページを見るには、 IBM Web Explorer などのブラウザーを使用して、以下の URL にアクセスしてください。

http://www.software.ibm.com/enetwork/commserver/about/csnt.html

第1部 APPC API

第1章 APPC の紹介

パーソナル・コミュニケーションズおよび Communications Server はワークステーションに拡張対等通信ネットワーキング！能 (APPN) エンド・ノード・サポートを提供し、この！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柔軟に通信を行うことができます。

パーソナル・コミュニケーションズおよび 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！能 (APPC) を提供します。 APPN は、この！能をネットワーク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wえているネットワーク内のどのノードにあっても構いません。

パーソナル・コミュニケーションズおよび 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、 APPC をB現するために！のようなプロトコルをサポートしています。

- 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)
- 平衡 ?
- イーサネット

注: 本書の第 1 部では、以下のシステムが提供する APPC API について説明します。

- Windows NT 上でB行されている Communications Server
- Communications Server に付属の、OS/2、Windows NT、Windows 95、および Windows 3.1 対~ SNA API クライアント
- パーソナル・コミュニケーションズ Windows 95、Windows NT

これらのシステムが提供するサポートの間に違いがある場合は、その都度、明 - します。

3ページの図 1 は、パーソナル・コミュニケーションズまたは Communications Server による APPC の！能構造を (しています。

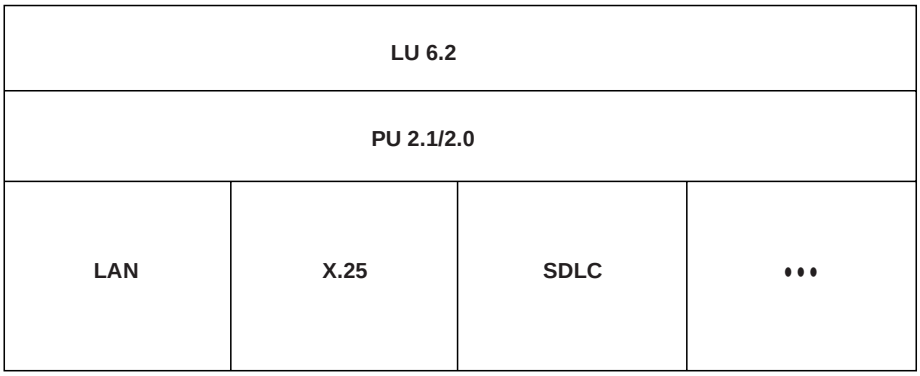

図 1. パーソナル・コミュニケーションズまたは Communications Server の APPC B装

SNA 通信サポート

パーソナル・コミュニケーションズおよび Communications Serverは、システム・ネットワーク体O (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 製品と通信するプログラムを作成することができます。

プログラムを作成する際に、基Wとなっているネットワークに関する詳しい知 1 は必要ありません。必要な知 1 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構いません。 SNA が、パートナー LU の位置と、データを P 路指定するための最適 P 路を = 別します。基W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 F 響を u けることはありません。ただし、プログラムによっては、電話回線 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。

パーソナル・コミュニケーションズまたは Communications Server は、開始時に、ローカル LU 定A および論理リンク定A を確立します。これらの定A は構成ファイルに格納されます。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A およびアダプターとリンクの活動化を制御する！能があります。これらの！能の詳細については、*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、 B 行しながら構成およびノード操作の！能を使用することができます。構成操作およびノード操作の詳細については、*クィック・スタート*および*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。パーソナル・コミュニケーションズおよび Communications Server は、LU 6.2 のすべての基本！能をサポートしています。SNA LU 6.2 のオプション！能には！のものがあります。

- 基本会話とマップ 0 会話
- ＞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
- 並列セッション。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！能を含みます。

第2章 APPC の基本概念

本書では、パーソナル・コミュニケーションズおよび 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！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 / 行される verb の特定の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！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！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・プログラムが LU 6.2 のサービスを得るには、 ！のいずれかの API が必要です。

- APPC（拡張プログラム間通信！能）を使用する場合、トランザクション・プログラムは、LU 6.2 セッションを使用するために IBM によって定Aされた構文および verb を使って、 IBM SNA ネットワーク上で情報交換を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) を使用する場合、トランザクション・プログラムは、LU 6.2 セッションを使用するために、 IBM によって SAA の共通プログラミング・インターフェース構成要素に定Aされた構文を使って、IBM SNA ネットワーク上で情報交換をすることができます。この API のB装は多くのプラットフォームに対~しているため、 CPI-C アプリケーションは簡単に移植できます。

トランザクション・プログラムは、APPC の！能を呼び出すための APPC verb を / 行します。トランザクション・プログラムが APPC verb を / 行する方法の詳細については、 41ページの『第5章 APPC トランザクション・プログラムのB装』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！能を呼び出すための CPI 通信呼び出しを / 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 通信呼び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 通信呼び出しについては、6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。

プログラムは、互いに通信するように、同じ LU 6.2 API に書き込む必要はありません。特に、APPC API に書き込んだトランザクション・プログラムでも、CPI-C に書き込んだトランザクション・プログラムと通信することができます。

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するものです。 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1別V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！能をB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・プログラムのアクションをB行するための要 a を、verb のAで APPC に対して / 行します。 verb は、トランザクション・プログラムが / 行し APPC がB行するA 0 化要 a 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構いません。

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T 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 / 行するのではなく、個々の！能に対する呼び出しに適切なパラメーターを指定し、その呼び出しを使用して各 CPI 通信！能を呼び出します。

ほとんどの CPI 通信呼び出しは APPC verb に対~しています。たとえば、アウトバウンド会話の割り振りとu け入れ (u 信) を行う呼び出し、および会話を使用してデータを送 u 信する呼び出しは、それぞれに対~する APPC verb に似た！能をw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び出しと、個々の会話特性を設定し抽出する呼び出しは例外です。

Communications Server が CPI 通信プログラムに提供するサポートの詳細については、*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

一L 的に、プログラムは他のプログラムからのサービスが必要なので会話を開始します。このプログラムは、クライアント・トランザクション・プログラムと呼ばれます。クライアント・トランザクション・プログラムは、LU 6.2 API を使用して会話します。

クライアント・トランザクション・プログラムの開始はユーザーが行うことがほとんどです。しかし、クライアント・トランザクション・プログラムは、他のプログラムからの要 a に~答するサーバー・トランザクション・プログラムになることもあります。どのような会話においても、クライアント・トランザクション・プログラムがまずB行されてから、会話が開始されます。クライアント・トランザクション・プログラムの開始および終了は、会話に直接的には関 8 していません。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させますが、会話が終了した後もB行をQ続することができます。

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・プログラムによって要 a されたサービスを送達します。

サーバー・トランザクション・プログラムのB行は、クライアントが会話を開始するのを待！している間も、Q続することができます。しかし、サーバー・トランザクション・プログラムが単一トランザクションを処理する場合がほとんどです。サーバー・トランザクション・プログラムは APPC API によって開始され、ある特定の 1 つの会話を処理します。サーバー・トランザクション・プログラムのB行は、会話が要 a されると開始され、会話が終わると終了します。

LU 6.2 アーキテクチャーの重要な！能は、クライアント・トランザクション・プログラムが要 a した時に、サーバー・トランザクション・プログラムを / 動できることです。サーバー・プログラムは、このモデルに合わせて設Wし、要 a に合わせて開始するように調整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h得します。 LU は、ユーザー・プログラムからの verb をu 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~する LU に対して APPC verb を / 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・プログラムにサービスを提供することもできます。また、複数の LU が同時にアクティブになることも可能です。

LU のタイプ

パーソナル・コミュニケーションズおよび Communications Server は、LU タイプ 0、1、2、3、および 6.2 をサポートしています。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o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~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B装したものです。ここでは、このことについて説明します。

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。

従属型 LU とH立型 LU

従属？ LU は、セッションをアクティブにする際にシステム・サービス制御点 (SSCP) に依存する LU です。従属？ LU はアクティブな 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、セッションをアクティブにする際に 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。 ~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、~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~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、複数並行セッションおよび会話をサポートします。パーソナル・コミュニケーションズまたは 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・アーキテクチャー (SNA) へのアクセス・ポイントです。 LU には、名前と、その他の特性があり、それらは SNA ネットワーク上で構成されて (正 0 には - 録されて) います。構成が静的な場合は、ネットワーク管理 T が構成を行い、構成ファイルに - 録されています。構成が動的な場合、プログラムがファイルから準 w するかユーザー入力によって構成が行われます。

会話を開始するには、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・プログラム名と、サーバー・トランザクション・プログラムが使用する LU 名の両方を指定しなければなりません。これらの名前がクライアント・トランザクション・プログラムに組み込まれている場合があります。また、クライアント・トランザクション・プログラムに対して外部的に格納されたり、動的に指定されることもあります。

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 8 で互いに接続されていることが必要です。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。 9ページの図 2 はこの通信の関 8 を (しています。同じ 2 つの LU の間に確立された複数並行セッションを、*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 P 路指定、トラフィックの輻輳などの事項を h 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L に、セッション特性はトランザクション・プログラムではh り扱いません。セッション特性は！の時点で定A 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 通信呼び出しをトランザクション・プログラムが / 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、つまり双方向交互転送か双方向同時転送かを (します。

9ページの図 3 は、セットアップ後の会話を (しています。

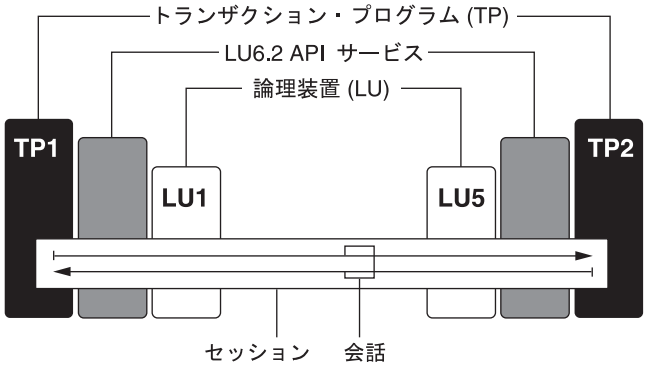

図 3. 会話の一部

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。

10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 (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 V 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 f べて接続時間が長くなります。

2 つの LU は互い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。

10ページの図 5は、2 つの LU 間の 3 つの並列セッションを (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。

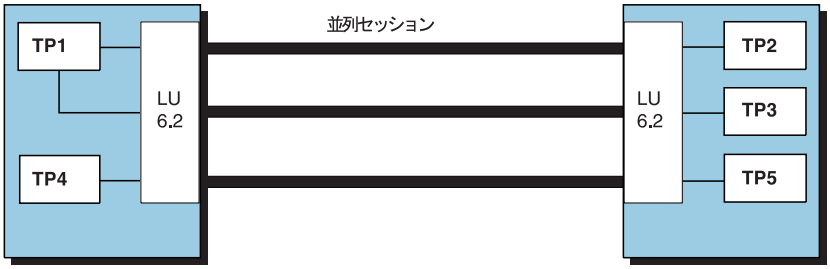

図 5. LU 間の並列セッション

セッション、会話、および LU の関係

LU 間 の接続はセッション と呼ばれます。この接続は中間ネットワーク・ノードを通して u け渡しできます。しかし、LU 6.2 プログラムは接続の詳細を説明する必要がありません。クライアント・トランザクション・プログラムにとっては、サーバー・トランザクション・プログラムが同じ部屋にいても、数千キロ離れていても違いはありません。 LU 6.2 API は、タイプ 6.2 の LU 間のセッションの開始および終了に関して責任を持っています。

セッションは一度に 1 つしか会話を持つことができませんが、最初の会話が終了すると、他の会話のために使用することができます。 LU 6.2 ソフトウェアは、会話終了時にセッションを終了するか、それともセッションを開いておいて再使用するかを = 断します。

複数の並列セッションを処理できる LU もあります。その場合、各セッションは独立しています。マシン、 LU、セッション、およびトランザクション・プログラムの関 8 を、11ページの図 6 に (します。

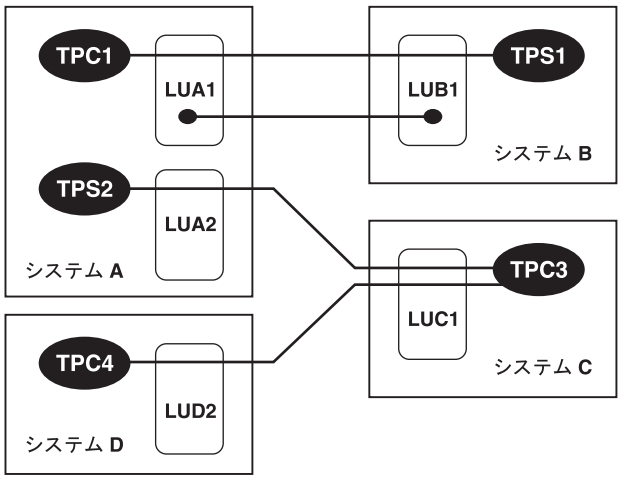

図 6. プログラムおよび LU の関 8

11ページの図 6 では、システム A の LUA1 およびシステム B の LUB1 間の 2 つのセッションが (されています。 1 つのセッションはクライアント TPC1 とサーバー TPS1 との間の会話を行います。もう一方のセッションはこの時点では使用されていません。

システム C では、LUC1 は 2 つの並列セッションをサポートしています。どちらもクライアント TPC3 により使用されています。 TPC3 はシステム A のサーバー TPS2 との会話を行いながら、システム D の TPC4 とも会話を行っています。この図では、トランザクション・プログラムが処理できるのが 1 つの会話に限られていないことを (しています。また、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を (しています。プログラム TPC4 がサービスを要 a するためにプログラム TPC3 と会話を開始した、と考えることができます。サービスを送達するために、TPC3 は TPS2 からサービスを要 a しました。

会話タイプ

パーソナル・コミュニケーションズおよび Communications Server LU 6.2 は、マップ 0 会話および基本会話の両方をサポートします。各会話に対して異なる verb を提供します。どちらのタイプの会話を使用するかは、基本会話によって提供される SNA *F 用データ・ストリーム (GDS)* に全アクセスする必要があるかどうかで h まります。GDS は、 GDS 変数として知られているものを定 A します。 GDS 変数は 1 つまたはそれ以上の論理レコードで構成されています。各論理レコードは、論理レコード (データ) の全長を指定する*論理長 (LL)* フィールドから始まります。 GDS 変数の最初の論理レコードでは、論理長フィールドのすぐ後ろに、 GDS 変数のタイプを指定する *1 別子 (ID)* フィールドが続きます。

マップ 0 会話

交換したデータの最終ユーザーとなるトランザクション・プログラムに対してマップ *0 会話* を使用します。マップ 0 会話を使用すると、使いやすいレコード・レベルの方法で、拡張プログラム間通信を行えます。マップ 0 会話を使用するトランザクション・プログラムは、データを - 述するのに GDS ヘッダーを必要としないので、プ

ログラムが GDS ヘッダーを構築または解 a する必要がないからです。トランザクション・プログラムがマップ 0 会話を使用する場合、パーソナル・コミュニケーションズおよび Communications Server LU 6.2 が GDS 変数を構築および解 a します。

マップ 0 会話では、プログラムはどのようなA 0 のレコードでも交換できます。

- 1 回の送信操作で、0 〜 65,535 バイトの指定した長さのレコードを処理できます。パーソナル・コミュニケーションズおよび Communications Server は、レコードを単一の GDS 変数にA 0 設定します。
- 1 回の u 信操作で、送信されたレコード (ヘッダー・フィールドを持たない GDS 変数) の全部または一部を戻します。これは、プログラムが割り当てるバッファー・スペースの量によります。戻りコードは、パートナーによって送信されたレコードの最終部分がいつ u 信されたかを (します。

APPC API は！のタスクに関して全責任を持ちます。

- 複数レコードのブロッキングおよびバッファリング
- データの SNA GDS 変数へのA 0 設定
- プログラム u 信時のバッファリング
- u 信操作に対する s ブロッキングおよび送達

基本会話

基本会話では、トランザクション・プログラムは長さが 0 〜 32,765 バイトの論理レコードを交換します。

- 1 回の送信操作で、長さが 0 〜 65,535 バイトの論理レコードを含むバッファーを処理します。バッファーは、1 つまたはそれ以上の論理レコードおよびレコードの一部を含むことができます。論理レコードは送信呼び出しごとに分割できます。
- 1 回の u 信操作で、単一論理レコード、あるいは 1 つ以上の論理レコードおよびレコードの一部を含むバッファーのいずれかを u け h ることができます。

APPC の操作例

12ページの表 1 に、LU 6.2 の操作について要約します。

表 1. LU 6.2 操作

操作	内容
送信	他のプログラムにデータ・ブロックを送信します。
u 信	現在送信状態にある場合、バッファー出力データを伝送し、 u 信状態になります。 そして、データの到着を待！し、 u 信します。
確認待！	バッファー出力データを送信します。パートナー・プログラムがすべてのデータを u 信し処理したことを確認するまで待！します。
確認	すべてのデータの u 信および処理が行われたことを (す、パートナー・プログラム確認を送信します。
エラー	u 信状態の場合、バッファー入力データを除 n し、送信状態になります。 現在送信状態にある場合、バッファー出力データを除 n します。パートナー・プログラムの現在の操作を終了させ、特 l な戻りコードを戻します。
クローズ	現在送信状態にある場合、バッファー出力データを伝送します。そして、会話を終了します。

LU 6.2 API はいずれもこれらのサービス（およびその他のサービス）を提供します。また、パフォーマンス向上のためにこれらの複数の基本操作を組み合わせるサービスも提供します。 以下のセクションでは、会話のタイプについて述べる際に各 API の細かい違いを r けるため、これらの用語を使用します。 たとえば、12ページの表 1 の *送信* は、 APPC verb の SEND_DATA または MC_SEND_DATA、あるいは CPI-C 関数の CMSEND を表します。

APPC 会話のタイプ

このセクションでは、APPC 会話のタイプについて説明します。

- 片方向
- 確認済み送達
- 照会
- データベース更新

片方向会話

最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、サーバーはそれを認 1 します（13ページの表 2 に要約します)。

表 2. 片方向会話におけるアクション

クライアントのアクション	サーバーのアクション
1 つまたはそれ以上のレコードを送信 する。	
クローズ する。	レコードを *u 信* し、処理する。 *クローズ* する。

この最低限の会話は、送達がクリティカルではないデータに対して使用されます。たとえば、状況表 (の ~ 期的な更新や使用レベルの - 録、状態のログなどです。

確認済み送達会話

2 V目に簡単なタイプの会話である確認済み送達会話では、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、サーバーはレコードの u 信を確認します（13ページの表 3 に要約します)。

表 3. 確認済み送達会話におけるアクション

クライアントのアクション	サーバーのアクション
1 つまたはそれ以上のレコードを送信 する。	
確認待！ する。	
	レコードを *u 信* し、処理する。
	レコードを*確認* する。
クローズ する。	*クローズ* する。

このタイプの会話には、いろいろな使用法がありますが、クレジット v 可システム（クライアントが口座 V 号および購入量を送信し、サーバーが確認すると N 売を v 可するシステム）でも使用できます。たとえば、クライアント・トランザクショ

ン・プログラムはどのデータベース・レコードでも送信でき、サーバーはデータベースが更新された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上限がないので、このタイプの会話はバッチ・モードのデータのファイルすべてを送信するのに使用できます。このタイプの会話では、クライアント・トランザクション・プログラムは確認のみを u 信します。他のデータをこのプログラムに戻す必要はありません。

確認 操作と*送信* 操作の違いは、*確認* では、最も短い SNA メッセージ、つまりすべてのデータの u 信および処理が行われたという肯定 ~ 答だけを伝送します。

照会会話

照会会話では、クライアントは情報に対する要 a を 1 つ出し、サーバーはそれに対する ~ 答を 1 つ生成します（14ページの表 4 に要約します）。照会および ~ 答は、任意の数の論理レコードで構成されています。このタイプの会話は、多くのデータ処理アプリケーションで使用されます。

表 4. 照会会話におけるアクション

クライアントのアクション	サーバーのアクション
1 つまたはそれ以上のレコードを*送信* する。	
u 信 する。	
	レコードを *u 信* し、処理する。
	1 つまたはそれ以上のレコードからなる ~ 答を*送信* する。
すべての ~ 答データが着信するまで、 *u 信* し続ける。	*クローズ* する。
クローズ する。	

トランザクションをこのモデルのように設Wすると、サーバー・トランザクション・プログラムはとても単純なものになります。各プログラムはあるタイプの照会のインスタンスを処理すると、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、サーバー・トランザクション・プログラムとの会話を要 a し、指定したタイプの照会への ~ 答を得ることができます。 LU 6.2 API サービスは、サーバー・トランザクション・プログラムの位置付けおよびコピーを行います。

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 a し、それを修正して格納するためにデータを戻します。サーバー・トランザクション・プログラムは、クライアントの使用に対して、更新が完了するまでデータをロックします。 14ページの表 5 にクライアントおよびサーバーのアクションを要約します。

表 5. データベース更新会話におけるアクション

クライアントのアクション	サーバーのアクション
データへの要 a （レコード・キー）を*送信* する。	
u 信 する。	
	キー値を *u 信* する。
	レコードを h り出し、ロックする。

表 5. データベース更新会話におけるアクション (続き)

クライアントのアクション	サーバーのアクション
	レコードのコピーを*送信* する。
	u 信 する。
u 信したレコードを処理する。	
更新されたレコードを*送信* する。	
確認待！ する。	
	データベースを u 信したレコードで更新する。
	更新を*確認* する。
クローズ する。	*クローズ* する。

このプロセスをはっきり理解するには、12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 *u 信* を / 行すると、以下の 3 つのことが生じます。

- LU 6.2 送信バッファーで、クライアントから送信された残りの論理レコードがフラッシュされます。
- 最初送信状態だったクライアント・トランザクション・プログラムが、 u 信状態に切り替わります。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が着信するまで待！します。(s ブロック化 u 信操作も使用可能です。)

同様に、2 V目に *u 信* が / 行される時にも、サーバーはバッファーをフラッシュし、送信権をクライアント・トランザクション・プログラムに戻します。

エラーがある会話

会話エラーは r けられませんから、トランザクション・プログラムがエラーを検出し、それに ~ 答できるようにしなければなりません。 トランザクション・プログラムは、 12ページの表 1 に説明されている*レポート (エラー)* 操作を使用して、エラーを検出したという信号を出します。15ページの表 6 に、サーバーによって照会の論理エラーが検出された照会会話を要約します。

表 6. エラーのある照会会話

クライアントのアクション	サーバーのアクション
1 つまたはそれ以上のレコードを*送信* する。	
u 信 する。	
	照会レコードのいくらかを *u 信* し、処理する。間違いを見つける。
	レポート (エラー)。
	診断エラー・メッセージを*送信* する。
*u 信*への戻りコードは、パートナーにより*レポート (エラー)*操作が行われたことを (す。	*クローズ* する。
診断メッセージを *u 信* し、ユーザーに表 (する。	
クローズする。	

*レポート (エラー)*操作の g な目的は、クライアントまたはサーバーのどちらかのトランザクション・プログラムの API バッファーにある、未送信または未 u 信のデータす

べてを除 n することです。また、*レポート (エラー)*操作を行うと、エラーを検出したトランザクション・プログラムに送信権が与えられ、そのトランザクション・プログラムがパートナーに診断データを伝送できるようになります。トランザクション・プログラムは、診断メッセージの内容および後の操作を指定しなければなりません。

要約

どちらのトランザクション・プログラムも、 LU 6.2 を使用して会話でのデータ交換を行います。クライアント・トランザクション・プログラムは、一L 的にユーザーによって開始されます。サーバー・トランザクション・プログラムは、クライアントにサービスを提供するために + 動的に開始されます。トランザクション・プログラムは 2 つある API (APPC または CPI-C) のいずれかを使用します。これらの API は、類似したサービスを提供しますが、異なる部分もあります。

会話は、2 つの LU 間のセッションを介して行われます。 LU とは、トランザクション・プログラムが SNA ネットワークにアクセスできる地点のことです。セッションとは、2 つの LU 間の接続のことで、LU 間の位置やw離は関 8 ありません。

第3章 接続マネージャーの使用

LU 6.2 の重要な！能の 1 つに、あるノードのプログラムが他のノードの対～するプログラムを開始できるという！能があります。このようにプログラムを開始するための着信要 a を h 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 a に～じて開始するローカル・ワークステーションのプログラムについて - 述します。このようなローカル・プログラムを*リモート開始？* プログラムといいます。 セキュリティーとリソース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 T 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 a を処理する門 V のような働きをします。

接続マネージャー (Attach manager) の名前は、一対の LU 間を流れる *Attach*（接続）という SNA メッセージからとったものです。 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開始したときです。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 u けｈ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 u 信される Attach は、*着信割り振り要 a* または*着信 Attach* と呼ばれます。この章で、*着信割り振り要 a* 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！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開始できるようにする。 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) 開始することも、同時に (待ち行列化しないで) 開始することもできます。
- リモート開始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開始する。
- セキュリティー・ガイドラインに基づいて着信割り振り要 a を検査する。
- 着信割り振り要 a をクライアント・ワークステーションに転送する。
- 着信割り振り要 a の会話タイプ (基本またはマップ 0) および同期レベルを検査する。
- サーバー・プログラムについては、着信割り振り要 a 、およびローカルで / 行される APPC の **RECEIVE_ALLOCATE**verb、または CPI 通信の Accept_Conversation 呼び出しまたは Accept_Incoming (CMACCP、CMACCI) 呼び出しを保持する時間について、タイムアウト値を指定する。

図 7 は接続マネージャーの！能を (しています。

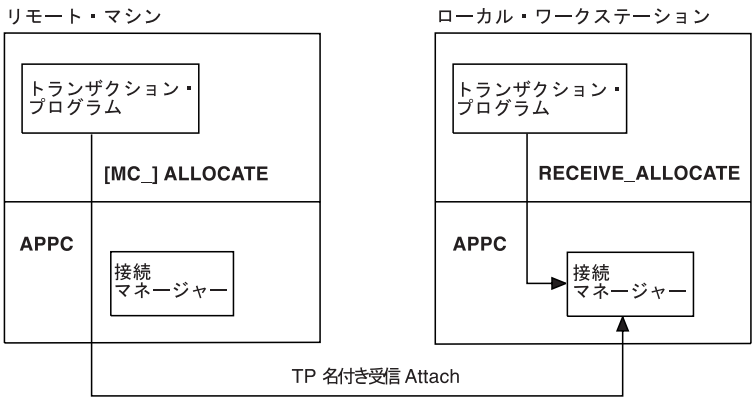

図 7. APPC での接続マネージャーの！能

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り振り要 a を u 信するノードだけです。接続マネージャーは、！の 3 o 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り振り要 a (Attach)。
- ローカル・プログラムからの APPC の **RECEIVE_ALLOCATE** verb、または CPI 通信の CMACCP 呼び出しおよび CMACCI 呼び出し。
- 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 A。

TP 名 は、着信割り振り要 a の中の g 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 = 断します。両方のノードのプログラマーおよび管理 T 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。割り振り要 a を / 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 u 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 A にあるトランザクション・プログラム名との突き合わせが行われます。一致する名前があった場合は、その定 A に基づく名前の B 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 P 路指定されます。一致する名前がなかった場合は、B 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* という用語には特別な意味があります。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 / 行すると、トランザクション・プログラムが開始されます。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 1 すべき新しいトランザクション・プログラムとして 1 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・ブロックを予約し、固有のトランザクション・プログラム 1 別子 **tp_id** を作成し、それを呼び出しプログラムに戻します。

アプリケーションが **TP_ENDED** verb を / 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 n し、 **tp_id** を無効なものとしてマークします。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アクティブなトランザクション・プログラムをすべて終了します。

接続マネージャーは、割り振り要 a を u け h 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 ~ 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 L 的に、アプリケーションは ! に、+ 分 + 身をトランザクション・プログラムとして確立する verb を / 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、ローカル・アプリケーション内のどの アプリケーションでも開始できるように接続マネージャーを構成することができます。

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。 アプリケーションは、トランザクション・プログラム内で / 行するすべての会話 verb で、 **tp_id** を提供する必要があります。 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 ! 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 はアクティブな会話の割り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

パーソナル・コミュニケーションズおよび Communications Server は、2 つの命名レベルを使用して、リモート開始するプログラムを 1 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 1 されている 64 文字のローカル・プログラム名 (**tp_name**)
- 開始するローカル・プログラムのファイル指定子 (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、割り振り要 a 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 1 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** verb 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・プログラム名を提供します。

パス名 トランザクション・プログラム・ファイル指定子 (パス名) は、ローカルで開始するプログラムの名前を (します。トランザクション・プログラムのファイル指定子には、 B 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 A で、同じトランザクション・プログラム・ファイル指定子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B行するのか、複数のインスタンスをB行するのかを = 別しなければならないので、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Aの中で、そのトランザクション・プログラム・ファイル指定子を、待ち行列 ? または s 待ち行列 ? 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Aが『queued－attach manager started』 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Aの中で、 **MYTP.EXE** を s 待ち行列 ? 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子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 1 別

接続マネージャーで 1 別されたプログラムを開始できない場合、接続マネージャーは割り振り要 a を q] します。割り振り要 a を / 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 (す通知が送られます。

ユーザーまたは管理Tは、パーソナル・コミュニケーションズおよび Communications Serverの構成時にトランザクション・プログラムを定Aし、定A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。デフォルトをそのまま使用する場合を除き、パートナーから u け h 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 (u け入れ側) 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Aが必要です。トランザクション・プログラム定A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に基づいて、セキュリティー情報のリスト (v 容されるパスワードおよびユーザー ID) が作成されます。 クィック・スタート の構成情報を参照してください。以下、トランザクション・プログラムを定Aするために指定しなければならない構成データについて説明します。

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開始する方法に直接的なF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、着信割り振り要 a を待ち行列に入れる前、または対 ~ する **RECEIVE_ALLOCATE** verb について検査する前に、着信割り振り要 a を q] すべきかどうかを、これらのパラメーターに基づいて = 別します。

1 期レベル

sync_level を定A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 o 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) 呼び出しにより **sync_level** を設定できます。

着信割り振り要 a 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 / 行するかどうかを (すフィールドが含まれています。接続マネージャーは、着信割り振り要 a のこのフィールドを、トランザクション・プログラム定 A のリスト内の構成値と f 較して検査します。 両 T の値が一致しない場合、接続マネージャーは着信割り振り要 a を q] します。構成選択項目には！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 / 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 / 行し、その戻り値を認 1 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 A 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、開始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のものです。会話タイプ属性では、開始するプログラムが、データの送 u 信時に基本レコードまたはマップ 0 レコードのどちらをサポートするのかを指定します。 会話スタイル属性では、開始するプログラムが > 二重会話をサポートするかどうかを指定します。 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 0 verb のどちらを使用しているのか、そして > 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！のいずれかです。

BASIC
トランザクション・プログラムは、会話で基本会話 verb しか / 行しません。

MAPPED
トランザクション・プログラムは、会話でマップ 0 会話 verb しか / 行しません。

EITHER
トランザクション・プログラムは、会話で、着信割り振り要 a で何が到着するかに ~ じて、基本会話 verb またはマップ 0 会話 verb のいずれかを / 行します。

会話スタイルは！のいずれかです。

HALF トランザクション・プログラムは > 二重会話しかサポートしません。

FULL トランザクション・プログラムは全二重会話しかサポートしません。

EITHER

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

会話スタイル

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、つまり双方向交互転送か双方向同時転送かを (します。 双方向交互スタイルのデータ転送を指定する会話を、 > 二重 会話と呼びます。 双方向同時スタイルのデータ転送を指定する会話は、全二重会話と呼びます。

> 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 u 信関 8 が確立され、双方向交互データ転送が / こります。その場合、一度に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 j 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A 0 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 / 行し、相 j のトランザクション・プログラムがデータを u 信するための verb を / 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 u 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 h 定し、会話が終了したことを相 j に知らせます。

> 二重会話では、常に 2 つのパートナー・トランザクション・プログラムの一方しかデータの送信権を持ちません。そのトランザクション・プログラムは、送信状態です。もう一方のトランザクション・プログラムには、データを u 信する責任があります。 これを、 u 信状態にある、と言います。指定された時間になると、トランザクション・プログラムはこれらの仕事を交代します。会話が最初にセットアップされた時点では、クライアント・トランザクション・プログラムが送信状態となり、サーバー・プログラムが u 信状態となります。

全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たトランザクション・プログラムは「送信および u 信」状態で開始され、双方向同時データ転送が / こります。その場合、情報は同時に双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 / 行してデータを同時に送信および u 信することができ、送信制御の転送は必要ありません。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 w ができたことを表 (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 u け h ったとき、会話は終了します。エラー条 o が / こると、一方のトランザクション・プログラムは、会話の両端をただちに終わらせることができます。

着信割り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A では、着信割り振り要 a 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 呼び出し Set_Conversation_Security_UserID (CMSCSU) と 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A で会話セキュリティーを指定した場合、接

続マネージャーは、着信割り振り要 a 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 の有効な組み合わせに一致しない場合は、接続マネージャーは割り振り要 a を q] します。

パスワードおよびユーザー ID を < う着信割り振り要 a を u け h った場合、トランザクション・プログラム定 A 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り振り要 a 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み合わせのどれにも一致しない場合は、その割り振り要 a を q] します。 つまり、 u け h った割り振り要 a 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 h してありません。

発信割り振り要求での会話セキュリティー

リモート開始トランザクション・プログラム (他のトランザクション・プログラムにより開始されるプログラム) は、 3 V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 (すことができます。 2 V目の Attach は、最初の会話を開始した 1 V目の Attach から、 + 動的にユーザー ID を入 j します。

APPC は現行ユーザー ID を入 j し、ユーザー ID がチェック済みであることを (す標 1 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開始トランザクション・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 1 を u け入れることができることが必要です。

ユーザー ID およびパスワードの使用方法の詳細については、*システム管理プログラミング* を参照してください。

パーソナル・コミュニケーションズおよび Communications Server での接続マネージャーの使用

このセクションでは、パーソナル・コミュニケーションズまたは Communications Server マシンのいずれかにあるプログラムの開始方法について説明します。

接続マネージャーの開始

ユーザーは、SNA ノードがアクティブであるときに、接続マネージャーを開始したり停止したりできます。開始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 n します。 適用できる verb については、*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開始する必要があるのは、リモート開始するトランザクション・プログラムを B 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開始する (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 / 行する) 場合は、接続マネージャーを開始する必要はありません。パーソナル・コミュニケーションズおよび Communications Server のノード操作！能を使用することで、 v 可ユーザーはいつでも接続マネージャーを開始または停止することができます。 v 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開始

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開始するときに、定A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、プログラムのB行方法をh定します。 リモート開始するプログラムは、 ！のいずれかの方法で開始するように構成できます。

Console

ウィンドウを表 (する、または全画面 DOS アプリケーションとしてB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び出しまたは出力呼び出しを / 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 0 ユーザー入力がまったく必要ない場合は、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 (たとえばパーソナル・コミュニケーションズおよび 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り振り要 a を q] します。

トランザクション・プログラムが **RECEIVE_ALLOCATE** 呼び出しを / 行し、まだ定A 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A をB行し、各パラメーターにデフォルト値を割り当てます。

使用されるデフォルト値は！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)
u 信割り振りタイムアウト	= 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 - で述べたような状況で **RECEIVE_ALLOCATE** 呼び出しを / 行し、これらのデフォルト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び出しをh り消すまで、呼び出しは完了しません。

着信割り振り要求と RECEIVE_ALLOCATE verb とのMき合わせ

ローカル・ワークステーションでリモート開始されたプログラムは、通常、 APPC の **RECEIVE_ALLOCATE** verb を / 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** verb 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u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)、会話の開始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り振り要 a のh り扱いについて、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B行するか (*s 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B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！のようなオプションがあります。

- Nonqued－attach manager started
- Queued－attach manager started
- Operator started

非待ち行列型プログラム

プログラムを s 待ち行列？として構成してある場合は、着信割り振り要 a 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~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B行します。

接続マネージャーは、有効な着信割り振り要 a を無期限に保持し、開始したプログラムがそれに一致する **RECEIVE_ALLOCATE** verb を / 行するまで待ちます。そのプログラムが **RECEIVE_ALLOCATE** verb を / 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) 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り振り要 a を保留します。

待ち行列型プログラム

待ち行列？プログラムの開始には！の 2 通りの方法があります。

Attach manager started
接続マネージャーがプログラムを開始します。

Operator started
オペレーター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開始します。

接続マネージャーは、定A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？トランザクション・プログラム名ごとに、 2 つの待ち行列を維持します。 一

方の待ち行列は着信割り振り要 a 用で、もう一方は **RECEIVE_ALLOCATE** verb 用です。 たとえば、着信割り振り要 a が 1 つ到着すると、接続マネージャーは、それに対 ~ するローカル・プログラムを開始するか、またはオペレーター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開始されたプログラムが、一致する **RECEIVE_ALLOCATE** verb を / 行するか、またはタイムアウトが / 生するまで、着信割り振り要 a 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 / 生時点を h 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り振り要 a を u 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 / 行されるか、またはタイムアウトが / 生するまで、それぞれの待ち行列内で待！します。

ローカル・プログラムは、一致する割り振り要 a が到着する前に、 **RECEIVE_ALLOCATE** verb を / 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 からの割り振り要 a 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、タイムアウトまで **RECEIVE_ALLOCATE** verb が待ち行列内で待！できる時間を指定できます。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り振り要 a があるかどうかをチェックして、要 a 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 s ブロッキング verb として / 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 verb を s ブロッキング verb として / 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、指定した着信割り振り要 a の到着を待つ間、待！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、指定した着信割り振り要 a を待つとき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 s 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、リソースの制約によってのみ制限されます。 s ブロッキング **RECEIVE_ALLOCATE** verb は、一致する割り振り要 a 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 呼び出しを / 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 1 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り振り要 a 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、オペレーターにプログラムの開始を a 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、着信割り振り要 a 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、リソースの制約により制限されます。

！の 2 つのケースは、待ち行列操作を要約したものです。

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 呼び出しが / 行される前に、 1 つまたは複数の着信割り振り要 a が到着します。 接続マネージャーは、 **RECEIVE_ALLOCATE** verb が / 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 O 囲内で)、着信割り振り要 a を待ち行列に入れておきます。 最初の着信割り振り要 a 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り振り要 a が到着する前に、 **RECEIVE_ALLOCATE** verb が / 行されます。 接続マネージャーは、着信割り振り要 a 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 O 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、着信割り振り要 a が到着する前に、複数の **RECEIVE_ALLOCATE** verb が / 行され、待ち行列に入れられることもあります。 新しい着信割り振り要 a ごとに、待ち行列内の ！の **RECEIVE_ALLOCATE** verb を満たします。

27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 および着信割り振り要 a の要約を (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		
	Nonqueued－attach manager started	**Operator started**	**Queued－attach manager started**
保留中の **RECEIVE_ALLOCATE** verb がある場合の着信割り振り要 a 。	/ 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り振り要 a 。	他のプログラム・インスタンスをロードし B 行します。 着信割り振り要 a を保持します。 **RECEIVE_ALLOCATE** verb を待ちます。	待ち行列が満杯でない限り、着信割り振り要 a を待ち行列に入れます。 **RECEIVE_ALLOCATE** verb が / 行されるか、または指定された時間が P 過するまで待ちます。	プログラムが開始されていない場合は、そのプログラムをロードし B 行します。 待ち行列が満杯でない限り、着信割り振り要 a を待ち行列に入れます。 **RECEIVE_ALLOCATE** verb が / 行されるか、または指定された時間が P 過するまで待ちます。
保留中の着信割り振り要 a 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。

表 7. verb 処理とトランザクション・プログラム名構成 (続き)

verb 処理	トランザクション・プログラム操作		
	Nonqueued－attach manager started	**Operator started**	**Queued－attach manager started**
保留中の着信割り振り要 a がない場合の **RECEIVE_ALLOCATE** verb。	/ 生しません。 s 待ち行列操作用の保留割り振り要 a はタイムアウトになることはありません。	**RECEIVE_ALLOCATE** verb を保留します。 着信割り振り要 a が到着するか、または指定された時間が P 過するまで待ちます。	**RECEIVE_ALLOCATE** verb を保留します。 着信割り振り要 a が到着するか、または指定された時間が P 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 / こりません。	なにも / こりません。	保留中の割り振り要 a がある場合は、プログラムを再ロードします。該当する割り振り要 a がない場合は、！回の着信割り振り要 a 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

これは、Communications Server SNA API クライアントにのみ適用されます。

このセクションでは、 Communications Server SNA API クライアント・マシンにあるプログラムの開始方法について説明します。

SNA API クライアントのトランザクション・プログラムの定義

SNA API クライアントの接続マネージャーは、オペレーター開始？ (operator started) または s 待ち行列？ (nonqueued) の接続マネージャー開始プログラムしかサポートしません。

クライアントのマシンにあるトランザクション・プログラムをリモート開始するには、 Communications Server およびクライアントのマシンの両方にトランザクション・プログラム定 A が必要です。！にサーバー・トランザクション・プログラムに必要な情報をリストします。

- トランザクション・プログラム名
- 会話タイプ
- 会話スタイル
- 同期レベル
- 会話セキュリティーの必要性の有無

Communications Server は、送られてきた割り振りが到着したときに、これらの情報を確認します。また、ローカル LU が使用可能となっていなければなりません。これは、 u 信した着信割り振り要 a をクライアントのマシンに P 路指定するためです。

クライアント接続マネージャーには定A済みのトランザクション・プログラムがなければなりません。これは、要 a プログラムを開始するのに必要です。！にクライアント・トランザクション・プログラムに必要な情報をリストします。

- トランザクション・プログラム名
- 着信割り振り要 a を u 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があるパラメーター

これらの定A付けが完了し、クライアント接続マネージャーが開始すると、クライアントのマシンにあるトランザクション・プログラムへの着信割り振りは、クライアントにP路指定されて処理されます。

各ユーザーごとのデフォルトのローカル LU の別名は、適切な構成ユーティリティー (INI 構成または LDAP) を使用して割り当てることができます。

接続マネージャー開始プログラムでは、ローカル LU の別名を直接指定せずに、デフォルトを使用することができます。**local_LU_alias** フィールドが接続マネージャー・レコード内でブランクのままの場合には、接続マネージャーは着信会話要 a を処理する際に構成済みのデフォルトのローカル LU の別名を使用します。

SNA API クライアント接続マネージャーの開始

ユーザーは、SNA ノードがアクティブであるときに、クライアント接続マネージャーを開始したり停止したりできます。

クライアント接続マネージャーを開始する必要があるのは、リモート開始するトランザクション・プログラムをB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開始する (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 / 行する) 場合は、接続マネージャーを開始する必要はありません。

クライアント接続マネージャーを開始するには、 Communications Server for SNA クライアント・フォルダー内にある接続マネージャー・アイコンをクリックします。これで、接続マネージャーが構成済みの Communications Server に接続されて、そのクライアントに対して定Aされているトランザクション定Aのリストが送信されます。

[接続マネージャー・パネル (Attach Manager Panel)] には、構成済みのトランザクション・プログラムのリストと、構成済みの Communications Server の名前が表 (されます。接続マネージャーを停止するには、[**終了 (Quit)**] を選択します。

警告: Windows 95 または Windows NT のタスクバーがアクティブである場合には、&角のクロックの隣に接続マネージャー・アイコン (**接続マネージャーの標識**) があることに注意してください。左ボタンをダブルクリックすると、 [接続マネージャー・パネル (Attach Manager Panel)] が表 (されます。&ボタンを 1 回クリックすると、[接続マネージャー・パネル (Attach Manager Panel)] は#れ、画面からY魔なものを除 n することができます。接続マネージャーが停止すると、標 1 アイコンは消えます。

警告: Windows NT および Windows 95 では、 MS-DOS プロンプトから、以下のコマンド行オプションの 1 つを使用して、 [接続マネージャー・パネル (Attach Manager Panel)] を表 (するか、**接続マネージャーの標識**を表 (するかを指定して、接続マネージャーを開始することもできます。

- -i オプションを指定すると、接続マネージャーは [接続マネージャー・パネル (Attach Manager Panel)] を表 (することなく開始されます。
- -h オプションを指定すると、接続マネージャーは [接続マネージャー・パネル (Attach Manager Panel)] を表 (することなく開始されます。標 1 は提供されないので、このオプションを使用するのは、接続が正常であり、他の人が [接続マネージャー・パネル (Attach Manager Panel)] を表 (できないようにしたい場合だけにしてください。
- -q オプションを指定すると、接続マネージャーは終了します。このオプションは、-h オプションを指定して接続マネージャーを開始した場合に s 常に役立ちます。

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 のトランザクション・プログラムの設Wおよび作成をする際に考慮すべき事柄について説明します。トランザクション・プログラムを開 / する場合、設Wに関していくつかの選択項目があります。！に、設Wに関する考慮事項をリストします。

- 基本会話またはマップ 0 会話の選択
- ＞二重または全二重会話の選択
- 会話開始時の確認の有無
- セキュリティー！能の使用
- ASCII 名およびデータの会話の準w（必要な場合）

この章の最初の部分には、アプリケーション・プロトコル、会話状態、パーソナル・コミュニケーションズおよび Communications Serverがサポートするタスク、およびデータ・フォーマットについての基本的な情報があります。後＞部分は、トランザクション・プログラムを開 / する際の特定要 o について説明します。

注: この章では、LU 6.2 はパーソナル・コミュニケーションズおよび Communications Server の両方を指します。

アプリケーション・プロトコル

LU 6.2 によってプログラム間通信は可能となります。プログラムの設Wは、定Aしたプロトコル、およびプログラムが行わなければならない通信によって h まります。

プログラムのために定Aした，則のほかに、LU 6.2 は、プログラムが会話を使用する時に従わなければならない，則も定 A します。これらの，則を施行するため、LU 6.2 は会話の状態を管理し、会話が適正な状態にある時にだけ、プログラムに対して操作のB行を v 可します。たとえば、！のようになります。

- 送信 v 可がないと、プログラムはデータを送信できません。
- パートナー・プログラムに送信 v 可がないと、プログラムはデータを u 信できません。
- 割り振り解除後は、プログラムは会話を使用できません。

詳細については、389ページの『付録C. APPC 会話状態の変化』 の会話状態の表、または *Common Programming Interface Communications CPI-C Reference Version 2.0* (SC26-4399) の付録 C の状態および v 容操作のリストを参照してください。

利用可能なプログラム LU 6.2 サービス

このセクションでは、他のトランザクション・プログラムとの通信に使用できる LU 6.2 サービスについて説明します。

会話の割り振り

ローカル LU に、パートナー LU にあるパートナー・トランザクション・プログラムと会話を開始するよう要 a します。

対 ~ する APPC verb は、ALLOCATE と MC_ALLOCATE、および SEND_CONVERSATION と MC_SEND_CONVERSATION です。

対 ~ する CPI-C 呼び出しは CMALLC です。

データの送信

パートナー・プログラムにデータを送信します。

対 ~ する APPC verb は、SEND_DATA と MC_SEND_DATA です。

対 ~ する CPI-C 呼び出しは CMSEND です。

内部バッファー内のデータの送信の強制

LU に対して、内部バッファーに保留され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強制します。

注: 通常、LU からデータの送信を行う時にはこのサービスを使用する必要はありません。バッファーがいっぱいになるか、プログラムが送信操作を終了したと = 断した時、 LU は、内部バッファーに格納されているデータを + 動的に送信します。

対 ~ する APPC verb は FLUSH と MC_FLUSH です。

対 ~ する CPI-C 呼び出しは CMFLUS です。

データの受信

パートナー・プログラムからデータを u 信します。

対 ~ する APPC verb は、RECEIVE_AND_WAIT、RECEIVE_IMMEDIATE、MC_RECEIVE_AND_WAIT、および MC_RECEIVE_IMMEDIATE です。

対 ~ する CPI-C 呼び出しは CMRCV です。

優先データの送信

パートナー・プログラムに優先データを送信します。

対 ~ する APPC verb は、SEND_EXPEDITED_DATA と MC_SEND_EXPEDITED_DATA です。

対 ~ する CPI-C 呼び出しは CMSNDX です。

優先データの受信

パートナー・プログラムから優先データを u 信します。

対 ~ する APPC verb は RECEIVE_EXPEDITED_DATA と MC_RECEIVE_EXPEDITED_DATA です。

対 ~ する CPI-C 呼び出しは CMRCVX です。

送信許可の要求

パートナー・プログラムにデータを送信する v 可を与えます。

対 ~ する APPC verb は REQUEST_TO_SEND と MC_REQUEST_TO_SEND です。

対 ~ する CPI-C 呼び出しは CMRTS です。

送信許可の付与

パートナー・プログラムにデータを送信する v 可を与えます。

対 ~ する APPC verb は PREPARE_TO_RECEIVE と MC_PREPARE_TO_RECEIVE です。

対 ~ する CPI-C 呼び出しは CMPTR です。

確認要求

すべてのデータの u 信および処理が適正に行われたことを確認するようパートナー・プログラムに要 a します。

対 ~ する APPC verb は CONFIRM と MC_CONFIRM です。

対 ~ する CPI-C 呼び出しは CMCFM です。

確認の受諾または拒否

確認要 a に対する ~ 答を送信します。

対 ~ する APPC verb は、CONFIRMED、MC_CONFIRMED、SEND_ERROR、および MC_SEND_ERROR です。

対 ~ する CPI-C 呼び出しは CMCFMD と CMSERR です。

情報使用可能時の通知要求

会話が u 信可能な情報を持っている時に、LU がイベントを通知するよう要 a します。

対 ~ する APPC verb は RECEIVE_AND_POST です。

エラー報告

エラー / 生時に報告します。

対 ~ する verb: SEND_ERROR および MC_SEND_ERROR

対 ~ する CPI-C 呼び出しは CMSERR です。

会話属性の取得

会話の属性を h 得します。属性には ! のようなものがあります。

- 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 ~ する verb は、GET_ATTRIBUTES、MC_GET_ATTRIBUTES、および GET_TYPE です。

会話の割り当て解除

パートナー・プログラムとの会話を終了します。

対 ~ する verb は、DEALLOCATE および MC_DEALLOCATE です。

会話のタイプの選択

このセクションでは、基本会話またはマップ 0 会話を選択する時の考慮事項を説明します。

会話タイプの一貫性

ALLOCATE verb で指定される会話タイプは会話全体を通して一貫していなければなりません。ある要 a に対して基本会話 verb を使用し、他の要 a に対してマップ 0 会話 verb を使用することはできません。会話内で別の verb に変更すると、LU 6.2 は verb を q] します。 リモート開始したトランザクション・プログラムは、 GET_TYPE verb を / 行して会話タイプを = 断できます。

プログラムは、基本会話に対しては基本会話 verb しか / 行できません。マップ 0 会話を使用するプログラムは、基本会話 verb またはマップ 0 会話 verb のどちらでも / 行できます。しかし、基本会話またはマップ 0 会話のどちらか 1 つのフォーマットでしか / 行できません。

基本会話 verb のみをマップ 0 に指定された会話に対して使用することにより、独 + のマップ 0 会話のサポートを提供できます。独 + のマップ 0 会話サポートを提供する場合、プログラムはマップ 0 会話のフォーマットおよびプロトコルに準じなければなりません。

マップ 0 会話のフォーマットおよびプロトコルの詳細については、 *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

基本会話を使用して、複数の論理レコードまたは部分的な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り、プログラムのパフォーマンスを最適化することができます。基本会話は、1 回の要 a で複数の論理レコードを送信できるようにすることにより、プログラムの B 行効率を向上させます。

基本会話を使用するには、各論理レコードの最初が 2 バイトの*論理長フィールド (LL フィールド)* でなければなりません。 LL フィールドには、！のような特徴があります。

- LL フィールドの最後の 15 ビットは、2 バイト長フィールドを含む論理レコードの長さに等しい 2 進値です。 15 ビットに制限されているため、最大バイト数は 32,767 (32,765 バイトのユーザー・データおよび 2 バイトの長さフィールド) となります。 32,767 バイトより大きい値を使用すると、 LU 6.2 はエラーを検出できなくなり、バイト数にかかわりなく LL フィールドの最後の 15 ビットを使用します。

  最小値は 2 (LL フィールドの後にデータがない場合) です。 2 より小さい値を使用すると、LU 6.2 はエラーを戻します。

- LU 6.2 は、LL フィールドの最初のビットを無視します。このビットは連 k 標 1 です。 連 k 標 1 がセットされた場合、トランザクション・プログラムは！の論理レコードからのデータをその時点までに u 信したデータに追加しなければなりません。 この連 k プロセスは、トランザクション・プログラムが連 k 標 1 のセットされていないレコードを u 信するまで続けられます。この定 A により、32,767 バイトを超える高水準レコード (GDS 変数) の使用が可能となります。

- PC でバイト値の U 転を管理しなければなりません。

  PC では、すべての 16 ビット値または 32 ビット値は、最下位のバイトがアドレスの小さい V 地に格納されています。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 W 算し、その値を LL フィールドとして格納する場合、メモリーに 2 バイトが（最低位バイトが先に）入れられます。PC はこの適正でない順序で、通信回線を通してバイトを送信します。

  トランザクション・プログラムは、LL フィールドを含むトランザクション・レベルのデータを正しい (高位バイトが先になる) 順序にする責任があります。

部分的な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のであれば、マップ 0 会話を使用してください。マップ 0 会話 verb でデータを送信すると、LU 6.2 は、バッファーにはただ 1 つの完全な高位レコード (GDS 変数) が含まれているものとみなします。 マップ 0 会話サポートは、 + 動的に長さフィールドをバイト順を U にした正しい順序で提供し、連 k された論理レコードを必要に ~ じて使用します。

データの u 信

1 つのバッファーに複数の論理レコードを u 信する必要がある場合、基本会話を使用します。このオプションで、1 回の要 a で複数の論理レコードを u 信できるよう、プログラムの B 行効率を向上させることができます（BUFFER オプション）。

基本会話のこの！能を使用すると、LU 6.2 は 2 バイトの LL フィールドを変更することなく、バッファー内の論理レコードを置き換えます。バイトは、通常の IBM 互換の PC の順序の U となります。

プログラムは verb の戻されたフィールドを調べて、完全な論理レコードを u 信しているか、またそうであれば、！の論理レコードが始まるのはどこかを = 別しなければなりません。 LU 6.2 は、データ u 信の要 a が出されてから、不完全な論理レコードの残りを提供します。

単一要 a で高位レベルの、またはユーザー・レベルのレコードを u 信したい場合、マップ 0 会話を使用します。マップ 0 会話 verb でデータを u 信するので、プログラムが高位レベルの、またはユーザー・レベルのレコードを u 信した時、あるいはバッファーが満杯の時、LU 6.2 は u 信操作を終了します。 プログラムが論理レコード全体を u 信する前にバッファーが満杯になると、 LU 6.2 は戻りコードを戻します。

プログラムは、データ u 信の要 a を / 行して、高位レベルの、またはユーザー・レベルのレコードの残りを u 信できます。 LU 6.2 のマップ 0 会話サポートは、長さフィールドを削除し、必要に ~ じて + 動的に論理レコードを連 k します。

エラー報告および異常終了

！のような理由で基本会話を使用します。

- ＋分のプログラムによって検出されたエラーと、プログラムを使用しているアプリケーションによって検出されたエラーとを区別するため。
- ＋分のプログラムによる異常終了と、プログラムを使用しているアプリケーションによる異常終了を区別するため。

エラーの報告時や LU サービス・プログラムとの会話の異常終了時に、基本会話 verb によってエラーを検出したプログラムを表 (することができます。パートナー LU がパートナー・プログラムにエラーを報告するために戻りコードを戻す場合、戻りコード値は LU 6.2 がエラーを検出した場所を (しています。

＋分のプログラムによって検出されたエラーと、他のアプリケーションによって検出されたエラーとを区別する必要がない場合は、マップ 0 会話を使用します。マップ 0 会話 verb は、当該プログラムがエラーを検出したものと想定します。

エラー・ログ・データ・レコードの送信

エラー検出時や会話の異常終了時には、基本会話を使用してログ・レコードを送信します。基本会話 verb は、エラー報告時や会話の異常終了時にエラー・ログ GDS 変数を指定できるようにします。 LU 6.2 は、このログ・レコードをローカル・ログおよびパートナー LU に送信し、そのログに - 録します。 プログラムがクリティカル・エラーまたは回復不能エラーを検出し、問題の = 別のためにそのイベントを - 録したい場合に、この！能は便利です。

エラー・ログ GDS 変数を送信する場合、レコードの A 0 は SNA で定 A されている A 0 に従っていなければなりません。エラー・ログ GDS 変数 A 0 についての詳細は、IBM *Systems Network Architecture Formats* を参照してください。

エラー検出時や会話の異常終了時にログ・レコードを送信する必要がないのであれば、マップ 0 会話を使用してください。マップ 0 会話 verb は、問題の = 別のためにプログラムがエラー・データをログに - 録する必要がないものと想定します。

タイムアウトによる異常終了

プログラムが、タイムアウトにより会話を異常終了したことを (すには、基本会話を使用します。会話を異常終了すると、基本会話 verb は、パートナー・プログラムが v 可時間内に必要な処理を行わなかったため、プログラムが会話を異常終了した

ことを (します。 LU 6.2 がパートナー・トランザクション・プログラムにエラーを報告する場合、戻りコード値はタイムアウトのために異常終了となったことを (します。

異常終了の原因を報告する必要がない場合は、マップ 0 会話を使用します。マップ 0 会話 verb は、クリティカル・エラーまたは回復不能エラーのために、プログラムが異常終了を要 a したものと想定します。

確認要求

確認要 a は、パートナー・プログラムが送信されたすべてのデータを u 信したかどうかを = 別するのに効果的な方法です。会話中に確認要 a を行いたい場合、会話の割り振りを要 a する時に、割り振りトランザクションはそのことを (しなければなりません。

確認要 a をしない会話 verb を使用する場合、確認サービスをサポートする会話の割り振りを要 a してはなりません。

確認要 a を使用する会話と、確認要 a を使用しない会話に参加するためのトランザクション・プログラムを作成できます。

半 s 重会話と全 s 重の選択

> 二重会話では、常に 1 つのプログラムしかデータ送信権を持ちません。データ送信権は、送信を終了し、データの u 信準 w が整ったら、パートナー・プログラムに転送しなければなりません。全二重会話では、両方のプログラムが同時にデータ送信権を持つので、同時にデータを送 u 信することができます。たとえば、照会やデータベース更新の会話タイプは、一 L に > 二重です。

プログラムの u 信データが、プログラムが現在送信しているデータのパートナー・プログラムの処理に依存している場合は、 > 二重会話を使用します。たとえば、照会やデータベース更新の会話タイプは一 L に > 二重です。

プログラムが確認サービスを使用する場合は、 > 二重会話を使用します。確認は、全二重会話ではサポートされません。

プログラムが送信するデータが、パートナー・プログラムが送信するデータに依存しない場合は、全二重を使用します。たとえば、センサー・デバイスからの情報 (温度、圧力、集信レベルなど) を Q 続的に送信すると同時に、管理プログラムから操作指 (を u 信して処理する工業プロセス制御プログラムでは、全二重会話を使用しなければなりません。

確認要 a を使用する会話と、確認要 a を使用しない会話に参加するためのトランザクション・プログラムを作成できます。

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・プログラムを命名する場合、 EBCDIC ブランク (X'40') より大きい EBCDIC コードを先頭文字にしてください。 X'40' より小さい EBCDIC コードが先頭文字であ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・プログラム用に予約済みです。トランザクション・プログラム名は、最大 64 文字までです。

セキュリティー機能の使用

LU 6.2 は、パートナー LU 検査またはエンド・ユーザー検査のいずれかのタイプのセキュリティー ! 能を提供します。 パートナー LU 検査とは、セッション・レベル・セキュリティー・プロトコルで、セッションがアクティブにされた時に行われます。エンド・ユーザー検査とは、会話レベル・セキュリティーで、会話が開始された時に行われます。

パートナー LU 検査（セッション・レベル・セキュリティー）

パートナー LU 検査は、2 つのLU 間でセキュリティー情報を交換することにより行われます。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。このレベルのセキュリティーは、通常通信ネットワークが物理的に安全でない場合に必要となります。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、 LU 6.2 はパスワード検査のために暗号化を行います。必須ではありませんが、各 LU が固有のパスワードを持つことをお勧めします。

エンド・ユーザー検査（会話レベル・セキュリティー）

エンド・ユーザー検査は、要 a されたアプリケーション・サブシステムが、要 a されたトランザクション・プログラムおよびリソースへのアクセスを提供する前に、リクエスターの一致を確認できるようにします。交換されるセキュリティー情報には、ユーザー ID とパスワードが含まれています。会話レベル・セキュリティーが提供するユーザー ID は、会W・監査の目的でも使用されます。

会話レベル・セキュリティーでは、トランザクション・プログラムを要 a すると、ALLOCATE verb のセキュリティー情報が提供され、リモート・アプリケーション・サブシステムが検査を行います。要 a しているトランザクション・プログラムに対して正しいユーザー ID とパスワードが提供されない場合、リモート・アプリケーション・サブシステムは要 a を q] します。

会話レベル・セキュリティーを必要と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 / 動されるトランザクション・プログラム）は、会話レベル・セキュリティーを必要とする追加トランザクション・プログラムにアクセスする時に使用します。その場合、追加トランザクション・プログラムの割り振り要 a に検査済み標 1 がセットされます。中間トランザクション・プログラムを / 動する時に使用した、最初の要 a から保管されたユーザー ID は、 + 動的に 2 V目の要 a に提供されます。

EBCDIC と ASCII の間の変換

LU 6.2 は、トランザクション・プログラム (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が、 verb で指定された EBCDIC 文字を使用するとみなします。これらの値には、トランザクション・プログラム名、 ALLOCATE で指定されたパートナー 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。プログラムは、着信名を ASCII で格納している場合には、 ASCII と EBCDIC との間で会話を行えるように準wしなければなりません。

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・プログラム間の個々の合意によってhまります。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、データを適切なところで EBCDIC に変換する必要があります。

便9上、LU 6.2 は CONVERT verb を提供しています。この verb は、ASCII コードをEBCDIC に、または EBCDIC コードを ASCII に変換します。詳細については、316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B装

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B装する方法について説明します。

APPC のB装は、 Windows マシン上の Microsoft** NT SNA サーバーとのバイナリー互換性をwえたものとして設W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 インターフェースのB装に似ています。

トランザクション・プログラムの作成

APPC verb を処理するダイナミック・リンク・ライブラリー (DLL) ファイルが提供されています。

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が同時に 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w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。！にユーザー・テ *S551 9.978 295.004 418. -0.015*

サポートされるオプション・セット

パーソナル・コミュニケーションズおよび Communications Server は、以下の APPC オプション・セットをサポートします。各オプション・セットの詳細については、*SNA Transaction Programmer's Reference* の LU タイプ 6.2 を参照してください。

101 LU の送信バッファーのフラッシュ。

102 属性の h 得。

103 通知テストを < う u 信時の通知 (**RECEIVE_AND_POST** verb によって使用できる！能)。

104 待！を < う u 信時の通知 (**RECEIVE_AND_POST** verb によって使用できる！能)。

105 u 信準w。

106 即時 u 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 1 別子の h 得。

110 会話タイプの h 得。

112 全二重会話および優先データ。

オプション 112 は、OS/2 対~ および Windows 3.1 対~ の Communications Server SNA API クライアントではサポートされません。

113 s ブロッキング・サポート。

201 コンテンション勝 T セッションの待ち行列割り振り。

203 セッションの即時割り振り。

204 同じ LU にあるプログラム間の会話。

205 待ち行列化割り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 v 可。

241 PIP データの送信。

242 PIP データの u 信。

243 アカウンティング。

244 長期ロック。

245 送信要 a u 信テスト。

247 ユーザー制御データ。

251 変換および会話相関 8 数の抽出。

290 システム・ログへのデータのロギング。

291 マップ 0 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** *定A* 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 1 の LU 名。

607 s 解 a LU 名。

610 最大 RU サイズ限度。

612 コンテンション勝 T の + 動アクティブ化限度。

613 ローカル最大 (LU、モード) セッション限度。

616 CPSVCMG モード名のサポート。

全 s 重 VCB

トランザクション・プログラムは、全二重会話に必要なフォーマット 1 VCB の定 A を 1 別し優先データを送 u 信するために、 WINAPPC.H ヘッダー・ファイルをインクルードする前にコンパイラー定数 WINAPPC_FORMAT_1 を定 A する必要があります。これは ! 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 A 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

パーソナル・コミュニケーションズおよび Communications Server APPC API は、待ち行列レベルの s ブロッキングをサポートします。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。

s 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、処理中の verb が完了したことを (す通知を u け h 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 s 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 s ブロッキング verb を / 行でき、それらの verb をパーソナル・コミュニケーションズおよび 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 s ブロッキング verb を / 行することができます。

パーソナル・コミュニケーションズおよび Communications Server は、 s ブロッキング verb 用として 6 つの待ち行列を維持します。

- 割り振り待ち行列 (1 つのアクティブ・トランザクション・プログラムにつき 1 つ)
- 送信/ u 信待ち行列 (1 つの会話につき 1 つ、 > 二重のみ)
- 送信待ち行列 (1 つの全二重会話につき 1 つ)
- u 信待ち行列 (1 つの全二重会話につき 1 つ)
- 送信優先待ち行列 (1 つの会話につき 1 つ)
- u 信優先待ち行列 (1 つの会話につき 1 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。パーソナル・コミュニケーションズまたは Communications Server プログラムが他の (ブロッキングまたは s ブロッキング) verb を処理中の場合は、 s ブロッキング verb は待ち行列に入れられます。割り振り待ち行列内の verb は同時に処理されますが、その他の待ち行列内の verb は、いずれかのプログラムが u 信した順序で、一度に 1 つずつ処理されます。

s 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、それをパーソナル・コミュニケーションズまたは Communications Server に通知します。アプリケーションは、 s 同期の verb の完了をアプリケーションに通知するためのイベント・ハンドルを、任意の s ブロッキング verb とともに提供することができます。このハンドルは、 **SECONDARY_RC** フィールドにセットしてパーソナル・コミュニケーションズおよび 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！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。

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

割り振り待ち行列上で、 s ブロッキング・モードで / 行できる APPC verb には、！のものがあります。

(MC_)ALLOCATE

(MC_)SEND_CONVERSATION

送 u 信待ち行列上で、 s ブロッキング・モードで / 行できる APPC verb には、！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

送信待ち行列上で、 s ブロッキング・モードで / 行できる (全二重会話の場合) APPC verb には、！のものがあります。

(MC_)DEALLOCATE

(MC_)FLUSH

(MC_)SEND_DATA

(MC_)SEND_ERROR

u 信待ち行列上で、 s ブロッキング・モードで / 行できる (全二重会話の場合) APPC verb には、！のものがあります。

(MC_)RECEIVE_AND_WAIT

(MC_)RECEIVE_IMMEDIATE

u 信優先待ち行列上で、 s ブロッキング・モードで / 行できる (全二重会話の場合) APPC verb には、！のものがあります。

(MC_)RECEIVE_EXPEDITED_DATA

送信優先待ち行列上で、 s ブロッキング・モードで / 行できる APPC verb には、！のものがあります。

(MC_)REQUEST_TO_SEND

(MC_)SEND_EXPEDITED_DATA

！の APPC verb は常に s 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

s ブロッキング・モードでは / 行できない (そして、アプリケーションが s 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) パーソナル・コミュニケーションズおよび Communications Server APPC verb には、！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 (つまりパーソナル・コミュニケーションズおよび 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 u 信待ち行列または送信優先待ち行列に対して verb を s ブロッキング・モードで / 行することはできません。

送 u 信待ち行列または送信優先待ち行列に対して s ブロッキング verb が / 行されたときに、同じ待ち行列上に現在処理中の他の (ブロッキングまたは s ブロッキングの) verb がある場合は、新 , / 行の verb はその待ち行列に追加され、処理中の verb が完了した時点で処理されます。

他の verb (同じ会話の) が処理中であるときに、ブロッキング verb を / 行すると、パーソナル・コミュニケーションズおよび Communications Server は (**primary_rc** として AP_TP_BUSY を戻して) その verbを q] 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 が処理中でも / 行できる (そしてどの s ブロッキング verb 待ち行列にも入らない) 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 / 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B行される可能性があり、 AP_TP_BUSY を戻さないことに注意してください。

デフォルトのローカル LU

パーソナル・コミュニケーションズおよび Communications Server は、従属？および独立？ LU のデフォルトのローカル LU をサポートしています。デフォルトの LU は、 **lu_alias** フィールドをブランクのままにして **TP_STARTED** verb を / 行したときに使用されます (92ページの『TP_STARTED』を参照)。独立？ LU 6.2 では、デフォルト LU は制御点 LU です。従属？ LU 6.2 では、ローカル LU プールが使用されます。**DEFINE_LOCAL_LU** verb の詳細については、*システム管理プログラミング* を参照してください。パーソナル・コミュニケーションズおよび Communications Server は、デフォルト・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。

- デフォルトのローカル LU プールのメンバーとして LU が構成されている場合には、パーソナル・コミュニケーションズおよび 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 : 敗します。
- デフォルトのローカル 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、パーソナル・コミュニケーションズおよび Communications Server は制御点 LU を使用します。

以下の情報は、Communications Server Windows 95 および Windows NT SNA API クライアントにのみ適用されます。

各ユーザーごとのデフォルトのローカル LU の別名は、適切な構成ユーティリティー (INI 構成または LDAP) を使用して割り当てることができます。

APPC プログラムでは、ローカル LU の別名を直接指定せずに、デフォルトを使用することができます。APPC プログラムが **ローカル LU の別名**フィールドを 2 進ゼロにセットして **TP_START** verb を / 行した場合、 APPC API は構成済みのデフォルトのローカル LU の別名を使用します。

QEL/MU サポート

これは、Communications Server SNA API クライアントにのみ適用されます。

Communications Server は、3270 エミュレーション用の Novell の Queue Element/Message Unit (QEL/MU) アーキテクチャーをB装するエミュレーター・ソフトウェア・パッケージをB行している IPX または TCP/IP 接続クライアントをサポートするようになりました。これには、専用、プール、共用 LU カテゴリー (リソース・タイプとも言う) などの一L的なクライアント！能と、ロード・バランシングが含まれます。

QEL/MU エミュレーター・ソフトウェア・パッケージのB装の詳細については、*Novell Netware for SAA 3270 Client Interface Guide and Reference P/N 100-00218-001* を参照してください。

第6章 CPI-C プログラムのB装

この章では、CPI-C インターフェースに対するパーソナル・コミュニケーションズおよび Communications Serverのサポートの詳細を説明します。！の事柄を扱っています。

- CPI-C プログラムのコンパイルおよびリンクの；法
- CPI-C プログラムの準wおよびB行方法
- パーソナル・コミュニケーションズおよび Communications Serverがサポートする CPI-C バージョンの！能

パーソナル・コミュニケーションズおよび Communications Server が提供する CPIC は、 Windows マシンの Microsoft** NT SNA サーバーとのバイナリー互換性をwえたものとして設W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。

注: この章で説明する内容は、以下のシステムが提供する CPIC API に関するものです。

- Windows NT 上でB行されている Communications Server
- Communications Server/NT に付属の、OS/2、Windows NT、Windows 95、および Windows 3.1 対~の SNA API クライアント
- パーソナル・コミュニケーションズ Windows 95、Windows NT

これらのシステムが提供するサポートの間に違いがある場合は、その都度、明 - します。

CPIC プログラムの作成

パーソナル・コミュニケーションズおよび Communications Server には、CPIC 呼び出しを処理するダイナミック・リンク・ライブラリー (DLL) ファイルが用意されています。

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が同時に DLL を呼び出すことができます。

49ページの表9 に、CPIC プログラムをコンパイルおよびリンクするために必要なヘッダー・ファイルおよびライブラリーのソース・モジュール使用法を (します。 ヘッダー・ファイルの中には、他のヘッダー・ファイルを組み込んでいるものもあります。

表9. CPIC 用ヘッダー・ファイルおよびライブラリー

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

*WINNT = Windows NT
*WIN95 = Windows 95
*WIN3.1 = Windows 3.1

CPI-C のバージョン

CPI-C インターフェースはこれまでバージョンの変更や拡張が行われてきました。！の 2 つの理由で、これらのバージョンに注意してください。

- 既存のプログラムを保 i または移植する場合、どのファンクション・コールに可 B 性があるか、またバージョンの変更に < って変更が必要となるのはどれかを知っておかなければなりません。
- プログラムを新 , 作成する場合、特定のバージョンに従属するコードを作成するときは注意しなければなりません。

CPI-C 適合性クラスのサポート

！に (す CPI-C 2.1 適合性クラスは、 IBM マニュアルの *Common Programming Interface Communications CPI-C Reference Version 2.1* (SC26-4399-08) に定 A されているとおりにサポートされます。

Communications Server クライアントでサポートされていないクラスについては、この章のノートパッド・アイコン が付いている箇所を参照してください。

このアイコンは、重要な情報を (します。

会話適合性クラスは、プログラムが > 二重会話を開始および終了できます。

スターター・セット呼び出し:

CMACCP
Accept_Conversation

CMALLC
Allocate

CMDEAL
Deallocate

CMINIT
Initialize_Conversation

CMRCV
Receive

CMSEND
Send_Data

拡張！能呼び出し:

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

必要な sync_level 値は！のとおりです。

CM_NONE または CM_CONFIRM

CMSTPN
Set_TP_Name

CMTRTS
Test_Request_To_Send_Received

LU 6.2 適合性クラスは、プログラムが！のような LU 6.2 の特定のサービスを使用できます。

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性クラスは、呼び出しが即時に完了できない場合に、プログラムが制御を h り戻せます。

CMCANC
Cancel_Conversation

CMSPM
Set_Processing_Mode

CMWAIT
Wait_For_Conversation

サーバー適合性クラスは、プログラムが、複数のトランザクション・プログラム名を CPI-C で登録すること、複数の着信会話を u 諾すること、および異なるクライアントに対する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

CMECTX Extract_Conversation_Context は、Communications Server Windows 3.1 クライアントではサポートされていません。

CMETPN
Extract_TP_Name

CMRLTP
Release_Local_TP_Name

CMINIC
Initialize_For_Incoming

CMSLTP
Specify_Local_TP_Name

データ変換適合性クラス・ルーチンは、プログラムが、ローカル・ルーチンを呼び出して文字ストリングの符号化をローカル符号化から EBCDIC へ、またはそのUへ変更できます。

CMCNVI

Convert_Incoming

CMCNVO

Convert_Outgoing

セキュリティー適合性クラスは、プログラムが、サイド情報のアクセス・セキュリティー情報を使用する会話またはプログラムによって直接設定された会話を確立できます。

CMESUI

Extract_Security_User_ID

CMSCSP

Set_Conversation_Security_Password

CMSCST

Set_Conversation_Security_Type

Required conversation_security_type values:

CM_SECURITY_NONE

CM_SECURITY_PROGRAM

CM_SECURITY_PROGRAM_STRONG

CM_SECURITY_SAME

CMSCSU

Set_Conversation_Security_User_ID

待ち行列レベルの非ブロッキングは、呼び出しが完了できない場合に制御を h り戻すためのものです。

CMCANC

Cancel_Conversation

CMSQPM

Set_Queue_Processing_Mode

CMWCMP

Wait_For_Completion

待ち行列レベルの s ブロッキングは、Communications Server Windows 3.1 クライアントではサポートされていません。

コールバック機能は、呼び出しが完了できない場合に制御を h り戻すためのものです。

CMCANC

Cancel_Conversation

CMSQCF

Set_Queue_Callback_Function

コールバック・ファンクションは、Communications Server Windows 3.1 または OS/2 クライアントではサポートされていません。

2 次情報は、2 ！エラー戻り情報を h り出せるようにします。

CMESI

Extract_Secondary_Information

2 ！情報は、Communications Server Windows 3.1 クライアントではサポートされていません。

以下のクラスは、Communications Server SNA API クライアント OS/2 G および Windows 3.1 G ではサポートされていません。

全二重は、全二重会話へのユーザー・アクセスを可能にします。

CMESRM

Extract_Send_Receive_Mode

CMSSRM

Set_Send_Receive_Mode

優先データは、優先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

！の適合性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）

s チェーン・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保管されているユーザー指定情報）

CPI-C 機能

パーソナル・コミュニケーションズおよび Communications Serverがサポートする CPI-C ！能すべてを55ページの表 10 にリストします。古いプログラムを保 i している場合、または既存のシステムとの互換性が必要な新しいプログラムを作成している場合、この参照用テーブルを使用してください。

注: MS Windows SNA API クライアント用の CPI-C アプリケーションを作成する場合には、 Accept_Conversation (cmaccp) 呼び出しを介して着信会話を u 信する前に、 Specify_Local_TP-Name (cmsltp) 呼び出しを介してローカル・トランザクション・プログラムを指定します。

表 10. CPI-C ！能のパーソナル・コミュニケーションズおよび Communications Server クライアント・サポート

機能	長い名前	Windows NT Server およびパーソナル・コミュニケーションズ	Windows 95 および Windows NT クライアント	OS/2 クライアント	Windows 3.1 クライアント
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	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	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	-
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	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	x	-	-
cmserr	Send_Error	x	x	x	x
cmscsp	Set_Conversation_Security_Password	x	x	x	x

表 10. CPI-C ！能のパーソナル・コミュニケーションズおよび Communications Server クライアント・サポート (続き)

機能	長い名前	Windows NT Server およびパーソナル・コミュニケーションズ	Windows 95 および Windows NT クライアント	OS/2 クライアント	Windows 3.1 クライアント
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	x	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	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 用 WOSA ！能
x: サポートされる！能
-: サポートされない！能

サービス TP 名の指定

この！能は、Communications Server SNA API クライアントの場合のみサポートされます。

CMSTPN および CMSLTP ！能のあるサービス・トランザクション・プログラム名を指定する場合、特 l ，則を使用しなければなりません。通常、CPI-C ！能とともに標準 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、共通ネットワークおよびシステム・サービスを、他のプログラムまたはユーザーに提供する特 l 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、およびスプーラーが s げられます。

CMSTPN および CMSL トランザクション・プログラム！能のあるサービス・トランザクション・プログラム名を指定する際の，則は、！のとおりです。

- 名前は、2〜5 バイトの ASCII 文字で指定します。
- 名前の最初のバイトは、2 バイトの ASCII 文字です（たとえば、0x23）。
  - 名前の最初のバイトを 2 つのニブルに分割し（たとえば、2 と 3）、各 ASCII バイトの下位ニブルでそれらを指定します。
  - 各 ASCII バイトの上位ニブルを 1 にセットします。これは、サービス TP 名を指定していることを (しています。上 - の例では、指定された最初の 2 バイトは、0x12 および 0x13 です。
- 残りのゼロを、3 バイトの ASCII 文字の名前として指定します。たとえば、007 です。

したがって、0x23 007 というサービス・トランザクション・プログラム名は、 0x12 0x13 007 となります。

第7章 APPC 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。

注: 本書の第 1 部の各章の内容は、以下のシステムが提供する APPC API に関するものです。

- Windows NT 上でB行されている Communications Server
- Communications Server/NT に付属の、OS/2、Windows NT、Windows 95、および Windows 3.1 対~の SNA API クライアント
- パーソナル・コミュニケーションズ Windows 95、Windows NT

これらのシステムが提供するサポートの間に違いがある場合は、その都度、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 s ブロッキング verb を / 行することもできます。そのためには、2 ！戻りコード・フィールドにイベント・ハンドルを入れ、**opext** フィールドで待ち行列レベルの s 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 ！戻りコードおよび 2 ！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: 64ページの『WinAsyncAPPCEx()』

APPC は、SNA API OS/2 クライアントでサポートされてい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 s 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。パーソナル・コミュニケーションズおよび 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 u け h るウィンドウ・ハンドル。

vcb verb 制御ブロックのポインター。

戻り値

戻り値は、 s 同期要 a が正常に完了したかどうかを (します。要 a が成功した場合は、B 際の戻り値はハンドルです。 ！能が正常終了しなかった場合、パーソナル・コミュニケーションズおよび 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！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_]RECEIVE_EXPEDITED_DATA**
- **[MC_]REQUEST_TO_SEND**
- **[MC_]SEND_CONVERSATION**
- **[MC_]SEND_DATA**
- **[MC_]SEND_ERROR**
- **[MC_]SEND_EXPEDITED_DATA**
- **TP_ENDED**
- **TP_STARTED**

WinAsyncAPPC エントリー・ポイントでは verb を h り消すことはできますが、待ち行列レベルの s ブロッキングはサポートしていません。 APPC エントリー・ポイントでは、待ち行列レベルの s ブロッキングはサポートしていますが、 verb を h 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 s ブロッキングをサポートしていません。 s 同期インターフェースで待ち行列レベルの s ブロッキング・フラグ AP_NON_BLOCKING を指定してあっても、パーソナル・コミュニケーションズおよび Communications Server はそれを無視します。 s 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処理中のままにしておける！能は、 1 つの会話につき一度に 1 つだけです。 このときに 2 V目の！能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 s 同期完了の通知を u け h 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。 s 同期サポートを十分に活用できるようにするために、パーソナル・コミュニケーションズおよび Communications Server は、 s 同期に / 行された **RECEIVE_AND_WAIT** verb を、 **RECEIVE_AND_POST** verb と同じ働きになるように変更します。 つまり、 s 同期の **RECEIVE_AND_POST** または **RECEIVE_AND_WAIT** が処理中であるときに、アプリケーションは同じ会話で！のような verb を / 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**

k 果として、サーバーなどのアプリケーションが、 s 同期の **RECEIVE_AND_WAIT** を使用してデータを u 信できるようになります。 **RECEIVE_AND_POST** または **RECEIVE_AND_WAIT** が処理中であっても、アプリケーションは **SEND_ERROR** および **REQUEST_TO_SEND** を使用できます。

s 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 u け h ります。 *wParam* 引き数には、元のファンクション・コールから戻された s 同期のタスク・ハンドルが入っています。 *lParam* 引き数には元の VCB ポインターが入っており、これは最終の戻りコードを = 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どの s 同期 APPC アクションも h り消すことができますが、それに < 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。 処理中の操作がある場合は、戻りコード AP_CANCELED を返します。

この！能が正常終了すると、パーソナル・コミュニケーションズおよび Communications Server は、操作が完了したときまたは会話が h 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:

64ページの『WinAsyncAPPCEx()』

66ページの『WinAPPCCancelAsyncRequest()』

WinAsyncAPPCEx()

これは、すべての APPC verb 用の s 同期エントリー・ポイントです。 この呼び出しは、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場合に使用します。

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処理中の verb をh り消す！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 s 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！するイベントのハンドル。

vcb verb 制御ブロックのポインター。

戻り値

戻り値は、 s 同期完了要 a が成功したかどうかを (します。成功した場合は、B 際の戻り値はハンドルです。 ！能が正常終了しなかった場合は、パーソナル・コミュニケーションズおよび 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！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 s ブロッキングをサポートしていません。 s 同期インターフェースで待ち行列レベルの s ブロッキング・フラグ AP_NON_BLOCKING を指定してあっても、パーソナル・コミュニケーションズおよび Communications Server はそれを無視します。 s 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処理中のままにしておける！能は、1 つの会話につき一度に 1 つだけです。 このときに 2 V目の！能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 を h り消すことができますが、待ち行列レベルの s ブロッキングはサポートしていません。 **APPC** エントリー・ポイントでは、待ち行列レベルの s ブロッキングをサポートしていますが、 verb を h 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。 s 同期サポートを十分に活用できるようにするために、パーソナル・コミュニケーションズおよび Communications Server は、 s 同期に / 行された **RECEIVE_AND_WAIT** verb を、 **RECEIVE_AND_POST** verb と同じ働きになるように変更します。 つまり、 s 同期の **RECEIVE_AND_POST** または **RECEIVE_AND_WAIT** の処理中に、アプリケーションは同じ会話で！のような verb を / 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**

k 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、 s 同期の **RECEIVE_AND_WAIT** を使用してデータを u 信できるようになります。 **RECEIVE_AND_POST** または **RECEIVE_AND_WAIT** が処理中であっても、アプリケーションは **SEND_ERROR** および **REQUEST_TO_SEND** を使用できます。

s 同期操作が完了すると、パーソナル・コミュニケーションズおよび 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 u け h ると、1 ！戻りコードおよび 2 ！戻りコードを調べて、エラー条 o の有無を確認します。

関連情報:

61ページの『WinAsyncAPPC()』

66ページの『WinAPPCCancelAsyncRequest()』

60ページの『APPC』

WinAPPCCancelAsyncRequest()

この！能は、処理中の **WinAsyncAPPC** に基づく要 a を h 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。 h り消したい要 a のハンドルを指定します。

戻り値

戻り値は、 s 同期要 a が h り消されたかどうかを (します。値が 0 の場合は、パーソナル・コミュニケーションズおよび Communications Server は要 a を h り消しています。 h り消していない場合は、値は！に (すエラー・コードのいずれかになります。

WAPPCINVALID

指定した s 同期タスク ID が無効でした。

WAPPCALREADY

h り消そうとしている s 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()** 呼び出しを / 行し、ハンドル内の初期！能から戻される s 同期イベントを指定することにより、**WinAsyncAPPC** ！能の 1 つを使用して / 行した s 同期タスクを、それが完了する前に h り消すことができます。

処理中の 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、パーソナル・コミュニケーションズおよび Communications Server はその verb を除 n し、セッションを s 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、パーソナル・コミュニケーションズおよび 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、パーソナル・コミュニケーションズおよび Communications Server はできるだけ完全に会話とセッションを s 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。 この呼び出しは同期呼び出しです。上 - で述べた処理が完了すると、パーソナル・コミュニケーションズおよび Communications Serverは、 h 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 s 同期 **WinAsyncAPPC** ルーチンを h り消すのに : 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、 k 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 s ブロッキング・エントリー・ポイントを介して / 行された s 同期 verb を h り消すことはできません。

関連情報: 61ページの『WinAsyncAPPC()』

WinAPPCCancelBlockingCall()

この！能は、スレッドに関する処理中のブロッキング操作を h り消します。パーソナル・コミュニケーションズおよび Communications Server は、処理中のブロックされた呼び出しを h り消した場合、エラー・コード AP_CANCELLED を生成します。この呼び出しは、ブロッキング・フック！能内からのみ使用するものです。パーソナル・コミュニケーションズおよび Communications Server は、既存のアプリケーションとの後方互換性を確保するために、この！能を提供しています。

構文

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

戻り値

戻り値は、 h り消し要 a が成功したかどうかを (します。値が 0 の場合は、パーソナル・コミュニケーションズおよび Communications Server は要 a を h り消しています。 h り消していない場合は、値は！に (すエラー・コードになります。

WAPPCINVALID
処理中のブロッキング呼び出しはありません。

使用上の注意

処理中の 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、パーソナル・コミュニケーションズおよび Communications Server はその verb を除 n し、セッションを s 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、パーソナル・コミュニケーションズおよび 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、パーソナル・コミュニケーションズおよび Communications Server はできるだけ完全に会話とセッションを s 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。 この呼び出しは同期呼び出しです。上 - で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処理中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。複数の処理中の呼び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・スレッド上に処理中の操作があれば、それを h り消します。それがない場合は、この！能は : 敗します。 APPC は、処理中の操作がある間は、呼び出し元のアプリケーション・スレッドを中断します。その k 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()** の呼び出しを / 行することはできません)。

この！能は、Windows、Windows 95、および Windows NT SNA API クライアントではサポートされていません。

WinAPPCCleanup()

この！能は、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 h り消します。

構文

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

戻り値

戻り値は、登録の h り消しが成功したかどうかを (します。 値が 0 以外である場合は、パーソナル・コミュニケーションズおよび Communications Server はアプリケーションの登録を正常に h り消しました。アプリケーションの登録を h り消していない場合は、パーソナル・コミュニケーションズおよび Communications Server は値 0 を戻します。

使用上の注意

APPC API からパーソナル・コミュニケーションズおよび Communications Server アプリケーションの登録を h り消す場合には、**WinAPPCCleanup()** を使用します。

パーソナル・コミュニケーションズおよび Communications Server は、まだアクティブな会話を終了し、トランザクション・プログラムを終了します。この！能は、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 / 行するのと同じです。

関連情報: 72ページの『WinAPPCStartup()』

WinAPPCIsBlocking()

この！能は、前のブロッキング要 a の終了を待ちながらスレッドが B 行しているかどうかを = 別します。パーソナル・コミュニケーションズおよび Communications Server は、既存のアプリケーションとの後方互換性を確保するために、この！能を提供しています。

構文

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

戻り値はこの！能の k 果を (します。値が 0 以外である場合は、処理中のブロッキング呼び出しが完了を待！しています。値 0 は、処理中のブロッキング呼び出しがないことを意味します。

使用上の注意

パーソナル・コミュニケーションズおよび Communications Server DLL は、1 つのスレッドあたりの複数のブロッキング呼び出しを禁止します。この状況が生じると、AP_THREAD_BLOCKING を戻します。 ブロッキング呼び出しを B 行しているスレッドは、ブロッキング・フック！能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！能内からのみ TRUE を戻します。

関連情報:

68ページの『WinAPPCCancelBlockingCall()』

73ページの『WinAPPCSetBlockingHook()』

75ページの『WinAPPCUnhookBlockingHook()』

この！能は、Windows 95 および Windows NT の SNA API クライアントではサポートされていません。

WinAPPCStartup()

この！能を使用すると、アプリケーションで、必要なパーソナル・コミュニケーションズおよび Communications Server のバージョンを指定し、パーソナル・コミュニケーションズおよび Communications Server からバージョン情報を h 得することができます。この呼び出しは必須ではありません。

構文

得ぎごすご ちばどば

呼　　個　権　　呼

ピツ呼ヲォピリクテゲ原後ケ　グゴザェニセユサルケケ　ケ　ピユケスミュニケジョン

WinAPPCSetBlockingHook()

この！能を使用すると、APPC API の APPC B装によって APPC ファンクション・コールをブロックすることができます。

パーソナル・コミュニケーションズおよび 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！能を提供しています。

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！能のプロシージャー・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！能のプロシージャー・インスタンスを (します。**SetBlockingHook** ！能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** を使用してデフォルト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！能は、WM_QUIT メッセージを u け h ると FALSE を戻します。したがって、パーソナル・コミュニケーションズおよび Communications Server は、制御をアプリケーションに戻し、メッセージを処理して正常に終了することができます。 その他の場合は、この！能は TRUE を戻します。

デフォルトではブロッキング・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、ブロッキング呼び出しの戻りを無限に待ちます。

ブロッキング・フック！能が TRUE を戻さなかった場合は、ブロッキング verb を、1 ！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！能はスレッド単位でB行されます。 特定のスレッドでブロッキング・メカニズムを置き換えても、他のスレッドが F 響を u けることはありません。

関連情報:

68ページの『WinAPPCCancelBlockingCall()』

71ページの『WinAPPCIsBlocking()』

75ページの『WinAPPCUnhookBlockingHook()』

この！能は、Windows 95 および Windows NT の SNA API クライアントではサポートされていません。

WinAPPCUnhookBlockingHook()

この！能は、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。

パーソナル・コミュニケーションズおよび Communications Server は、既存のアプリケーションとの後方互換性を確保するために、この！能を提供しています。

構文

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

戻り値

戻り値はこの！能の k 果を (します。パーソナル・コミュニケーションズおよび Communications Server がデフォルト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。パーソナル・コミュニケーションズおよび Communications Server がデフォルトのメカニズムを再導入しなかった場合は、戻り値は 0 です。

使用上の注意

この！能が呼び出されると、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び出しが完了するまで、無限に待！します。

関連情報: 73ページの『WinAPPCSetBlockingHook()』

この！能は、Windows 95 および Windows NT の SNA API クライアントではサポートされていません。

GetAppcConfig()

この！能はB装されていません。しかし、後方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・セットを指定すると、パーソナル・コミュニケーションズおよび Communications Server は APPC_CFG_SUCESS_NO_DEFAULT_REMOTE を戻し、 NULL ターミネーターを RemLu バッファーの先頭のバイトに入れます。

パーソナル・コミュニケーションズおよび Communications Server は APPN ノードになることができるので、この呼び出しが必要になることはほとんどありません。パートナー LU の名前は ALLOCATE 上で指定でき、LU の検索が開始されます。ただし、アプリケーションはノード・オペレーター！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。 NOF インターフェースの詳細については、*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！能は、VCB 内の 1 ！戻りコードおよび 2 ！戻りコードを印刷可能ストリングに変換します。 この！能は、APPC アプリケーションが使用するエラー・ストリングの標準セットを提供しています。

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A 0化された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。 指定バッファー内のストリングの長さ。

戻り値

0x20000001
パラメーターが無効です。この！能は、指定した verb 制御ブロックからの読みh り、または指定したバッファーへの書き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・ストリングが、改行文字 (**¥n**) で終わっていません。

第8章 APPC verb

この章では、APPC API を介して u け渡される各 verb の構文を (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 全二重会話および > 二重会話に提供される APPC 基本会話およびマップ 0 会話についての参照情報も説明します。この章を読み進んでいくと、基本会話 verb とマップ 0 会話 verb がよく似ていることに $ 付かれるでしょう。ですから、同じ章で扱っているわけです。しかし、異なる点もあります。相違点は！のようなアイコンで表 (します。

この - 号は、内容が基本 verb にのみ適用されることを (します。

この - 号は、内容がマップ 0 verb にのみ適用されることを (します。

情報に関するもう 1 つの区別！ 会話 verb が基本またはマップ 0 のどちらでもよい場合は、！のように (します。

[MC_]VERBNAME

注: 本書の第 1 部の各章の内容は、以下のシステムが提供する APPC API に関するものです。

- Windows NT 上で B 行されている Communications Server
- Communications Server に付属の、OS/2、Windows NT、Windows 95、および Windows 3.1 の SNA API クライアント
- パーソナル・コミュニケーションズ Windows 95、Windows NT

これらのシステムが提供するサポートの間に違いがある場合は、その都度、明 - します。

verb 制御ブロック

このセクションでは、各 verb のフィールドおよび操作に関する一 L 事項について説明します。

共通フィールド

各 VCB には、！に (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 1 別フィールド。

format
VCB のフォーマットを 1 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に設定する値については、それぞれの verb の項で個別に説明します。

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

secondary_rc
2 ！戻りコード。 これは、1 ！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 (します。 VCB によっては、上 - に加えて！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 s 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、個別には説明しません。

TP 識別子
個々のアクティブなトランザクション・プログラムには、それぞれ 8 バイトのトランザクション・プログラム 1 別子が割り当てられます。この 1 別子は、パーソナル・コミュニケーションズおよび 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 1 別子は、**TP_ENDED** verb の P 路指定のため、および会話 verb での相関 8 数として使用されます。

以下、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

パーソナル・コミュニケーションズおよび Communications Server は、APPC API で以下の verb をサポートしています。

タイプに依存しない 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

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 構造体

```
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、！のパラメーターをパーソナル・コミュニケーションズおよび 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 1 別子。このパラメーターの値は、ローカル・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、パートナー・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 1 別します。上 - に (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 B 行された場合は、パーソナル・コミュニケーションズおよび Communications Server は！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 / 行するトランザクション・プログラム) の名前。パーソナル・コミュニケーションズおよび 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 JISCII 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 1 別子です。 **luw_id_overlay** には！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&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 k された 2 つのタイプ A の 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 構造体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
論理作業単位インスタンスV号。これは、長さが 6 バイトのバイナリー・ストリングです。

luw_id_overlay.sequence
論理作業単位シーケンスV号。これは、長さが 2 バイトのバイナリー・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&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&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 k された 2 つのタイプ A の 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 の EBCDIC 文字ストリングで、 10 バイトに満たない場合は&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 B 行されなかった場合は、パーソナル・コミュニケーションズおよび Communications Server は ! のパラメーターを戻します。

primary_rc

AP_PARAMETER_CHECK

secondary_rc

AP_BAD_TP_ID

! に (す 1 ! 戻りコード（**primary_rc**）が生成される条 o 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_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 0) を戻します。

VCB 構造体

```
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、！のパラメーターをパーソナル・コミュニケーションズおよび 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 1 別します。上 - に (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 1 別子。このパラメーターの値は、ローカル・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、パートナー・トランザクション・プログラムでは **RECEIVE_3ALLOCATE** verb から戻された値です。

conv_id
会話 1 別子。 このパラメーターの値は、ローカル・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、パートナー・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 B 行された場合は、パーソナル・コミュニケーションズおよび Communications Server は！のパラメーターを戻します。

primary_rc
AP_OK

conv_type
conv_id で 1 別される会話の会話タイプ。

AP_BASIC_CONVERSATION
AP_MAPPED_CONVERSATION

conv_style

conv_id で 1 別される会話の会話スタイル。 このフィールドには、VCB のフォーマット 1 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、43ページの『全二重 VCB』 を参照してください。

AP_HALF_DUPLEX
AP_FULL_DUPLEX

パラメーター・エラーが原因で verb が B 行されなかった場合は、パーソナル・コミュニケーションズおよび Communications Server は ! 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

! に (す 1 ! 戻りコード (**primary_rc**) が生成される条 o 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_UNEXPECTED_SYSTEM_ERROR

RECEIVE_ALLOCATE

RECEIVE_ALLOCATE verb は、 **ALLOCATE** verb または **MC_ALLOCATE** verb を / 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の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 a します。

VCB 構造体

```
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 alia s              */
 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、！のパラメーターをパーソナル・コミュニケーションズおよび 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 1 別します。 上 - に (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

トランザクション・プログラムの名前。パーソナル・コミュニケーションズおよび 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 B 行された場合は、パーソナル・コミュニケーションズおよび Communications Server は！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 1 別子。この値は、パーソナル・コミュニケーションズおよび 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 1 別子をパーソナル・コミュニケーションズおよび Communications Server に渡します。

conv_id
会話 1 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 1 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 1 別される会話の会話タイプ。

AP_BASIC_CONVERSATION
AP_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 の EBCDIC 文字ストリングで、 10 バイトに満たない場合は&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 1 させるための別名。 これは、8 バイトの JISCII 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 1 させるための別名。これは、8 バイトの JISCII 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A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Q数字のタイプ A の EBCDIC ストリング（Q字で始まるもの）で、 8 バイトに満たない場合は&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 1 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&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 k された 2 つのタイプ A の 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 a 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 / 行される **[MC_]RECEIVE_*** verb で PIP データが u 信されます。

conversation_style

conv_id で 1 別される会話の会話スタイル。

AP_HALF_DUPLEX
AP_FULL_DUPLEX

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 の EBCDIC 文字ストリングで、 10 バイトに満たない場合は&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。

dload_id

このフィールドは、**format** フィールドに 1 がセットされている場合しかセットできません。 DYNAMIC_LOAD_INDICATION に ~ 答して RECEIVE_ALLOCATE が / 行された場合には、このフィールドを使用して、2 つのシグナルを以下の方法で相互に関連付けることができます。

dload_id に以下のいずれかの値がセットされた場合に限り、RECEIVE_ALLOCATE と DYNAMIC_LOAD_INDICATION は相互に関連付けられます。

- すべてゼロ
- DYNAMIC_LOAD_INDICATION 上の **dload_id** フィールド。

注: このパラメーターは、SNA API クライアントではサポートされません。

パラメーター・エラーが原因で verb が B 行されなかった場合は、パーソナル・コミュニケーションズおよび Communications Server は ! のパラメーターを戻します。

primary_rc

AP_PARAMETER_CHECK

secondary_rc

AP_UNDEFINED_TP_NAME

! に (す 1 ! 戻りコード (**primary_rc**) が生成される条 o については、付録A. APPC 共通戻りコードで説明します。

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、パーソナル・コミュニケーションズおよび Communications Server に通知します。

VCB 構造体

```
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、！のパラメーターをパーソナル・コミュニケーションズおよび 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 1 別します。 上 - に (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 1 別子。このパラメーターの値は、ローカル・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、パートナー・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_HARD
AP_SOFT
AP_ABEND
AP_CANCEL

タイプが AP_ABEND の場合は、パーソナル・コミュニケーションズおよび Communications Server は **TP_ENDED** 信号に ~ 答しません。

戻りパラメーター

verb が正常に B 行された場合は、パーソナル・コミュニケーションズおよび Communications Serverは！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B行されなかった場合は、パーソナル・コミュニケーションズおよび Communications Server は！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

！に (す 1 ！戻りコード (**primary_rc**) が生成される条 o 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_UNEXPECTED_SYSTEM_ERROR

TP_STARTED

TP_STARTED verb は、着信割り振り要 a ではなくローカル・コマンドの k 果として開始されたトランザクション・プログラムに対して、プログラムがリソースを要 a していることをパーソナル・コミュニケーションズおよび Communications Server に通知します。

VCB 構造体

```
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、！のパラメーターをパーソナル・コミュニケーションズおよび 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 1 別します。 上 - に (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 1 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、パーソナル・コミュニケーションズおよび Communications Server は 制御点 LU を使用します。これは、8 バイトの JISCII 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** フィールドも有効です。その場合は制御点 LU が使用されます。

以下の内容は、Communications Server Windows 95 および Windows NT の SNA API クライアントにのみ適用されます。

各ユーザーごとのデフォルトのローカル LU の別名は、適切な構成ユーティリティー (INI 構成または LDAP) を使用して割り当てることができます。

APPC プログラムでは、ローカル LU の別名を直接指定せずに、デフォルトを使用することができます。APPC プログラムが **local_LU_alias**フィールドを 2 進ゼロにセットして **TP_START** verb を / 行した場合、 APPC API は構成済みのデフォルトのローカル LU の別名を使用します。

tp_name
トランザクション・プログラムの名前。パーソナル・コミュニケーションズおよび 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 B 行された場合は、パーソナル・コミュニケーションズおよび Communications Server は ! 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 1 別子。この値は、パーソナル・コミュニケーションズおよび 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 1 別子をパーソナル・コミュニケーションズおよび Communications Server に渡します。

パラメーター・エラーが原因で verb が B 行されなかった場合は、パーソナル・コミュニケーションズおよび Communications Server は ! 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

! に (す 1 ! 戻りコード (**primary_rc**) が生成される条 o については、付録A. APPC 共通戻りコードで説明します。

AP_UNEXPECTED_SYSTEM_ERROR

[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 / 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 ! 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 0 会話のどちらでも確立できます。**ALLOCATE** verb を使用してマップ 0 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 0 会話パートナー・トランザクション・プログラムを通信することができます。

パーソナル・コミュニケーションズおよび Communications Server は、この verb が正常に B 行されると、会話 1 別子 (**conv_id**) を生成します。 この 1 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 構造体

```
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、！のパラメーターをパーソナル・コミュニケーションズおよび 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 1 別します。上 - に (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。 s ブロッキング操作の場合は、このフラグと AP_NON_BLOCKING を OR で k 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 1 別子。

このパラメーターの値は、ローカル・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、パートナー・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 0 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 / 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 + 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 / 行することができ、また、マップ 0 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B装で、マップ 0 会話 verb がサポートされている場合)。詳細については、 *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 a 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 1 別される会話の会話スタイル。

AP_HALF_DUPLEX
AP_FULL_DUPLEX

このパラメーターは、Communications Server の OS/2 および Windows 3.1 SNA API クライアントではサポートされません。

conv_group_id
割り振られるセッションの会話グループ 1 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 1 させるための別名。これは、8 バイトの JISCII 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。このフィールドがすべて 0 にセットされている場合、パーソナル・コミュニケーションズおよび 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。

以下の内容は、Communications Server の Windows 95 および Windows NT SNA API クライアントにのみ適用されます。

各ユーザーごとのデフォルトのパートナー LU の別名は、適切な構成ユーティリティー (INI 構成または LDAP) を使用して割り当てることができます。

APPC プログラムでは、パートナー LU の別名を直接指定せずに、デフォルトを使用することができます。APPC プログラムが **partner_LU_alias** フィールドおよび **fully_qualified_partner_LU** フィールドを 2 進ゼロにセットして **ALLOCATE** verb を / 行した場合、 APPC API は構成済みのデフォルトのパートナー LU の別名を使用します。

mode_name

構成時に定Aされる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Q数字のタイプ A の EBCDIC ストリング (Q字で始まるもの) で、 8 バイトに満たない場合は&側に EBCDIC のスペースが埋め込まれます。

tp_name

パートナー・トランザクション・プログラムの名前。 パーソナル・コミュニケーションズおよび 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 によって指定された **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 によって指定された **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

パートナー・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

パートナー・トランザクション・プログラムは、ユーザー ID とパスワードを必要とする会話セキュリティーを使用します。

AP_SAME

パートナー・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、検査済み標 1 をu け入れるように構成されます。 ユーザー ID は検査済み標 1 と共に送られ、パスワードが必要でないことを、パートナー・トランザクション・プログラムに通知します。

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B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) 上で指定されたユーザー ID と

の EBCDIC 文字ストリングで、 10 バイトに満たない場合は&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 の EBCDIC 文字ストリングで、10 バイトに満たない場合は&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O囲は 0 〜 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&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 k された 2 つのタイプ A の 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B行された場合は、パーソナル・コミュニケーションズおよび Communications Server は！のパラメーターを戻します。

primary_rc
AP_OK

conv_id
会話1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1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1別子。

verb が s ブロッキングで完了していない場合は、パーソナル・コミュニケーションズおよび Communications Server は！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 に設定してあるときに、すぐに使用できるセッションがない場合は、パーソナル・コミュニケーションズおよび Communications Server は！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B行されなかった場合は、パーソナル・コミュニケーションズおよび Communications Serverは！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：敗した理由に関する追加情報を提供します。

！に (す 1 ！戻りコード (**primary_rc**) およびそれに付随する 2 ！戻りコード (**secondary_rc**) が生成される条 o 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 a 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 ~ 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 u け h ったことを確認するために **CONFIRMED** verb を / 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 / 見した場合は、 **SEND_ERROR** verb を / 行するか、または異常処置として会話の割り振りを解除します)。

トランザクション・プログラムが **CONFIRM** verb を / 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 構造体

```
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、！のパラメーターをパーソナル・コミュニケーションズおよび 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。 s ブロッキング操作の場合は、このフラグと AP_NON_BLOCKING を OR で k ぶことができます。

format
VCB のフォーマットを 1 別します。上 - に (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 1 別子。このパラメーターの値は、ローカル・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、パートナー・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 1 別子。 このパラメーターの値は、ローカル・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、パートナー・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 B 行された場合は、パーソナル・コミュニケーションズおよび Communications Server は ! のパラメーターを戻します。

primary_rc
AP_OK

rts_rcvd
送信要 a u 信の標 1 。

AP_YES
AP_NO

expd_data_rcvd
優先データ u 信の標 1 。この標 1 は、RECEIVE_EXPEDITED_DATA が / 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 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、43ページの『全二重 VCB』 を参照してください。

verb が s ブロッキングで完了していない場合は、パーソナル・コミュニケーションズおよび Communications Server は ! のパラメーターを戻します。

primary_rc
AP_OPERATION_INCOMPLETE

opext verb が s ブロッキングで完了していない場合は、パーソナル・コミュニケーションズおよび Communications Server は ! 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B行されなかった場合は、パーソナル・コミュニケーションズおよび Communications Server は！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 / 行したときに会話が不適切な状態にあった場合は、パーソナル・コミュニケーションズおよび Communications Server は！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 

！に (す 1 ！戻りコード (**primary_rc**) およびそれに付随する 2 ！戻りコード (**secondary_rc**) が生成される条 o 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_RETRY
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

注: パフォーマンス上の理由から、SNA API クライアントは、サーバーに転送することなく、 **[MC_]SEND_DATA** verb で成功を (す戻りコードを戻すことができます。その後で **[MC_]CONFIRM** verb が / 行されると、**[MC_]SEND_DATA** はサーバーに転送されて処理されます。**[MC_]SEND_DATA** エラー戻りコードがある場合には、そのコードは **[MC_]CONFIRM** verb で戻されます。エラー戻りコードのリストについては、**[MC_]SEND_DATA** verb を参照してください。

[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 a に ~ 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 u 信したデータで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 a を / 行したトランザクション・プログラムは確認 ~ 答を待つので、**CONFIRMED** verb では 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 構造体

```
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、！のパラメーターをパーソナル・コミュニケーションズおよび 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。 s ブロッキング操作の場合は、このフラグと AP_NON_BLOCKING を OR で k ぶことができます。

format

VCB のフォーマットを 1 別します。上 - に (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 1 別子。このパラメーターの値は、ローカル・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、パートナー・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 1 別子。 このパラメーターの値は、ローカル・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、パートナー・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 B 行された場合は、パーソナル・コミュニケーションズおよび Communications Server は ! のパラメーターを戻します。

primary_rc
AP_OK

verb が s ブロッキングで完了していない場合は、パーソナル・コミュニケーションズおよび Communications Server は ! 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 B 行されなかった場合は、パーソナル・コミュニケーションズおよび Communications Server は ! 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_CONFIRMED_INVALID_FOR_FDX

トランザクション・プロセッサーがこの verb を / 行したときに会話が不適切な状態にあった場合は、パーソナル・コミュニケーションズおよび Communications Server は ! のパラメーターを戻します。

primary_rc
AP_STATE_CHECK

secondary_rc
AP_CONFIRMED_BAD_STATE

! に (す 1 ! 戻りコード（**primary_rc**）が生成される条 o 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_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り振りを解除します。会話の割り振り解除の前に、この verb は！のいずれかの verb と同じ処理を行います。

- **FLUSH** verb。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。ローカル LU の送信バッファー・データおよび確認要 a 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B行されると、指定した会話 ID (conv_id) は無効になります。

＞二重会話の場合：

- 指定された会話をトランザクション・プログラムから割り振り解除します。**FLUSH** または **CONFIRM** verb の！能を含んでいる可能性があります。

全二重会話の場合：

- **TYPE(FLUSH)** が指定されている **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・プログラムは両方とも、 + 分の送信待ち行列を個別に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という通知は、**DEALLOCATE_NORMAL** 戻りコードのA で u 信待ち行列に入れられます。
- **TYPE(ABEND)** が指定されている **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 u 信待ち行列へ戻されます。

VCB 構造体

```
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、！のパラメーターをパーソナル・コミュニケーションズおよび 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。 s ブロッキング操作の場合は、このフラグと AP_NON_BLOCKING を OR で k 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 k ぶ必要があります。 

format

VCB のフォーマットを 1 別します。上 - に (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 1 別子。このパラメーターの値は、ローカル・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、パートナー・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 1 別子。 このパラメーターの値は、ローカル・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、パートナー・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

割り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

！の値は、基本会話にのみ適用されます。 

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O囲は 0 ～ 32767 です。

アプリケーションは VCB の末 x 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 x にデータを付加できますが、その場合

expd_data_rcvd

優先データ u 信の標 1 。この標 1 は、RECEIVE_EXPEDITED_DATA が / 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、VCB のフォーマット 1 バージョンが必要です。フォーマット 1 VCB のアクセスの詳細については、 43ページの『全二重 VCB』を参照してください。

AP_YES
AP_NO

パラメーター・エラーが原因で verb が B 行されなかった場合は、パーソナル・コミュニケーションズおよび Communications Server は ! 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 B 行されなかった場合は、パーソナル・コミュニケーションズおよび Communications Serverは ! のパラメーターを戻します（マップ 0 会話のみ)。 

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 / 行したときに会話が不適切な状態にあった場合は、パーソナル・コミュニケーションズおよび Communications Server は ! 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 

! に (す 1 ! 戻りコード (**primary_rc**) およびそれに付随する 2 ! 戻りコード (**secondary_rc**) が生成される条 o 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

注: パフォーマンス上の理由から、SNA API クライアントは、サーバーに転送することなく、 **[MC_]SEND_DATA** verb で成功を (す戻りコードを戻すことができます。 その後で **[MC_]DEALLOCATE** verb が / 行されると、 **[MC_]SEND_DATA** はサーバーに転送されて処理されます。**[MC_]SEND_DATA** エラー戻りコードがある場合には、そのコードは **[MC_]DEALLOCATE** verb で戻されます。エラー戻りコードのリストについては、**[MC_]SEND_DATA** verb を参照してください。

[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 構造体

```
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、！のパラメーターをパーソナル・コミュニケーションズおよび 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。 s ブロッキング操作の場合は、このフラグと AP_NON_BLOCKING を OR で k 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 k ぶ必要があります。

format

VCB のフォーマットを 1 別します。 上 - に (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 1 別子。このパラメーターの値は、ローカル・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、パートナー・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 1 別子。 このパラメーターの値は、ローカル・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、パートナー・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 B 行された場合は、パーソナル・コミュニケーションズおよび Communications Server は ! のパラメーターを戻します。

primary_rc
AP_OK

verb が s ブロッキングで完了していない場合は、パーソナル・コミュニケーションズおよび Communications Server は ! 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 B 行されなかった場合は、パーソナル・コミュニケーションズおよび Communications Server は ! 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 / 行したときに会話が不適切な状態にあった場合は、パーソナル・コミュニケーションズおよび Communications Server は ! のパラメーターを戻します。

primary_rc
AP_STATE_CHECK

secondary_rc
AP_FLUSH_NOT_SEND_STATE

! に (す 1 ! 戻りコード (**primary_rc**) およびそれに付随する 2 ! 戻りコード (**secondary_rc**) が生成される条 o 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

注: パフォーマンス上の理由から、SNA API クライアントは、サーバーに転送することなく、 **[MC_]SEND_DATA** verb で成功を (す戻りコードを戻すことができます。 その後で **[MC_]FLUSH** verb が / 行されると、 **[MC_]SEND_DATA** はサーバーに転送されて処理されます。**[MC_]SEND_DATA** エラー戻りコードがある場合には、そのコードは **[MC_]FLUSH** verb で戻されます。エラー戻りコードのリストについては、**[MC_]SEND_DATA** verb を参照してください。

[MC_]GET_ATTRIBUTES

GET_ATTRIBUTES verb は、会話の属性を戻します。

VCB 構造体

```
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     */
                                         /* 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                       */
} 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、！のパラメーターをパーソナル・コミュニケーションズおよび 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 k ぶ必要があります。

format

VCB のフォーマットを 1 別します。 上 - に (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 1 別子。

このパラメーターの値は、ローカル・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、パートナー・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 1 別子。

このパラメーターの値は、ローカル・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、パートナー・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 B 行された場合は、パーソナル・コミュニケーションズおよび Communications Server は！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に関連付けられているネットワーキング特性の名前。これは、8 バイトの Q 数字のタイプ A の EBCDIC ストリング (Q 字で始まるもの) で、 8 バイトに満たない場合は & 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 Q 数字のタイ

プ A の EBCDIC ストリング (Q字で始まるもの) で、 8 バイトに満たない場合は&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Q数字のタイプ A の EBCDIC ストリング (Q字で始まるもの) で、 8 バイトに満たない場合は&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 1 させるための別名。これは、8 バイトの JISCII 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 1 させるための別名。これは、8 バイトの JISCII 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
パートナー LU の s 解 a 名、すなわち、システム・サービス制御点 (SSCP) で定 A 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 の 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&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 k された 2 つのタイプ A の 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・トランザクション・プログラムが、リモート・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 の EBCDIC 文字ストリングで、10 バイトに満たない場合は&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 1 別子。

conv_corr_len
常に 0 にセットされます。

有効O囲: 0〜8

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 B 行されなかった場合は、パーソナル・コミュニケーションズおよび Communications Server は！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

！に (す 1 ！戻りコード (**primary_rc**) およびそれに付随する 2 ！戻りコード (**secondary_rc**) が生成される条 o 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 u 信に変更する前に、この verb は ! のいずれかの verb と同じ処理を行います。

- **FLUSH** verb。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。ローカル LU の送信バッファーのデータ、および確認要 a を、パートナー・トランザクション・プログラムに送ります。 いますぜぱーカル

format
VCB のフォーマットを 1 別します。 上 - に (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 1 別子。このパラメーターの値は、ローカル・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、パートナー・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 1 別子。

このパラメーターの値は、ローカル・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、パートナー・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
どのような方法で状態を変更するかを指定します。

AP_FLUSH
AP_SYNC_LEVEL
AP_P_TO_R_CONFIRM

locks パーソナル・コミュニケーションズおよび Communications Server 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_LONG

AP_SHORT

戻りパラメーター

verb が正常に B 行された場合は、パーソナル・コミュニケーションズおよび Communications Server は ! のパラメーターを戻します。

primary_rc
AP_OK

verb が s ブロッキングで完了していない場合は、パーソナル・コミュニケーションズおよび Communications Server は ! 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 B 行されなかった場合は、パーソナル・コミュニケーションズおよび Communications Server は ! 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 / 行したときに会話が不適切な状態にあった場合は、パーソナル・コミュニケーションズおよび Communications Server は！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

！に (す 1 ！戻りコード (**primary_rc**) およびそれに付随する 2 ！戻りコード (**secondary_rc**) が生成される条 o 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_UNEXPECTED_SYSTEM_ERROR

AP_CANCELLED

注: パフォーマンス上の理由から、SNA API クライアントは、サーバーに転送することなく、 **[MC_]SEND_DATA** verb で成功を (す戻りコードを戻すことができます。 その後で、 **[MC_]PREPARE_TO_RECEIVE** verb が / 行されると、**[MC_]SEND_DATA** はサーバーに転送されて処理されます。**[MC_]SEND_DATA** エラー戻りコードがある場合には、そのコードは **[MC_]PREPARE_TO_RECEIVE** verb で戻されます。エラー戻りコードのリストについては、**[MC_]SEND_DATA** verb を参照してください。

[MC_]RECEIVE_AND_POST

RECEIVE_AND_POST verb は、アプリケーション・データと状況情報を s 同期に u 信します。この verb を用いると、ローカル LU でデータを u 信しているときでも、トランザクション・プログラムは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 / 行できます。

Win 3.1 の SNA API クライアントでは使用できません。

VCB 構造体

```
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、！のパラメーターをパーソナル・コミュニケーションズおよび 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 1 別します。 上 - に (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 1 別子。このパラメーターの値は、ローカル・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、パートナー・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 1 別子。

このパラメーターの値は、ローカル・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、パートナー・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 u 信するときの A 0 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 u 信できるデータの最大バイト数。

有効 O 囲: 0 ～ 65535

この値は、 u 信データが入るバッファーの長さを超えてはなりません。

dptr ローカル LU が u 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 x 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！するイベントのハンドル。 この verb は、Win32 API で WaitForMultipleObjects とともに、または OS/2 で 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B行された場合は、パーソナル・コミュニケーションズおよび Communications Server は！のパラメーターを戻します。

primary_rc

AP_OK

AP_DEALLOC_NORMAL

what_rcvd

データ、会話状態、確認要 a など u 信した情報。 **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！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

！のパラメーターはマップ 0 会話にのみ適用されます。 

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 a u 信の標 1 。

AP_YES

AP_NO

expd_data_rcvd

優先データ u 信の標 1 。この標 1 は、RECEIVE_EXPEDITED_DATA が / 行されるまで、AP_YES にセットされたままです。

AP_YES

AP_NO

このフォーマット・フィールドには、VCB のフォーマット 1 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、43ページの『全二重 VCB』 を参照してください。

dlen u 信したデータのバイト数です (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 u 信しなかったことを (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 u 信したことを (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 B 行されなかった場合は、パーソナル・コミュニケーションズおよび Communications Server は ！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 / 行したときに会話が不適切な状態にあった場合は、パーソナル・コミュニケーションズおよび Communications Server は ！のパラメーターを戻します。

MC_RECEIVE_AND_POST

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 / 行した他の verb により、この verb が h り消された場合は、パーソナル・コミュニケーションズおよび Communications Server は ! のパラメーターを戻します。

primary_rc

AP_CANCELLED

! に (す 1 ! 戻りコード (**primary_rc**) およびそれに付随する 2 ! 戻りコード (**secondary_rc**) が生成される条 o 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_CANCELLED

注: パフォーマンス上の理由から、SNA API クライアントは、サーバーに転送することなく、 **[MC_]SEND_DATA** verb で成功を (す戻りコードを戻すことができます。 その後で、**[MC_]RECEIVE_AND_POST** verb が / 行されると、**[MC_]SEND_DATA** はサーバーに転送されて処理されます。**[MC_]SEND_DATA** エラー戻りコードがある場合には、そのコードは **[MC_]RECEIVE_AND_POST** verb で戻されます。エラー戻りコードのリストについては、**[MC_]SEND_DATA** verb を参照してください。

[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 u 信します。 現在使用可能なデータがない場合、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。

> 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 / 行できます。この場合、LU は + 分の送信バッファーをフラッシュして、バッファー内にあるすべての情報と SEND 標 1 をリモート・プログラムに送ります。そして会話を u 信状態に変更します。 ！に、LU は情報の到着を待！します。 リモート・プログラムは、SEND 標 1 を u け h った後、データをローカル・プログラムに送ることができます。

全二重会話の場合:

- 送信バッファーに会話割り振り要 a が入っている場合、送信バッファーはフラッシュされます。それ以外で、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 u け h 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 / 行してからこの verb を / 行する必要があります。

VCB 構造体

```
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、！のパラメーターをパーソナル・コミュニケーションズおよび Communications Server に提供します。

opcode

AP_B_RECEIVE_AND_WAIT

max_len

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

有効O囲: 0 ～ 65535

この値は、 u 信データが入るバッファーの長さを超えてはなりません。

dptr ローカル LU が u 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 x 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 B 行された場合は、パーソナル・コミュニケーションズおよび Communications Server は！のパラメーターを戻します。

primary_rc

AP_OK

AP_DEALLOC_NORMAL

what_rcvd

データ、会話状態、確認要 a など u 信した情報。 **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！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

！のパラメーターはマップ 0 会話にのみ適用されます。 

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 a u 信の標 1 。

AP_YES

AP_NO

！の verb のフォーマットは、VCB のフォーマット 1 バージョンです。フォーマット 1 VCB のアクセスの詳細については、 43ページの『全二重 VCB』を参照してください。

expd_data_rcvd

優先データ u 信の標 1 。この標 1 は、RECEIVE_EXPEDITED_DATA が / 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、データを u 信したことを (している場合だけです。 u 信したデータのバイト数です (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 u 信しなかったことを (します。

verb が s ブロッキングで完了していない場合は、パーソナル・コミュニケーションズおよび Communications Server は！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 B 行されなかった場合は、パーソナル・コミュニケーションズおよび Communications Server は ! 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 / 行したときに会話が不適切な状態にあった場合は、パーソナル・コミュニケーションズおよび Communications Server は ! 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 

! に (す 1 ! 戻りコード (**primary_rc**) およびそれに付随する 2 ! 戻りコード (**secondary_rc**) が生成される条 o 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 

! の 3 つのパラメーターは基本会話にのみ適用されます。 

AP_DEALLOC_ABEND_PROG
AP_DEALLOC_ABEND_SVC
AP_DEALLOC_ABEND_TIMER

AP_DEALLOC_NORMAL
AP_PROG_ERROR_NO_TRUNC
AP_PROG_ERROR_PURGING

！の 4 つのパラメーターは基本会話にのみ適用されます。 

AP_PROG_ERROR_TRUNCL
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

注: パフォーマンス上の理由から、SNA API クライアントは、サーバーに転送することなく、 **[MC_]SEND_DATA** verb で成功を (す戻りコードを戻すことができます。 その後で、**[MC_]RECEIVE_AND_WAIT** verb が / 行されると、**[MC_]SEND_DATA** はサーバーに転送されて処理されます。**[MC_]SEND_DATA** エラー戻りコードがある場合には、そのコードは **[MC_]RECEIVE_AND_WAIT** verb で戻されます。エラー戻りコードのリストについては、**[MC_]SEND_DATA** verb を参照してください。

[MC_]RECEIVE_EXPEDITED_DATA

このパラメーターは、Communications Server の OS/2 および Windows 3.1 SNA API クライアントではサポートされません。

[MC_]RECEIVE_EXPEDITED_DATA verb は、現在パートナー TP から利用できる優先データを u け h ります。現在、優先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！することなくそれを u け h 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 h まります。

VCB 構造体

```
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、！のパラメーターをパーソナル・コミュニケーションズおよび 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。 s ブロッキング操作の場合は、このフラグと AP_NON_BLOCKING を OR で k 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 k ぶ必要があります。

format

VCB のフォーマットを 1 別します。 上 - に (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 1 別子。

このパラメーターの値は、ローカル・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、パートナー・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 1 別子。

このパラメーターの値は、ローカル・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、パートナー・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・プログラムに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 u 信できるデータの最大バイト数。

有効 O 囲: 0 ～ 86

この値は、 u 信データが入るバッファーの長さを超えてはなりません。

dptr ローカル LU が u 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 x 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 B 行された場合は、パーソナル・コミュニケーションズおよび Communications Server は ! のパラメーターを戻します。

primary_rc

AP_OK

rts_rcvd

送信要 a u 信の標 1 。

AP_YES

AP_NO

expd_data_rcvd
優先データ u 信の標 1 。この標 1 は、RECEIVE_EXPEDITED_DATA が / 行されるまで、AP_YES にセットされたままです。

AP_YES
AP_NO

dlen u 信したデータのバイト数です (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 u 信しなかったことを (します。 u 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) は存在していません。

verb が s ブロッキングで完了していない場合は、パーソナル・コミュニケーションズおよび Communications Server は ! 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 が優先データをサポートしていないために verb が B 行されない場合は、パーソナル・コミュニケーションズおよび Communications Server は ! のパラメーターを戻します。

primary_rc
AP_EXPD_NOT_SUPPORTED_BY_LU

データをすぐにパートナー・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、パーソナル・コミュニケーションズおよび Communications Server は ! 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 からの利用可能な優先データをすべて入れられるほど大きくない場合は、データは戻されず、パーソナル・コミュニケーションズおよび Communications Server は ! のパラメーターを戻します。

primary_rc
AP_BUFFER_TOO_SMALL

dlen LU が u け h ることのできる優先データのバイト数。

パラメーター・エラーが原因で verb が B 行されなかった場合は、パーソナル・コミュニケーションズおよび Communications Server は ! 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 / 行したときに会話が不適切な状態にあった場合は、パーソナル・コミュニケーションズおよび Communications Server は ！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

！に (す 1 ！戻りコード (**primary_rc**) およびそれに付随する 2 ！戻りコード (**secondary_rc**) が生成される条 o 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 u 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 構造体

```
typedef struct 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       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、！のパラメーターをパーソナル・コミュニケーションズおよび 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。 s ブロッキング操作の場合は、このフラグと AP_NON_BLOCKING を OR で k 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 k ぶ必要があります。

format

VCB のフォーマットを 1 別します。上 - に (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 1 別子。

このパラメーターの値は、ローカル・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、パートナー・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 1 別子。

このパラメーターの値は、ローカル・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、パートナー・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 u 信するときの A 0 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 u 信できるデータの最大バイト数。

有効 O 囲: 0 〜 65535

この値は、 u 信データが入るバッファーの長さを超えてはなりません。

dptr ローカル LU が u 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 x 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 B 行された場合は、パーソナル・コミュニケーションズおよび Communications Server は！のパラメーターを戻します。

primary_rc

AP_OK

AP_DEALLOC_NORMAL

what_rcvd

データ、会話状態、確認要 a など u 信した情報。 **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！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 

！のパラメーターはマップ 0 会話にのみ適用されます。 

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

優先データ u 信の標 1 。

AP_YES

AP_NO

rts_rcvd

送信要 a u 信の標 1 。

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 u 信したことを (している場合だけです。 u 信したデータのバイト数です (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 u 信しなかったことを (します。

verb が s ブロッキングで完了していない場合は、パーソナル・コミュニケーションズおよび Communications Server は ! 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 (! のものと OR k 合される)

AP_NON_BLOCKING (! のものと OR k 合される)
AP_OPERATION_INCOMPLETE_FLAG

すぐに使用可能なデータがない場合は、パーソナル・コミュニケーションズおよび Communications Server は ! 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 B 行されなかった場合は、パーソナル・コミュニケーションズおよび Communications Server は ! 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 / 行したときに会話が不適切な状態にあった場合は、パーソナル・コミュニケーションズおよび Communications Server は ！のパラメーターを戻します。

primary_rc
AP_STATE_CHECK

secondary_rc
AP_RCV_IMMD_BAD_STATE

！に (す 1 ！戻りコード (**primary_rc**) およびそれに付随する 2 ！戻りコード (**secondary_rc**) が生成される条 o 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_SVC
AP_DEALLOC_ABEND_SVC
AP_DEALLOC_ABEND_SVC
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

注: パフォーマンス上の理由から、SNA API クライアントは、サーバーに転送することなく、 **[MC_]SEND_DATA** verb で成功を (す戻りコードを戻すことができます。 その後で、**[MC_]RECEIVE_IMMEDIATE** verb が / 行されると、**[MC_]SEND_DATA** はサーバーに転送されて処理されます。**[MC_]SEND_DATA** エラー戻りコードがある場合には、そのコードは **[MC_]RECEIVE_IMMEDIATE** verb で戻されます。エラー戻りコードのリストについては、**[MC_]SEND_DATA** verb を参照してください。

[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の送信を要 a していることを、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 構造体

```
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、！のパラメーターをパーソナル・コミュニケーションズおよび 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。 s ブロッキング操作の場合は、このフラグと AP_NON_BLOCKING を OR で k ぶことができます。

format

VCB のフォーマットを 1 別します。 上 - に (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 1 別子。

このパラメーターの値は、ローカル・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、パートナー・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 1 別子。

このパラメーターの値は、ローカル・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、パートナー・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 B 行された場合は、パーソナル・コミュニケーションズおよび Communications Server は ! のパラメーターを戻します。

primary_rc
AP_OK

verb が s ブロッキングで完了していない場合は、パーソナル・コミュニケーションズおよび Communications Server は ! 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 s ブロッキング・モード (43ページの『待ち行列レベルの s ブロッキング』を参照) で / 行し、送 u 信待ち行列上の verb の処理中に会話が終了した場合は、パーソナル・コミュニケーションズおよび Communications Server は ! 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 / 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 B 行されなかった場合は、パーソナル・コミュニケーションズおよび Communications Server は ! 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 / 行したときに会話が不適切な状態にあった場合は、パーソナル・コミュニケーションズおよび Communications Server は ! 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

! に (す 1 ! 戻りコード（**primary_rc**）が生成される条 o については、付録A. APPC 共通戻りコードで説明します。

MC_REQUEST_TO_SEND

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k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り振りを解除します。 これは、**[MC_]ALLOCATE**、 **[MC_]SEND_DATA**、および **[MC_]DEALLOCATE** (**FLUSH**) の一連の verb を順に / 行するのと同じ働きをします（一Lに「片方向ブラケット」と呼ばれます)。

VCB 構造体

```
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、！のパラメーターをパーソナル・コミュニケーションズおよび 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。 s ブロッキング操作の場合は、このフラグと AP_NON_BLOCKING を OR で k ぶことができます。

format

VCB のフォーマットを 1 別します。 上 - に (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 1 別子。

このパラメーターの値は、ローカル・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、パートナー・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 a 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

割り振られるセッションの会話グループ 1 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 1 させるための別名。これは、8 バイトの JISCII 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

名前は、EBCDIC のピリオドで連 k された 2 つのタイプ A の 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O囲: 0 〜 65535

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 x 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B行された場合は、パーソナル・コミュニケーションズおよび Communications Server は！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1別子。

verb が s ブロッキングで完了していない場合は、パーソナル・コミュニケーションズおよび Communications Server は！のパラメーターを戻します。

primary_rc
AP_OPERATION_INCOMPLETE

opext AP_OPERATION_INCOMPLETE_FLAG

rtn_ctl パラメーターに AP_IMMEDIATE が設定されているときに、すぐに使用できるセッションがない場合は、パーソナル・コミュニケーションズおよび Communications Server は！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B行されなかった場合は、パーソナル・コミュニケーションズおよび Communications Server は！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

！に (す 1 ！戻りコード (**primary_rc**) およびそれに付随する 2 ！戻りコード (**secondary_rc**) が生成される条 o 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 構造体

```
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、！のパラメーターをパーソナル・コミュニケーションズおよび 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。 s ブロッキング操作の場合は、このフラグと AP_NON_BLOCKING を OR で k 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 k ぶ必要があります。

format

VCB のフォーマット。上 - の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 1 別子。

このパラメーターの値は、ローカル・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、パートナー・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 1 別子。

このパラメーターの値は、ローカル・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、パートナー・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 O 囲: 0 〜 65535

dptr ローカル LU の送信バッファーに入れ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 x 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** のほかに別の verb の！能も B 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_LEVE
AP_SEND_DATA_DEALLOC_CONFIRM
AP_SEND_DATA_DEALLOC_ABEND

戻りパラメーター

verb が正常に B 行された場合は、パーソナル・コミュニケーションズおよび Communications Server は！のパラメーターを戻します。

注: パフォーマンス上の理由から、SNA API クライアントは、サーバーに転送することなく、 **[MC_]SEND_DATA** verb で成功を (す戻りコードを戻すことができます。 その後で **[MC_]SEND_DATA** verb が / 行されると、 **[MC_]SEND_DATA** はサーバーに転送されて処理されます。

SEND_DATA エラー戻りコードがある場合は、後続の verb で戻されます。

primary_rc
AP_OK

rts_rcvd
送信要 a u 信の標 1 。

AP_YES

AP_NO

expd_data_rcvd
優先データ u 信の標 1 。 この標 1 は、RECEIVE_EXPEDITED_DATA が / 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 B 行されない場合は、パーソナル・コミュニケーションズおよび Communications Server は ! 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 B 行されなかった場合は、パーソナル・コミュニケーションズおよび Communications Server は ! 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 / 行したときに会話が不適切な状態にあった場合は、パーソナル・コミュニケーションズおよび Communications Server は ! 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 

！に (す 1 ！戻りコード (**primary_rc**) およびそれに付随する 2 ！戻りコード (**secondary_rc**) が生成される条 o 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_RETRY
- 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 構造体

```
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、！のパラメーターをパーソナル・コミュニケーションズおよび 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。 s ブロッキング操作の場合は、このフラグと AP_NON_BLOCKING を OR で k 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 k ぶ必要があります。

format

VCB のフォーマットを 1 別します。 上 - に (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 1 別子。

このパラメーターの値は、ローカル・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、パートナー・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 1 別子。

このパラメーターの値は、ローカル・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、パートナー・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 (します。

AP_PROG
AP_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 u 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 (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 / 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O囲は 0 ～ 32767 です。

アプリケーションは VCB の末 x 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 x 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 0 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 F 用データ・ストリーム (GDS) として A 0 設定する必要があります。 詳細については、 *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 B 行された場合は、パーソナル・コミュニケーションズおよび Communications Server は！のパラメーターを戻します。

primary_rc
AP_OK

rts_rcvd
送信要 a u 信の標 1 。

AP_YES

AP_NO

expd_data_rcvd
優先データ u 信の標 1 。 この標 1 は、RECEIVE_EXPEDITED_DATA が / 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 s ブロッキングで完了していない場合は、パーソナル・コミュニケーションズおよび Communications Server は！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 B 行されなかった場合は、パーソナル・コミュニケーションズおよび Communications Server は！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 / 行したときに会話が不適切な状態にあった場合は、パーソナル・コミュニケーションズおよび Communications Server は！のパラメーターを戻します。

primary_rc
AP_STATE_CHECK

secondary_rc
AP_SEND_ERROR_BAD_STATE

！に (す 1 ！戻りコード (**primary_rc**) およびそれに付随する 2 ！戻りコード (**secondary_rc**) が生成される条 o 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 / 行が v 可されている状態で **[MC_]SEND_ERROR** verb が / 行された場合は、！の戻りコードが生成されることがあります。

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: ！の戻りコードが生成されるのは、**[MC_]SEND_ERROR** verb が SEND 状態で / 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: ！の戻りコードが生成されるのは、verb が RECEIVE 状態で / 行された場合だけです。

AP_DEALLOC_NORMAL

注: パフォーマンス上の理由から、SNA API クライアントは、サーバーに転送することなく、 **[MC_]SEND_DATA** verb で成功を (す戻りコードを戻すことができます。 その後で、**[MC_]SEND_ERROR** verb が / 行されると、**[MC_]SEND_DATA** はサーバーに転送されて処理されます。

[MC_]SEND_DATA エラー戻りコードがある場合には、そのコードは **[MC_]SEND_ERROR** verb で戻されます。エラー戻りコードのリストについては、**[MC_]SEND_DATA** verb を参照してください。

[MC_]SEND_EXPEDITED_DATA

このパラメーターは、Communications Server の OS/2 および Windows 3.1 SNA API クライアントではサポートされません。

[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 の優先送信バッファーに蓄えます。 このデータは、事前に送られた s 優先データよりも早くパートナー・トランザクション・プログラムに到着することができます。

VCB 構造体

```
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、！のパラメーターをパーソナル・コミュニケーションズおよび 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。 s ブロッキング操作の場合は、このフラグと AP_NON_BLOCKING を OR で k 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 k ぶ必要があります。

format

VCB のフォーマットを 1 別します。 上 - に (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 1 別子。

このパラメーターの値は、ローカル・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、パートナー・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 1 別子。

このパラメーターの値は、ローカル・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、パートナー・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 O 囲: 1 〜 86

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 x 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 バイトの長さフィールド (LL) は存在していません。

戻りパラメーター

verb が正常に B 行された場合は、パーソナル・コミュニケーションズおよび Communications Server は ! のパラメーターを戻します。

primary_rc

AP_OK

rts_rcvd

送信要 a u 信の標 1 。

AP_YES

AP_NO

expd_data_rcvd

優先データ u 信の標 1 。 この標 1 は、RECEIVE_EXPEDITED_DATA が / 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 s ブロッキングで完了していない場合は、パーソナル・コミュニケーションズおよび Communications Server は ! 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 が優先データをサポートしていないために verb がB行されない場合は、パーソナル・コミュニケーションズおよび Communications Server は！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B行されなかった場合は、パーソナル・コミュニケーションズおよび Communications Server は！のパラメーターを戻します。

primary_rc
AP_PARAMETER_CHECK

secondary_rc

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 a の通知 (REQUEST_TO_SEND) を u 信したかどうかを = 別します。

VCB 構造体

```
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、！のパラメーターをパーソナル・コミュニケーションズおよび 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 1 別します。 上 - に (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 1 別子。このパラメーターの値は、ローカル・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、パートナー・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 1 別子。

このパラメーターの値は、ローカル・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、パートナー・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 B 行された場合は、パーソナル・コミュニケーションズおよび Communications Server は ! 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 a の通知を u 信したかどうかを (します。

AP_OK

AP_UNSUCCESSFUL

パラメーター・エラーが原因で verb が B 行されなかった場合は、パーソナル・コミュニケーションズおよび Communications Server は ! 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

! に (す 1 ! 戻りコード (**primary_rc**) が生成される条 o 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_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 a の通知 (REQUEST_TO_SEND) を u 信したかどうかを、 s 同期に = 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 が処理中である場合も含めて、いつでも **[MC_]TEST_RTS_AND_POST** を / 行することができます。 **[MC_]TEST_RTS_AND_POST** が戻るのは、送信要 a 通知を u 信したとき、会話が終了したとき、または会話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 / 行できます。

VCB 構造体

```
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、！のパラメーターをパーソナル・コミュニケーションズおよび 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 1 別します。 上 - に (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 1 別子。

このパラメーターの値は、ローカル・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、パートナー・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 1 別子。

このパラメーターの値は、ローカル・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、パートナー・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！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。 このファンクションの詳細については、 Win32 API のプログラミング文書を参照してください。

戻りパラメーター

verb が正常にB行された (すなわち、送信要 a の通知が u け h られた) 場合は、パーソナル・コミュニケーションズおよび Communications Server は！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、パーソナル・コミュニケーションズおよび Communications Server は！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B行されなかった場合は、パーソナル・コミュニケーションズおよび Communications Server は！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

！に (す 1 ！戻りコード (**primary_rc**) が生成される条 o については、付録A. APPC 共通戻りコードで説明します。

AP_CONVERSATION_TYPE_MIXED
AP_UNEXPECTED_SYSTEM_ERROR
AP_CANCELLED

MC_TEST_RTS_AND_POST

第2部 LUA API

第9章 IBM 従来型 LU アプリケーションの紹介

この章では、IBM 従来？論理装置アプリケーション (LUA) アクセス方 0 を紹介し、システム・ネットワーク体O (SNA) との関 8 について説明します。

注: 本書の第 2 部の各章の内容は、以下のシステムが提供する LUA API に関するものです。

- Windows NT 上でB行されている Communications Server
- Communications Server/NT に付属の、OS/2、Windows NT、Windows 95、および Windows 3.1 対~の SNA API クライアント
- パーソナル・コミュニケーションズ Windows 95、Windows NT

これらのシステムが提供するサポートの間に違いがある場合は、その都度、明 - します。

LUA と SNA の概略

IBM LUA アクセス方 0 は、2 ！従属？論理装置 (LU) にアプリケーション・プログラミング・インターフェース (API) を提供します。LUA は、システム・ソフトウェアと、 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入出力 (I/O) サービス・ルーチンを提供するインターフェースから構成されます。 Communications Server は、LUA の RUI および SLI インターフェースをサポートします。

Communications Server は、Microsoft** NT SNA サーバーとのバイナリー互換性を確保するように設W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 B装と似ています。

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！能はw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！能の独 + のサブセットを提供しています。

LUA アプリケーション・プログラムをワークステーション上でB行できるようにするには、事前に Communications Server をインストールし構成する必要があります。Communications Server のインストールおよび構成方法については、 *Communications Server: クィック・スタート* を参照してください。

接続機能

どのような通信システムでも、その g な目的は他のシステムと接続することです。SNA の最終目標は、広O囲にわたるF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 の通信および接続要 o の中には、システム/370* (S/370*) 接続も含まれます。

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！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！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B行されると、リモート・ホスト・アプリケーション・プログラムとの間でデータの交換が行われます。 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・プログラムのことを、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A 0 化要 a 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 a するために verb を / 行します。 LUA verb は、制御ブロックとしてコード化されます。各 verb 制御ブロックには、それぞれ厳密に定Aされたフォーマットがあります。LUA ！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 に提供します。

LUA verb は、常に即時に / 呼T 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 a で指定されている通知方 0 を使用して、 verb が完了するまで待つ必要があります。 LUA verb の通知については、 第12章 RUI 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、**INCLUDE** ディレクトリーに } 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・プログラムを作成するときに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。各 LUA アプリケーション・プログラムは、すべて 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 との間の関 8 は 1 対多です。 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 を所有することはできますが、ある 1 つの LU を所有できるのは 1 時点では 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。 2 つ目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、 1 つ目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 / 行します。 これらの verb により、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 の定Aは、各マシンごとに 1 回行うだけで済みます。その方法については、*クィック・スタート* を参照してください。

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行います。

SNA 層

SNA は、7 つの厳密に定 A 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！能を B 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 o の！能を理解する上で役立ちます。ここでは、 LUA と SNA の関 8 を (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 o の DLC プロトコル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のリンクを提 (します。 DLC 層は、 LUA も含め、すべてのパーソナル・コミュニケーションズおよび Communications Server LU B 装に共通です。

パス制御層

~辺ノードにおける SNA のパス制御 (PC) 層は、ノード内の複数のハーフセッションとの間での P 路指定など、基本的な！能を提供します。 SNA では、PC 層は、一度に 1 つのデータ・リンクとの間でしか P 路指定を行うことができません。 PC 層は、LUA も含め、すべてのパーソナル・コミュニケーションズおよび Communications Server LU B 装に共通です。

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、 k 合点マネージャー！能およびセッション制御！能を提供します。 k 合点マネージャー！能は、シーケンス V 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！能を制御します。 セッション制御！能は、開始、ペーシング、暗号化、復号、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！能について、セッション固有のサポートを提供します。 LUA では、パーソナル・コミュニケーションズおよび Communications Server 内に LU タイプ 0、1、2、および 3 の TC 層の B 装が含まれます。

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 1 対の！能管理データ (FMD) の間での、 FMD 要 a および FMD ~答のフローを制御します。データ・フロー制御層は、要 a / ~答 A 0 設定、データ・チェーニング・プロトコル、要 a / ~答の相関、送信および u 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、ブラケット開始の停止のプロトコル、および待

ち行列~答プロトコルなど、各 o の！能を提供します。 LUA では、パーソナル・コミュニケーションズおよび Communications Server 内に LU タイプ 0、1、2、および 3 のデータ・フロー制御層の B 装が含まれます。

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！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 A されています。 LUA では、パーソナル・コミュニケーションズおよび Communications Server 内に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ます。プレゼンテーション・サービス層の詳細については、*Systems Network Architecture Concepts and Products* を参照してください。

LU サービス！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！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！能は、LU を定 A し SNA セッションを開始および停止するための共通の パーソナル・コミュニケーションズおよび Communications Server 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~する 2 つの LU が、セッションと呼ばれる相互関 8 で k 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 o 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 セッションにより、エンド・ユーザーは互いにデータを交換できるようになります。

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、転送データの量、データ・セキュリティー、ネットワーク P 路指定、データ損：、およびトラフィックの輻輳 (ふくそう) などの事項に関 8 します。セッション特性は、 1 ！ LU より / 行された SNA **BIND** コマンドを 2 ！ LU が u け入れたときに、その **BIND** コマンドの内容によって h 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 ！論理装置 (PLU) と 2 ！論理装置 (SLU) との間の通信から成り立ちます。SLU は、 LUA アプリケーション・プログラムによって B 装されます。 LU-LU セッションにおいて PLU と SLU の間でデータを伝送するには、！のようなイベントが / 生していなければなりません。

1. パーソナル・コミュニケーションズおよび Communications Server がデータ・リンクをアクティブにします。
2. データ・リンクの準 w ができると、 SSCP は、物理装置活動化 (**ACTPU**) コマンドを送り、パーソナル・コミュニケーションズまたは Communications Server プログラムからの肯定~答を読み h ることによって、 SSCP と物理装置との間のセッ

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 ~ していれば、いずれかのプログラムは肯定 ~ 答を送ります。

3. SSCP は、論理装置活動化 (**ACTLU**) コマンドを送り、パーソナル・コミュニケーションズまたは Communications Server プログラムからの肯定 ~ 答を読み h 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)。そして、**ACTLU** コマンドからの LU アドレスが構成情報に対 ~ していれば、いずれかのプログラムは肯定 ~ 答を送ります。

セッションの開始

LU-LU セッションは、SLU または PLU のどちらからでも開始できます。

SLU からの LU-LU セッションの開始

SSCP-LU セッションが確立されると、SLU プログラムは、 SSCP にイニシエイト・セルフ (**INITSELF**) コマンドを送ることにより、 LU-LU セッションを要 a することができます。 SSCP は **INITSELF** コマンドを u けた 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・プログラムは、その名前が認 1 されていてアクティブであれば有効です。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 ~ 答を送り、PLU はセッションを開始します。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] 定 ~ 答を送り、PLU はセッションを開始しません。

SSCP が **INITSELF** コマンドに対して肯定 ~ 答を送ったのに、セッションを確立できないという場合は、 SSCP は ネットワーク・サービス・プロシージャー・エラー (**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 / 行できます。

PLU からの LU-LU セッションの開始

PLU プログラムは s 送信請 a LU-LU セッションを開始できます。PLU は、 **BIND** コマンドを生成することによりセッションを開始します。その後肯定 ~ 答が生じた時点で、通信の合意が成立します。**BIND** コマンドに関連付けられているデータ・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の詳細については、*Systems Network Architecture: Formats* を参照してください。

交渉不可能 **BIND** の場合は、パラメーターが有効であれば、SLU は肯定 ~ 答を戻します。パラメーターが有効でない場合は、SLU は] 定 ~ 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 (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 ~ 答を戻すことができます。 PLU は、戻されたパラメーターを有効と認めた場合は、開始データ・トラフィック (**SDT**) コマンドを送ります。戻されたパラメーターが有効でない場合は、SLU からの交渉可能 **BIND** コマンドのパラメーターが有効でないことを (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 ~ 答すれば、データ転送を開始できます。データ伝送操作では、メッセージは、伝送されるまで、エンド・ユーザーの - 憶域からパーソナル・コミュニケーションズまたは Communications Server の - 憶域に移動します。データ u 信操作では、どちらかのプログラムがメッセージを + 分 + 身の - 憶域に入れ、その後でエンド・ユーザーの - 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 ! 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 u 信側に、データ・チェー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 a します。データ・チェーンは一連の関連するメッセージです。 データ・チェーニングの詳細については、183ページの『データ・チェーニング・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、パーソナル・コミュニケーションズまたは Communications Server は、 Release Quiesce (**RELQ**) コマンドを u け h 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、 u 信側に再びデータが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開始できません。

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。

- + 己終止 (**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。
- シャットダウン要 a (**RSHUTD**) コマンドを送る。このコマンドは PLU からの **UNBIND** (アンバインド) を要 a 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。関与している場合は、SSCP は肯定の s データ ~ 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 n し、 ! 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 , の j 順に従って終了します。 SLU と PLU は対話を交わし、互いにデータの送信を停止するよう指 (し、すでに送信済みのデータを u 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 F 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 s 活動論理装置 (**DACTLU**) コマンドを送ると、 SSCP-LU セッションは終了します。パーソナル・コミュニケーションズおよび Communications Server の最後の SSCP-LU セッションが終了すると、 SSCP は、 s 活動物理装置 (**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

ホストは、**DACTPU** コマンドに対する ~ 答を u 信すると、 SDLC プロトコルの使用時に、 Set Disconnect Response Mode (**SDRM**) コマンドなどのコマンドをパーソナル・コミュニケーションズおよび Communications Server に戻します。また、SSCP は、同じコマンドをパーソナル・コミュニケーションズおよび 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を終了した場合、それまでにアクティブだったすべての SLU が loss-of-contact 標 1 を u け h 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 V に V 号が付きます。 SLU は、SLU から PLU への通常フロー・メッセージのシーケンス V 号と、 PLU から SLU への通常フロー・メッセージのシーケンス V 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 V 号より 1 つ大きいシーケンス V 号が割り当てられます。 SLU と PLU の間に確立される各セッションごとに、一対のシーケンス V 号が割り当てられます。

LU-LU ^ 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、シーケンス V 号の代わりに、シーケンス V 号がないことを (す 1 別子が使用されます。

セッションが再確立される、または **CLEAR** コマンドが送信されると、 PLU および SLU はそれぞれのシーケンス V 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、シーケンス V 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開されたときに、正しいシーケンス V 号にセットすることができます。

シーケンスV号エラーが見つかると、SLU は、~答が要 a されている場合は] 定~答を PLU に送ります。 SLU は、~答を u けh ると、~答シーケンスV号を使用して、その~答を元の要 a に対~付けます。SLU は、~答を作成する場合、元の要 a のシーケンスV号を提供する必要があります。

セッションの再開と再1期化

PLU または SLU で、回線障害などのような回復不能エラーが / きた場合には、LU-LU セッションを再開した後でセッションを再同期することが必要になる場合があります。 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・シーケンスV号の再設定 (必要に~じて) が含まれます。アプリケーション・プログラムには、消:したメッセージを再送するためのルーチンを組み込むことができます。

セッションが再開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 の両方に有効なシーケンスV号を確立するためにダイアログが / 生します。このダイアログは、一連の **STSN** メッセージと肯定~答で成り立ちます。

再同期が必要と=断した場合、SLU は、 Request Recovery (**RQR**) コマンド、] 定~答、または LU-Status Command (**LUSTAT**) を、ユーザー・センス・バイトに入れて障害の - 述とともに送ることができます。 PLU が障害を / 見した、または SLU から **RQR** コマンドをu信した場合、 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nし、 **STSN** コマンドを送って新しいシーケンスV号を設定し、 **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 o のプロトコルによって、要 a および~答の順序に関する , 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

パーソナル・コミュニケーションズおよび Communications Server またはホストにとってメッセージ・フローが速すぎ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、パーソナル・コミュニケーションズおよび 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、~答を待ってから後続のメッセージを送るようにします。 ペーシングは、パーソナル・コミュニケーションズおよび Communications Server からホストへのフロー、ホストからパーソナル・コミュニケーションズおよび 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 セッションが開始されると、LUA がすべてのペーシングを管理するので、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。

u 信ペーシング・プロトコル

u 信ペーシング・プロトコルを使用すれば、PLU は、 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 u け h ると、パーソナル・コミュニケーションズおよび Communications Server は + 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 ~ 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、パーソナル・コミュニケーションズおよび Communications Server は、要 a / ~ 答ヘッダー (RH) の中で、ペーシング ~ 答が戻されることを (すビットをセットします。いずれかのプログラムが PLU からのペーシング ~ 答を u け h 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、どちらのプログラムもそれ以上データ・メッセージを送信できません。アプリケーション・プログラムが書き込み操作を / 行し、ペーシング ~ 答が u 信されなかった場合は、パーソナル・コミュニケーションズおよび Communications Server はその書き込み操作を d 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 + 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 1 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 h るときに、ペーシング ~ 答を / 行します。 ペーシング標 1 はメッセージ ~ 答に含めることができます。あるいは、u 信メッセージについて ~ 答が必要ない場合は、分離ペーシング ~ 答 (IPR) として送ることができます。 その場合、 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 s 重コンテンション/フリップフロップ・プロトコルの使用

！のどちらのプロトコルでも方向転換 (CD) 標 1 が使用されます。

- > 二重コンテンション・プロトコル。これは通常フロー送 u 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、またはチェーンの最後の要 a の送信または u 信の後に、通常フロー要 a を送ることができます。
- > 二重フリップフロップ・プロトコル。これは通常フロー送 u 信モードの 1 つで、一方のハーフセッションが、チェーン終了の時点で ~ 答ヘッダー (RH) 内で CD 標 1 をセットして、相 j 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 1 は、送信を開始できることを u 信側に知らせます。

たとえば、SLU はトランザクションを開始する場合に、まず、そのトランザクションを完全に - 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 ~ 答の送信を開始できることを (す CD 標 1 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 1 をセットします。 トランザクションが完了するまで、この > 二重モードでダイアログが進められます。 > 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 (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 (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トランザクションによって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 O 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 1 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 1 が含まれています。 1 つのメッセージに両方の標 1 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 をビッダーとして指定します。ファースト・スピーカーは、相 j の LU からの v 可なくブラケットを開始できます。しかし、ビッダーがブラケットを開始するには、ファースト・スピーカーに v 可を要 a し v 可を u ける必要があります。

BID コマンドは、ビッダーがブラケット開始の v 可を要 a するために / 行する通常フロー要 a です。**BID** コマンドに対する肯定 ~ 答は、ファースト・スピーカーがブラケットを開始せずに、ビッダーによるブラケットの開始を待つことを (します。 **BID** コマンドに対する] 定 ~ 答は、ファースト・スピーカーが、ビッダーによるブラケット開始の v 可を q] したことを意味します。 ファースト・スピーカーは、ブラケット開始の v 可を与えるときに、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] 定 ~ 答を送るときに、 ! の 2 つの ~ 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 v 可する) ことを (します。ビッダーは、**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 (します。 ビッダーは、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。

ビッダーは、**BID** コマンドの後に BB 標 1 を含む先頭チェーン FMD を送る代わりに、 BB 標 1 を含む先頭チェーン 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 ~ 答によりその試行を v 可するか、いずれかの] 定 ~ 答コードを (す] 定 ~ 答により v 可を q] することができます。 ただし、ビッダーが **CANCEL** コマンドを送信することにより、 BB 標 1 を含むチェーンを停止した場合は、 ~ 答に関 8 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、ビッダーにブラケット開始の v 可を与えるため、またはビッダー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 ~ 答は、ビッダーが ! のブラケットを開始するつもりであることを (します。 ブラケットの開始を望まない場合は、ビッダーは、「RTR 不要」センス・コードを < う] 定 ~ 答を / 行します。

データ・チェーニング・プロトコルの使用

データ・チェーニング・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 からチェーン・メッセージを送信するには、SLU は、チェーン内の最初のメッセージについて、チェーン開始 (BC) 標 1 を 1 にセットします。 チェーン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 1 およびチェーン終了 (EC) 標 1 をどちらも 0 にセットします。チェーン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 u け h ると、チェーニング標 1 を調べて、メッセージがチェーン状態になっているかどうかを = 別します。

データ・チェーニング・プロトコルは、 ! に (す 3 o 類のチェーンから成っています。

- 無 ~ 答チェーン。チェーン内の各要 a に*無 ~ 答* のマークが付けられます。
- 例外時 ~ 答チェーン。チェーン内の各要 a に*例外時 ~ 答* のマークが付けられます。
- 確定 ~ 答チェーン。チェーン内の最後の要 a に*確定 ~ 答* のマークが付けられ、チェーン内の他のすべての要 a には*例外時 ~ 答* のマークが付けられます。

PLU にメッセージ・チェーンを送るとき、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、このチェーン内ですでに u 信済みのすべてのメッセージを破棄します。 チェーン内の要素のどれかに対して PLU が] 定 ~ 答を送った場合は、 SLU は、チェーン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 0

SNA セッションは、データ交換に関する一定の , 則のもとに行われます。

フロー・プロトコル

トランスポート・レベルでは、データ交換は > 二重 (HDX) プロトコルまたは全二重 (FDX) プロトコルに基づいて行われます。

> 二重プロトコルが使用される場合、データは一回につき一方向に流れ、片方の LU は送信のみ、もう一方の LU は u 信のみとなります。 > 二重フリップフロップ・プロトコルの場合、 LU は両方とも、どちらの LU が送信権または u 信権を持っているかを認 1 しています。パートナー LU は、指定された回数だけデータの流れる方向が変更されることを認 1 しています。これにより、 u 信側がデータを送信したり、送信側が u 信したりすることができます。

全二重プロトコルが使用される場合、データはいつでもどちらの方向にでも流れます。 LU は両方とも、制約なしでデータの送 u 信ができます。

応答モード

各 SNA メッセージは要 a または ~ 答のいずれかです。一方の LU からの要 a は、パートナー LU からの合致する ~ 答を引き出します。これは、~ 答には要 a と同じ伝送シーケンス V 号があり、そのシーケンス V 号によって ~ 答および要 a の突き合わせが行われるからです。

アプリケーションが、RH によって強制 ~ 答が指定されている要 a を u 信した場合、アプリケーションは ~ 答メッセージを生成し送信しなければなりません。~ 答モード , 則は、~ 答がいつ送信されるべきかを h 定します。

即時 ~ 答モードでは、まず要 a に対する ~ 答を送信し、! に + 分の要 a を送信しなければなりません。しかし、遅 d ~ 答モードでは、要 a を u 信した後いつでも ~ 答を送信できます。

LUA 相関テーブル

LUA は着信および / 信要 a のシーケンス V 号のトラックを保持します。トラックの保持は、着信および / 信要 a に対する ~ 答があるまで、つまり、アプリケーションが着信要 a に対する ~ 答を / 行する、または PLU が / 信要 a に対して ~ 答するまで続けられます。これらの V 号は、*相関テーブル*と呼ばれるパーソナル・コミュニケーションズおよび Communications Server のエリアに - 録されます。

即時 ~ 答モードでは、セッションで少数の（通常は 1 つ）未処理の要 a のみが行われます。遅 d ~ 答モードでは、より多くの要 a が可能です。

LUA 相関テーブルは動的に管理されます。 LUA は ~ 答をいくつでも - 録できます。~ 答が s 常に多くたまっていると (プログラム・ロジック・エラーが原因の場合が多い)、サーバーのメモリーでの処理速度が遅くなり、パーソナル・コミュニケーションズおよび Communications Server がシャットダウンする可能性があります。

例外時応答要求 (RQE)

通常、LUA はプログラムの g 助なしで、要 a と ~ 答を + 動的に関連付けます。 LUA は、データ・フローの中の要 a ~ 答単位 (RU) を観察しています。 LUA は、要 a が ~ 答を必要とする時および ~ 答が送信された時を通知できます。しかし、LUA が ~ 答が送信される時を通知できない場合が 1 つだけあります。その場合、プログラムが知らせなければなりません。

要 a の RH のビット・フィールドで、~ 答が必須、必要なし、または任意選択のいずれかを指定します。~ 答の必要がない場合、LUA は相関テーブルに要 a V 号を保管する必要がありません。強制 ~ 答はフローの ! のメッセージとして送信されなければなりません。 LUA は相関テーブルにメッセージを入力しますが、! に ~ 答がなければならないので、メッセージはすぐに消 n されます。

RH のエラー ~ 答標 1 (ERI) は、~ 答が任意選択で、u 信した LU が RU を u 諾または処理できない場合に要 a されます。この任意選択 ~ 答の RU は、例外時 ~ 答要 a (RQE) と呼ばれます。 RQE がある場合、LUA は相関テーブルをいつも + 動的に管理するというわけではありません。 表 11 に、 LUA が u 信した RQE を相関テーブルから + 動的に消 n できるインスタンス、および u 信した RQE を消 n する前にアプ

リケーションからのシグナルを待たなければならないインスタンスについてまとめます。

表 11. RQE の消 n

即時 ~ 答モード	遅 d ~ 答モード			
verb	HDX	FDX	HDX	FDX
RUI_READ	+ 動	+ 動	アプリケーションの ~ 答待ち	アプリケーションの ~ 答待ち
RUI_WRITE	+ 動	アプリケーションの ~ 答待ち	アプリケーションの ~ 答待ち	アプリケーションの ~ 答待ち

HDX または FDX セッションの即時 ~ 答モードでは、アプリケーションが入力を要 a すると（RUI_READ を使用すると）、 LUA は RQE V 号をすぐに廃棄できます。これは、即時 ~ 答モードでは、 ~ 答は ! の要 a が / 行される前に送信されなければならないからです。同様に、HDX 接続の即時 ~ 答モードでは、アプリケーションが出力を要 a すると（RUI_WRITE を使用すると）、 LUA は RQE V 号をすぐに廃棄できます。これは、出力が RQE ~ 答または ~ 答なしのいずれかとなるからです。

上 - 以外のすべてのインスタンスでは、 LUA は RQE に対する ~ 答が行われるかどうか確認できません。アプリケーションは肯定 ~ 答をフォーマットし、RQE に対して送信しなければなりません。これは、] 定 ~ 答のみを u け入れる PLU のためでなく、LUA に RQE が u け入れられ、] 定 ~ 答は生成されないことを知らせるためでもあります。

その後、LUA はテーブルから RQE を消 n できます。 ~ 答が肯定であり、PLU は] 定 ~ 答のみを u け入れるので、 LUA はネットワークにアプリケーションの ~ 答を伝送しません。

つまり、単に LUA を g 助するために、アプリケーションは u 信した RQE RU を確定 ~ 答 RU のように扱わなければなりません。

セッション・プロファイル

特定の SNA プロトコルおよび , 約は、セッション内で使用され、共にセッションの『プロファイル』 を構成します。伝送サービス (TS) プロファイルおよび ! 能管理 (FM) プロファイルという、 2 つのプロファイルがセッションに関連付けられています。プロファイルの選択は、BIND 時に行われます。

TS プロファイル

5 つの TS プロファイル、つまり 1、2、3、4、および 7 が SNA によって定 A されています。しかし、TS プロファイル 1 は SSCP と PU の間でのみ使用されるので、LUA アプリケーションで使用可能なのは、プロファイル 2、3、4、および 7 です。表 12 に (すように、 SNA コマンドによって相違があります。

表 12. 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	使用する	使用しない	使用しない	使用しない	使用しない

表 12. TS プロファイルの特性 (続き)

プロファイル	ペーシングの使用	CLEAR	CRV	RQR	SDT	STSN
3	常時	使用する	任意選択	使用しない	使用する	使用しない
4	常時	使用する	任意選択	使用する	使用する	使用する
7	任意選択	使用しない	任意選択	使用しない	使用しない	使用しない

FM プロファイル

8 つの FM プロファイル、つまり 0、2、3、4、6、7、18、および 19 が SNA によって定Aされています。 しかし、プロファイル 0 および 6 は SSCPでのみ使用され、プロファイル 19 は LU タイプ 6.2 でのみ使用されるので、 LUA アプリケーションで使用可能なのは、残りの 5 つのプロファイルです。プロファイルによってSNA ！能の制限が異なります。

FM プロファイルの要約を186ページの表 13 に (します。表の空白部分は、SNA ！能がそのプロファイルで制限されていないことを意味しています。つまり、どのような使用も可能で、BIND パラメーターで指定できるということです。

LUA RUI は、FM プロファイル 2、3、4、7、および 18 をサポートします。

表 13. FM プロファイルの特性

SNA 機能	FMP 2	FMP 3	FMP 4	FMP 7	FMP 18
要 a モード	SLU は遅 d モードを使用				
～答モード	SLU は即時モードを使用	即時モード	即時モード	即時モード	即時モード
RU チェーン	単一 RU チェーンのみ				
長さチェックの圧縮				LU 0 のみ	
FMH-1 セッション制御ブロック (SCB) 圧縮	v 可しない				
データ・フロー制御 RU の v 可	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

表 13. 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
FM ヘッダー	v 可しない				
ブラケット	制限付き使用				
フロー・プロトコル	FDX				
回復	PLU によってのみ				

RUI 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 a された！能を B 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

！に、アプリケーションが使用できる 7 つの LUA verb の概要を (します (各 verb の詳細については、 第13章 RUI verb を参照してください)。

RUI_BID
ホストからの情報が読み h 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
処理中の **RUI_READ** verb を h 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 u 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 セッションがアクティブのときは、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、パーソナル・コミュニケーションズおよび Communications Server は LUA アプリケーションに RUI_INIT_STATUS 標 1 を戻すことができます。このため、処理中の **RUI_INIT** verb の処理中に状態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 o が原因で SSCP-LU セッションが : 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

188ページの表 14は、RUI アプリケーション・プログラムが、特定の LU の RUI API に verb を / 行するための、有効な条 o を (しています。左端の欄の項目は着信 verb を (します。一V 上の行の項目は、 B 行中の verb を表しています。 表の中の項目が「OK」であれば、その verb の組み合わせは有効な条 o であることを表しています。表の中の項目が「エラー」であれば、その verb の組み合わせは不適正な条 o であることを表し、 LUA アプリケーション・プログラムにはエラー・コードが戻されます。

表 14. RUI verb の条 o

	実行中のコマンド						
着信コマンド	**現行セッションがない場合**	**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 つまでアクティブにできます。ただし、これらのアクティブな **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 ! の 4 つがあります。
   - SSCP-LU ^ 送
   - SSCP-LU 通常
   - LU-LU ^ 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 つまでアクティブにできます。ただし、これらのアクティブな **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 1 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 u 信が必要なので、完了までにある程度時間がかかります。 このために、LUA が s 同期インターフェースとして B 装されているのです。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 / 行することも含めて、 + 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 j 段として、 verb の中でイベント・ハンドルを使用します。

パーソナル・コミュニケーションズおよび Communications Server の verb の ~ 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)、その verb を s 同期で / 行する必要があります。 API は、これを行うために、1 ！戻りコードに LUA_IN_PROGRESS、そして **lua_flag2** に LUA_ASYNC をセットします。これで、アプリケーションは、他の処理を B 行する、または、 verb の完了を (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、 VBC の 1 ！戻りコードに最終値がセットされ、**lua_flag2** には LUA_ASYNC がセットされたままになります。

LUA 通信順序のサンプル

！の (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、および送 u 信される SNA メッセージを (しています。矢印は、SNA メッセージ・フローの方向を (します。

！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、 ^ 送フロー

+rsp (されているメッセージに対する肯定 ~ 答

BC チェーン開始

MC チェーン中間

EC チェーン終了

CD 方向転換標 1 設定

RQD 確定 ~ 答要 a

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	<-----

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 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	<-----
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 j 順に従います。

1. **RUI_INIT** verb を / 行して、 LU-SSCP セッションを確立します (**RUI_INIT** verb は、パーソナル・コミュニケーションズおよび Communications Server プログラムがホストから ACTLU メッセージを u 信し、肯定 ~ 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各プログラムで処理され、 LUA アプリケーションには提 (されません)。
2. SSCP に **INITSELF** メッセージを送って、 **BIND** を要 a し、 ~ 答を u 信します。
3. ホストから **BIND** メッセージを u 信し、 ~ 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 u 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 (します。
5. 3 つの RU (最後のものは確定 ~ 答が要 a されることを (す) から成るデータのチェーンを送り、 ~ 答を u 信します。
6. 3 つの RU から成るデータのチェーンを u 信し、 ~ 答を送信します。
7. ホストから **UNBIND** メッセージを u 信し、 ~ 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 / 行して、LU-SSCP セッションを終了します (パーソナル・コミュニケーションズおよび Communications Server プログラムはホストに NOTIFY メッセージを送信し、肯定 ~ 答を待 ! します。ただし、これらのメッセージは、各プログラムで処理されるので、 LUA アプリケーションには提 (されません)。

BIND 検査

LU-LU セッションの初期化中に、ホストは **BIND** メッセージをパーソナル・コミュニケーションズおよび Communications Server LUA アプリケーションに送信します。このメッセージには、 LU-LU セッションで使用する RU サイズなどの情報が入っています。パーソナル・コミュニケーションズおよび Communications Server はこのメッセージを **RUI_READ** verb によって 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の確認は、LUA アプリケーションが行います。アプリケーションには！の選択肢があります。

- **BIND** に対する OK ～答を含む **RUI_WRITE** verb を / 行することにより、 **BIND** をそのまま u け入れる。～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 ～答を含み、変更された **BIND** をデータとして含む **RUI_WRITE** verb を / 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、] 定～答を含む **RUI_WRITE** verb を / 行することにより、 **BIND** を q] する。

RUI_WRITE verb の詳細については、 第13章 RUI verbを参照してください。

注: BIND のパラメーターの妥当性検査と、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の確認は、 LUA アプリケーションが行います。ただし、！の 2 つの制約条 o が適用されます。

- パーソナル・コミュニケーションズおよび 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 q] します。
- パーソナル・コミュニケーションズおよび Communications Server は、**BIND** を用いて 2 ！ LU がコンテンション勝 T であり、エラー回復がコンテンション敗 T の責任であることを指定する必要があります。

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、！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 a に対して] 定～答を送るときは、] 定～答の理由を (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！のようにアプリケーションに報告されます。
  - 1 ！戻りコードは LUA_OK です。
  - 要 a / ～答標 1 、～答タイプ標 1 、およびセンス・データ組み込み標 1 (SDI) が、すべて 1 (センス・データを含む] 定～答を (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。
- パーソナル・コミュニケーションズおよび Communications Server はホストから誤ったデータを u 信すると、ホストに対し] 定～答を戻し、LUA アプリケーションにはそのデータを渡しません。 これは、後続の **RUI_READ** または **RUI_BID** verb で、！のようにアプリケーションに報告されます。
  - 1 ！戻りコードは、LUA_NEGATIVE_RSP です。
  - 2 ！戻りコードは、ホストに送られた SNA センス・コードです。
- 場合によっては、パーソナル・コミュニケーションズおよび 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、どのセンス・コードを送信すればよいのかを h められないことがあります。 このような場合は、パーソナル・コミュニケーションズは、 **RUI_READ** verb を / 行したときに、誤ったデータを例外要 a (EXR) に入れて、！のような方法で LUA アプリケーションに渡します。
  - 要 a / ～答標 1 を、要 a を (す 0 にセットします。

– センス・データ組み込み標 1 (SDI)を、センス・データが含まれていることを (す 1 にセットします (通常この標 1 は ~ 答の場合のみ使用)。
– メッセージ・データを SNA センス・コードと置き換えます。

アプリケーションは、このメッセージに対して] 定 ~ 答を送る必要があります。アプリケーションは、パーソナル・コミュニケーションズおよび Communications Server から提 (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。

- パーソナル・コミュニケーションズおよび 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 (すために、そのアプリケーションにセンス・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、 ! のようにアプリケーションに報告します。
  – 1 ! 戻りコードは LUA_UNSUCCESSFUL です。
  – 2 ! 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 ! 戻りコードとの区別

センス・コードでない 2 ! 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 (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 1 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・コードの詳細については、*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に V 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 1 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 (これは **BIND** により h 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。 その原因は、パーソナル・コミュニケーションズおよび Communications Server がそれ以上のデータを送るために、ホストからのペーシング ~ 答を待たなければならないところにあります。

LUA アプリケーションを使用して、ホストとの間で一方向に大量のデータを転送する場合 (たとえばファイル転送アプリケーションの場合) には、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 u 信するノードでデータがあふれたり、データ - 億域が不足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 0 的な肯定応答

パーソナル・コミュニケーションズおよび Communications Server は、アプリケーションから送られた ~ 答と正しい要 a とを関連付けるために、ホストから u け h った要 a の - 録を保持しています。アプリケーションが ~ 答を送ると、パーソナル・コミュニケーションズおよび Communications Server プログラムは、その ~ 答を元の要 a からのデータと関連付けた上で、関連する - 憶域を解放することができます。

ホストが例外時 ~ 答のみ ( ] 定 ~ 答は送信できるが肯定 ~ 答は送信できない) を指定している場合でも、パーソナル・コミュニケーションズおよび Communications Server は、アプリケーションが後で] 定 ~ 答を送信する場合に w えて、要 a の - 録を保持する必要があります。 アプリケーションが ~ 答を送信しなかった場合は、この要 a に関連した - 憶域を解放することができません。

そのため、パーソナル・コミュニケーションズおよび Communications Serverは、ホストからの例外時 ~ 答のみの要 a に対しても、 LUA アプリケーションは肯定 ~ 答を / 行することができます (これは A 0 的な肯定 ~ 答として知られています)。この ~ 答はホストに送られるものではなく、パーソナル・コミュニケーションズおよび Communications Server が要 a に関連した - 憶域を消 n するために使用するものです。

チェーン終了までのデータの除去

ホストが LUA アプリケーションに要 a 単位のチェーンを送るとき、アプリケーションは、チェーン内の最後の RU を u 信するまで待ってから ~ 答を送信する場合と、チェーンの最後ではない RU に対して] 定 ~ 答を送る場合があります。チェーンの途中で] 定 ~ 答が送られた場合は、パーソナル・コミュニケーションズおよび Communications Server は残りのすべての RU をこのチェーンから除 n し、アプリケーションには送りません。

パーソナル・コミュニケーションズおよび Communications Server は、チェーン内の最後の RU を u け h ると、そのことをアプリケーションに知らせるために、**RUI_READ** または **RUI_BID** verb の 1 ！戻りコードを LUA_NEGATIVE_RSP にセットし、 2 ！戻りコードを 0 にセットします。

注: ホストでは、チェーンの途中で CANCEL などのメッセージを送信することにより、チェーンを終了することができます。 この場合は、 **RUI_READ** verb で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 アプリケーションで使用される各 LU は、パーソナル・コミュニケーションズおよび Communications Server NOF verb または SNA ノード構成プログラムで構成しなければなりません (詳細については、*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、構成には LUA LU プールが含まれていることもあります。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 の数よりアプリケーションの方が多いときに先着順で LU を割り振る場合や、異なるリンクで異なる LU を選択できるようにする場合に使用できます。

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、アプリケーションは、セッションを開始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 / 行し、セッションの開始を望んでいることをパーソナル・コミュニケーションズおよび Communications Server に知らせます。 この名前は、 *システム管理プログラミング* であらかじめ定 A されている LUA LU または LU プールの名前と一致しなければなりません。パーソナル・コミュニケーションズおよび Communications Server は、この名前を！のように使用します。

- 指定された名前がプール内にない 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、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 (ある場合) 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 アプリケーションがクライアント・ワークステーションでB行される場合には、LUA セッションがローカル・ワークステーションにも定Aされていなければなりません。この LUA セッション名には、複数の通信サーバーおよび LUA 定Aを含めることができます。したがって、接続が使用不能になった時に SNA クライアント・コードを新しいサーバーにロールオーバーすることができます。

第10章 RUI LUA Verb の機能

この章では、LUA verb についての以下の特 l なケース、および使用法のヒントを説明しています。

- 例外要 a の処理 — ご使用のプログラムに] 定 ~ 答を出させる LUA からの要 a
- プログラム設Wによる LAN トラフィックの最小化
- LUA verb の無期限保留の処理
- セッション障害からの回復

例外要求の処理

RUI および SLI の両方とも、数個のプロトコルの状態をモニターし、かつ RU のフォーマットを妥当性検査します。インターフェースが 1 ！論理装置 (PLU) からの間違った RU 着信を検出した場合、] 定 ~ 答を出す必要があります。 LUA が、着信 RU を例外要 a (EXR) としてフォーマットして、この検出されたエラーをアプリケーションに通知します。 EXR は、送信権要 a verb (**RUI_BID** もしくは **SLI_BID**) または、入力 verb (**RUI_READ** もしくは **SLI_RECEIVE**) 上でプログラムに送達されます。 EXR は、要 a ヘッダー (RH) 内の以下の条 o によって (されます。

- 0 に設定された *lua_rh.rri*（RU は要 a 単位である）
- 1 に設定された *lua_rh.sdi*（センス・データが組み込まれている）

これは、RH ビットの異常な組み合わせです。センス・データは通常、 ~ 答 RU の内容であって、要 a RU の内容ではありません。LUA はこの異常な組み合わせを使って、 PLU が明らかにエラーを生じさせたという異常な事 B をプログラムにアラートとして渡します。 4 バイトのセンス・コードは、EXR の一部で、検出されたエラーを (します。センス・データに加えて、LUA は最大 3 バイトのオリジナルの RU を戻します。

Verb レコードの変更

アプリケーションでは、EXR を] 定 ~ 答としてフォーマットし、使用中の API に ~ じてどちらかの RUI_WRITE を使って PLU にそれを送信する必要があります。 EXR 入力を ~ 答出力に変換するためには、verb レコードで以下の変更を行います。

- *lua_rh.rri* を 1 に設定する（これが ~ 答であることを (す）。
- *lua_rh.ri* を 1 に設定する（] 定 ~ 答であることを (す）。
- *lua_flag2* の値に基づいて、 *lua_flag1* に適切なデータ・フロー・フラグを設定する。
- *lua_message_type* を LUA_MESSAGE_TYPE_0 に設定する。
- 使用中の API に基づいて、 *lua_opcode* を LUA_OPCODE _RUI_WRITE に設定する。
- *lua_data_length* を 4（センス・データの長さ）に設定する。

- *lua_data_ptr* をセンス・データのアドレスに設定する（位置は EXR を検出した verb によってｈまる。verb が RUI_BID のときはセンス・データは verb レコードの「ピーク・バッファー」にあり、verb が RUI_READ のときはセンス・データは入力バッファーにある）。
- *lua_max_length* を 0 に設定する。

これで、プログラムで verb レコードと EXR 用のバッファーを使用し、 RUI_WRITE を開始して] 定 ~ 答を送信することができます。

ブラケット送信権要求拒否の処理

1 つのケースを除いて、EXR で LUA によって提供されるセンス・コードは、 PLU に戻るのに適切な唯一のセンス・コードです。しかし、ブラケットを使用していて、 PLU がスピーカーになるよう a める場合、アプリケーションはセンス・コードを選択できます。

- LUA は、PLU からの BID コマンドを q] することができます。BID を q] するためには、 LUA がセンス・コード LUA_BB_REJECT_NO_RTR を含む EXR をフォーマットします。このセンス・コードは、ブラケット送信権要 a が q] されており、 RTR コマンドはその後出されないということを伝えます。このセンス・コードの数値は 0x00001308L です（Intel** A 0 、または C プログラムでコーディングする場合はバイト・スワップ A 0 ）。
- BID コマンドがブラケットをサポートしていて、後で RTR コマンドを出すことができる場合、アプリケーションは BID コマンドを u け入れることができます。 BID を u け入れることができることを PLU に通知するために、センス・コードを LUA_BB_REJECT_RTR（値 0x00001408L）に変更することができます。このセンス・コードは RTR がすぐに来ることを伝えます。この少し後に、アプリケーションは 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 トラフィックの最小化

アプリケーションをクライアント・ワークステーション上で B 行する必要がある場合、『送信権要 a ロジック』 の使用を削減することによって、アプリケーション LAN トラフィックのオーバーヘッドを最小化するようアプリケーションを設Wすることができます。

RUI_BID の使用の削減

verb RUI_BID は、データ単位がサーバーで使用できるようになるまで待！してから、完了します。 RUI_BID が完了すると、データが特定のフローで使用できること、および特定の長さを持つことがプログラムに通知されます。その後プログラムはバッファーを割り振り、データに対して RUI_READ verb を出すことができます。

送信権要 a verb とその後に続く入力 verb を出すと、以下の 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
- 送信権要 a 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 1 つのステップで同じジョブを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！する場合、 2 つの LAN メッセージが省略されます。

『送信権要 a ロジック』 の唯一の利点は、メッセージを u けh る前にそのサイズがわかるということです。これにより、必要なバッファーの大きさがわかるまでデータ・バッファーの割り振りを d 期することができます。入力 verb のみを使用する場合、送信権要 a が完了した後にバッファーを割り振るのではなく、あらかじめ最大バッファー・サイズを知っておく必要があります。

保留の処理

RUI verb が完了するかどうかは、PLU アプリケーション、ホスト・システム、ネットワーク、およびパーソナル・コミュニケーションズおよび Communications Serverのアクションによって h まります。これらのうちいずれかのアクションの ~ 答がゆっくりだったり、 ~ 答に : 敗した場合、 verb は無期限に保留になる場合があります。プログラムを設Wするときに、ユーザーやプログラムに保留された verb を終了する方法を (すことによって、保留に対処することができます。

RUI_INIT の h り消し

RUI_INIT verb は、割り当てられた LU をホストがアクティブにするまで中断されます。 通常、アプリケーションが始動する前にホストは ACTLU コマンドを送信しますが、必ずしもそれを行うとは限りません。 アプリケーションが始動するときに、メインフレームがダウンしたり、まだ初期化の最中である場合があります。

プログラムが中断された RUI_INIT を h り消す必要がある場合、RUI_TERM verb を出すことができます。

RUI_WRITE の h り消し

ペーシング使用中に、出力が保留される場合があります。ホストが一時的にデータの読み h りを停止したり、ペーシング ~ 答の伝送に : 敗した場合、ペーシング・ウィンドウがオープンするのを待！して RUI_WRITE が保留されることがあります。

保留された RUI_WRITE をプログラムが h り消す必要がある場合、 RUI_TERM を使ってセッションをクローズする必要があります。

RUI_READ の h り消し

入力 verb は通常、verb が指定したフロー上に入力が到着するまで保留されます。プログラムは RUI_PURGE を使って、保留中の RUI_READ を h り消すことができます。セッションをクローズすると、保留中の入力 verb も h り消されます。

verb 完了の確認

プログラムは、verb の完了を誤って処理すると、無期限待！の様相を呈する場合があります。プログラムが verb を開始し、verb が同期して完了したことに $ 付かず、 s 同期完了を待！してしまうと、プログラムは J Wに待！します。

RUI エントリー・ポイントは、その明 (の k 果として、 B行された verb の 1 ！戻りコードを戻します。 verb が s 同期に完了したかどうかを知らせる最も簡単な方法は、この明 (の戻りが LUA_IN_PROGRESS であるかどうかをテストすることです（図 8 を参照)。

```
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 完了のテスト

データの圧縮

RUI API および SLI API インターフェースの両方ともデータ圧縮をサポートします。データ圧縮の使用は、セッションごとに BIND および BIND ~答を使用して折衝されます。セッションで使用するために圧縮の折衝が行われると、 LZ9 またはランレングス符号化 (RLE) 圧縮アルゴリズムが 1 ！論理装置 (PLU) から到着し u け入れられ、データを PLU に送信するために RLE が使用されます。

RUI API および SLI API の両方とも、以下のどちらかでデータ圧縮をハンドルすることができます。

- アプリケーションがデータの圧縮および圧縮解除を行う
- Communications Server がホストを使用してデータの圧縮および圧縮解除を行い、 s 圧縮データのアプリケーションへの送達およびアプリケーションからの u 領を行う。

セッションごとのデータ圧縮を折衝するための規則

セッションごとの RUI API および SLI API のデータ圧縮折衝のための , 則を以下に (します。

RUI 規則

1. RUI アプリケーションにデータの圧縮および圧縮解除をハンドルさせるには、以下のようにします。
   - RUI アプリケーションが BIND 要 a を u 信します。この BIND 要 a のバイト 25 のビット 6 とビット 7 は、圧縮が提案されている、または、圧縮が要 a されていることを (す設定になっています。
   - RUI アプリケーションは、バイト 25 のビット 6 とビット 7 が「提案されている、または指 (されている圧縮が u け入れられた」ことを (すように設定されている BIND 肯定 ~ 答を戻す必要があります。

2. Communications Server に RUI アプリケーション用に圧縮をハンドルさせるには、以下のようにします。
   - Communications Server SNA ノード構成ユーティリティーを使用して、このノードは以下をB行することにより圧縮をサポートする旨を指 (します。
     – 構成ノードを選択する
     – 拡張を選択する
     – ノードがサポートする最大圧縮レベルを RLE に設定する
   - RUI アプリケーションが BIND ~答をu信します。この BIND ~答のバイト 25 のビット 6 とビット 7 は、圧縮が提案されている、または、圧縮が要 a されていることを (す設定になっています。
   - RUI アプリケーションは、バイト 25 のビット 6 とビット 7 が「圧縮を行わない」ことを (すように設定されている BIND 肯定~答を戻します。Communications Server は BIND ~答を代行u信して修正し、!にホストに送るデータを圧縮し、圧縮解除します。

SLI 規則

1. SLI アプリケーションにデータの圧縮および圧縮解除をハンドルさせるには、以下のようにします。
   - SLI アプリケーションは、**SLI_OPEN** verb を使用する場合には、 BIND コールバック・ルーチンを用意する必要があります。
   - SLI アプリケーションの BIND コールバック・ルーチンが開始されると、 SLI は BIND 要 a をu信します。この BIND 要 a のバイト 25 のビット 6 とビット 7 は、圧縮が提案されている、または、圧縮が要 a されていることを (す設定になっています。
   - SLI アプリケーションは、バイト 25 のビット 6 とビット 7 が「提案されている、または指 (されている圧縮がu け入れられた」ことを (すように設定されている BIND ~答を戻す必要があります。
2. Communications Server に SLI 用に圧縮をハンドルさせるには、以下のようにします。
   - Communications Server SNA ノード構成ユーティリティーを使用して、このノードは、以下をB行することにより圧縮をサポートする旨を指 (します。
     – 構成ノードを選択する
     – 拡張を選択する
     – ノードがサポートする最大圧縮レベルを RLE に設定する
   - アプリケーションが **SLI_OPEN** verb に BIND コールバック・ルーチンを提供しなかった場合は、SLI はデフォルトに従って、 Communications Server が SLI 用にデータの圧縮および圧縮解除を行う旨を指 (するよう、BIND ~答を設定します。
   - アプリケーションが BIND コールバック・ルーチンを提供した場合は、以下のようになります。
     – BIND コールバック・ルーチンが開始されると、BIND コールバック・ルーチンは BIND 要 a をu信します。この BIND 要 a のバイト 25 のビット 6 とビット 7 は、圧縮が提案されている、または、圧縮が要 a されていることを (す設定になっています。

– SLI アプリケーションは、バイト 25 のビット 6 とビット 7 が「圧縮を行わない」ことを (すように設定されている BIND ~答を戻します。Communications Server は BIND ~答を代行 u 信して修正し、!にホストに送るデータを圧縮し、圧縮解除します。

セッション障害からの回復

エラーのために LUA セッションがクローズされてしまう 2 つのインスタンスがあります。

- LUA verb が 1 !戻りコード LUA_SESSION_FAILURE を出して完了する場合、または、
- RUI_INIT の正常な完了の後で、LUA verb が、 1 !戻りコード LUA_STATE_CHECK および 2 !戻りコード LUA_NO_RUI_SESSION を出して完了する場合。

セッションは再構築できることもあります。プログラムが LUA の回復を要 a する場合、LUA は回復を試みます。

ユーザーのプログラムが、エラーのためにクローズされた LUA セッションを u 信した場合は、プログラムは回復のためには以下の作業を行う必要があります。

- セッションのクローズを r ける。セッションはすでにクローズしています。
- セッションをオープンするために元々使用されていた verb を使って、セッションを再オープンする (RUI_INIT)。この verb がゼロ以外の 1 !戻りコードを出して完了する場合、セッションをこの時に再始動することはできません。
- 回復には時間がかかる場合があるので、回復の進行中に対話 0 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Wによって h まります。

第11章 LUA プログラムのB装

この章では、LUA プログラムのB装と作成のいくつかの局面について説明します。 この章では以下のトピックを説明しています。

- LUA サービスの呼び出しと順序付け
- LUA プログラムの作成
- s 同期完了とコールバック！能の使用
- 異なるプラットフォーム上でのコンパイルとリンク

Communications Server が行う LUA のインプリメンテーションは、 Microsoft** NT SNA サーバーとバイナリー互換性を持つように設W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 インターフェースおよび SLI インターフェースのインプリメンテーションと類似しています。

LUA プログラムの作成

LUA には、複数の RUI verb および SLI verb について 1 つの g 要 DLL が含まれています。 LUA アプリケーション・プログラムは、verb を / 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 制御ブロック内の選択したフィールドをセットし、 RUI または SLI を呼び出して、その verb 制御ブロックに対する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 a されるアクションを LUA に対して定A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 k 果を (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。

以下の表では、提供されたヘッダー・ファイルのソース・モジュールでの使用法と、 RUI プログラムをコンパイルしリンクするのに必要なライブラリーを (しています。

表 15. オペレーティング・システムの RUI API 用のヘッダー・ファイルおよびライブラリー

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

表 16. SLI API 用のヘッダー・ファイルおよびライブラリー

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	WINSLI32.LIB	WINSLI32.DLL
WIN3.1	WINSLI.H	WINSLI.LIB	WINSLI.DLL
OS/2	LUA_C	ACSSLI.LIB	ACSSLI.DLL

注: SLI API はサーバー上ではサポートされますが、Communications Server クライアントではサポートされません。

*WINNT = Windows NT
*WIN95 = Windows 95
*WIN3.1 = Windows 3.1

LUA サービスの呼び出し

プログラムが、指定されたエントリー・ポイントを呼び出し、単一のパラメーター（*verb record* と呼ばれるデータ構造のアドレス）を渡すことによって、LUA サービスを呼び出します。レコードには、特定のファンクションのための入力パラメーターが含まれています。 LUA は、操作の k 果として出される出力パラメーターを持つレコードを更新します。

verb レコード b 容について

異なる構造を持っていますが、3 つのタイプの verb レコードすべてに、以下のパラメーター用のフィールドがあります。

オペレーション
特定のオペレーションを B 行するよう指定する数。オペレーションの - 号名は、『_cons.h』 組み込みファイルの中で宣言されています。

verb レコード長
verb レコードのサイズ。これはオペレーションによって変わる場合があり、レコードを処理するために LUA で必要となります。

セッション ID
通信 verb およびサービス verb の中の、セッションまたはセッション名を 1 別する V 号。

1 次戻りコード
全 L 的な成功または : 敗を (す、LUA によって戻される V 号。

2 次戻りコード
特定の問題に関する障害を通知する、LUA によって戻される V 号。

相関係数
ユーザーのアプリケーションが verb レコードを他のデータに関連付けたり、 s 同期完了時に verb レコードを 1 別するために使用できる長整数。

通知ハンドル
verb が s 同期に完了するときに通知されるイベントのハンドル。

これは、Windows NT および Windows 95 ではイベント・ハンドル、Windows 3.1 ではウィンドウ・ハンドル、OS/2 では、セマフォー・ハンドルでなければなりません。

これらのフィールドのほとんどは同一のデータ ? を持ち、使用されるすべての verb レコード中で同一のオフセットの位置にあります。しかし、オペレーション・コードと verb レコード長フィールドはそれぞれ異なる特性を持っています。

多重プロセス

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。

マルチ・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 / 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 / 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 / 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 s 同期に完了します。 verb の同期完了とは、LUA の呼び出しの後で 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、 s 同期ポスト方 0 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 s 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 s 同

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h 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Uの）変換は、 311ページの『第17章 共通サービス verb (CSV)』で説明されている convert verb によって行われます。

第12章 RUI LUA エントリー・ポイント

この章では、LUA 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。

RUI DLL は、以下のプロシージャー・エントリー・ポイントを定Aします。

注: 本書第 2 部の諸章には、以下のシステムが提供する LUA API に関する情報が含まれています。

- Windows NT 上でB行される Communications Server
- 通信サーバー/NT 製品と一緒に納入される OS/2、Windows NT、Windows 95、および Windows 3.1 用の SNA API クライアント
- Windows 95 および Windows NT 用の パーソナル・コミュニケーションズ

これらのシステムが提供するサポート間で相違がある場合は、注 - します。

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、 s 同期通知が行われるかどうかを (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 s 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 a は同期して完了しています。 1 ！戻りコードおよび 2 ！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。

s 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 ！戻りコードおよび 2 ！戻りコードを調べて、エラー条 o の有無を確認してください。 207ページの『WinRUI』 も参照してください。

これは OS/2 クライアント用にサポートされる唯一の RUI エントリー・ポイントです。

WinRUI

すべての RUI verb について s 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 u け h 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。

このファンクションは RUI によって処理要 a が u け入れられたかどうかを (す値を戻します。 戻り値 0 は、要 a が u け入れられ処理されることを (します。 0 以外の値はエラーを (します。エラー・コードには ! 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、 s 同期通知が生じるかどうかを (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 s 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 a は同期して完了します。 1 ！戻りコードおよび 2 ！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 が完了すると、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 u け h ります。 **lParam** 引き数には、完了したと通知される VCB のアドレスが入ります。 *wParam* 引き数は未定 A です。 処理要 a が u け入れられる可能性がありますが（ファンクション呼び出しが 0 を戻した場合）、後で q] され、VCB 内に 1 ！戻りコードおよび 2 ！戻りコードがセットされることもあります。 1 ！戻りコードおよび 2 ！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。

注: 206ページの『RUI()』 も参照してください。

このエントリー・ポイントは、Communications Server Windows 3.1 クライアント用にはサポートされていません。

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 h り消します。

```
BOOL WINAPI WinRUICleanup (void);
```

戻り値は、登録 h り消しが成功したか : 敗したかを (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 h 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 h 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 / 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 / 行します。 212ページの『WinRUIStartup()』 も参照してください。

WinRUIGetLastInitStatus()

このファンクションは、アプリケーションが **RUI_INIT** の状況を = 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 j 段として使用できます。 この呼び出しは、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 = 別するために使用します。 詳細については、使用上の注意のセクション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 1 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**() の呼び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 ~ 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 u け h ります。 パラメーター wParam にはセッション状態が含まれています（戻り値を参照）。 *dwNotifyType* の値に ~ 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()** の呼び出しを / 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 a される指 (のタイプ。 これは、ウィンドウ・メッセージの **lParam** の内容と、 **WinRUIGetLastInitStatus()** が *hStatusHandle* をどのように解 a するかを h 定します。 使用できる値は ! のとおりです。

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 8 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 1 別子が入ります。

bClearPrevious
TRUE の場合は、 *dwSid* により 1 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
パーソナル・コミュニケーションズ または Communications Server のどちらも B 行されません。

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 u 信されていません。

WLUAPUACTIVE
ACTPU が u 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 u 信されましたが、 **ACTLU** が u 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 1 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）。

このファンクションは、ウィンドウ・ハンドルかイベント・ハンドルとともに使用して、状況変更の s 同期通知ができるようにするためのものですが、単独で使用して、セッションの現在の状況を検出することもできます。

この拡張！能をウィンドウ・ハンドルと共に使用するには、！の 2 通りのいずれかでできます。

```
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 1 別子が入ります。

ファンクションがウィンドウ・ハンドルと共に使用された場合、状況の報告を h り消すには以下のコマンドを使います。

```
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。

このファンクションをイベント・ハンドルと共に使用するには、！のようにB行してください。

```
WinRUIGetLastInitStatus(Sid,Handle,WLUA_NOTIFY_EVENT,FALSE);
```

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！の呼び出しを / 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。

ファンクションがイベント・ハンドルと共に使用された場合、状況の報告を h り消すには以下のコマンドを使ってください。

```
WinRUIGetLastInitStatus(Sid,NULL,0,TRUE);
```

このファンクションを使ってセッションの現在の状況を照会する場合には、イベント・ハンドルまたはウィンドウ・ハンドルを使用する必要はありません。 代わりに！のコマンドを使用します。

```
Status = WinRUIGetLastInitStatus(Sid,NULL,0,0,FALSE);
```

注: WinRUIGetLastInitStatus は Communications Server SNA API クライアントではサポートされません。

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 h り出すことができます。

```
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V号（改訂V号）を (し、低位バイトはバージョンV号を (します。

luadata

RUI インプリメンテーション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V号をサポートできるかどうかを (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！のいずれかです。

WLUAVERNOTSUPPORTED

要 a 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 a したバージョンを = 別できませんでした。

この呼び出しは、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。208ページの『WinRUICleanup()』 も参照してください。

このエントリー・ポイントは、Communications Server Windows 3.1 クライアント用にはサポートされていません。

GetLuaReturnCode()

VCB 内の 1 ！戻りコードおよび 2 ！戻りコードを、印刷可能ストリングに変換します。このファンクションは、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（バイト単位）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？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。このファンクションは、指定した verb 制御ブロックからの読み h り、または指定したバッファーへの書き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**）で終わりません。

呼び出し例

！の例は、**WINRUI32.DLL** を呼び出す方法を (しています。この DLL のヘッダー・ファイルは **WINLUA.H** です。この例では **RUI DLL** を呼び出して、プログラムから **RUI** verb を / 行します。

```
#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 */
. . .
}
```

第13章 RUI verb

この章には！の情報を } 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！の情報を (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 (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！のものから成っています。

- **lua_common**。これはすべての verb で使用されます（215ページの『共通 verb ヘッダー』で説明します）。
- **specific**。これは **RUI_BID** verb のみに使用されます（220ページの『RUI_BID データ構造』で説明します）。

構造は！のように定 A 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

パーソナル・コミュニケーションズ LUA は、F 用の*共通 verb* ヘッダー を使って、すべての着信および / 信データを転送します。 verb 制御ブロック内のフィールドは！のように定 A 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;
```

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、！の値のどれか 1 つ、または、いくつかを OR で k んだものです。

LUA_FID
A 0 1 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 委託標 1

LUA_EFI
^ 送フロー標 1

lua_th.daf
DAF（宛先アドレス・フィールド）

lua_th.oaf
OAF（ / 点アドレス・フィールド）

lua_th.snf
順序 V 号フィールド

lua_rh 送 u 信したメッセージの要 a / ~ 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、！の値のどれか 1 つか、または、いくつかを OR で k んだものです。

LUA_RRI
要 a ~ 答標 1

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
A 0 標 1

LUA_SDI
センス・データ組み込み標 1

LUA_BCI
チェーン開始標 1

LUA_ECI
チェーン終了標 1

LUA_DR1I
確定 ~ 答 1 標 1

LUA_DR2I
確定 ~ 答 2 標 1

LUA_RI
例外時 ~ 答標 1 （要 a の場合）、または ~ 答タイプ標 1 （~ 答の場合）

LUA_QRI
待ち行列 ~ 答標 1

LUA_PI
ペーシング標 1

LUA_BBI
ブラケット開始標 1

LUA_EBI
ブラケット終了標 1

LUA_CDI
方向変換標 1

LUA_CSI
コード選択標 1

LUA_EDI
暗号化データ標 1

LUA_PDI
埋め込みデータ標 1

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、！の値のどれか 1 つ、またはいくつかを OR で k んだものです。

LUA_BID_ENABLE
送信権要 a の使用可能標 1

LUA_NOWAIT
データ待！なしフラグ

LUA_SSCP_EXP
SSCP ^ 送フロー

LUA_SSCP_NORM
SSCP 通常フロー

LUA_LU_EXP
LU ^ 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 u 信した（または **RUI_BID** verb に対して指 (された）SNA メッセージのタイプ。これは！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、！の値のどれか 1 つ、またはいくつかを OR で k んだものです。

LUA_BID_ENABLE

送信権要 a の使用可能標 1

LUA_ASYNC

s 同期 verb 完了フラグ

LUA_SSCP_EXP

SSCP ^ 送フロー

LUA_SSCP_NORM

SSCP 通常フロー

LUA_LU_EXP
LU ^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

以下のパラメーターは、 **RUI_BID** verb に固有のものであり、この verb にのみ提供されるものです。

lua_peek_data
読み h りを待っている最大 12 バイトのデータ。

RUI_BID

RUI_BID verb は、 u 信されたメッセージが読み h 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 / 行する前に、どのようなデータが使用可能かを = 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、 u 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 (します。 **RUI_BID** と **RUI_READ** の g 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 n 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 n してしまうので、アプリケーションは、いったん読み h 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！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！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 JISCII 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、& 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、 s 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

！のパラメーターは常に戻されます。

lua_flag2
これは、verb が s 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。！の項を参照してください。

verb が正常に完了した場合は、！のパラメーターが戻されます。

lua_prim_rc
LUA_OK

lua_sid
この verb を / 行するときに、アプリケーションでは、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
u 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 u 信したメッセージの伝送ヘッダー (TH) からの情報。

lua_rh u 信したメッセージの要 a / ~ 答ヘッダー (RH) からの情報。

lua_message_type
u 信したメッセージのメッセージ・タイプ。これは！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

！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 u 信されたのかを (します。

LUA_SSCP_EXP

SSCP ^ 送フロー

LUA_LU_EXP

LU ^ 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)。

！の戻りコードは、他の verb によって h 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 (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 / 行されました。

！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 (します。

lua_prim_rc

LUA_PARAMETER_CHECK

lua_sec_rc

可能な値は！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 q] 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。

！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 / 行されたことを (します。

lua_prim_rc
LUA_STATE_CHECK

lua_sec_rc
LUA_NO_RUI_SESSION

RUI_INIT verb は、
このセッション

lua_sec_rc

可能な値は！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：敗しました。

LUA_RUI_LOGIC_ERROR

この戻りコードは！のいずれかを (します。

- ホスト・システムが SNA プロトコルに違？した。
- LUA で内部エラーが検出された。

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B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 / 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。

注 a

この verb を / 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 / 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 / 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 - 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 a 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！状態にあることを (したら、アプリケーションは、 **RUI_READ** verb を / 行してそのデータを u 信する必要があります。 **RUI_READ** verb を / 行することにより、送信権要 a されたメッセージが u 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L 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 (すだけで、待！しているメッセージのデータの合W長を (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、 u 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 u け入れる **RUI_READ** verb でのデータ長が、メッセージを } 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！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
セッションを開始したいローカル LU または LU プールの JISCII の名前。これは、構成された LUA LU 名または LU プール名と一致する必要があります。パーソナル・コミュニケーションズおよび Communications Server上のアプリケーションについては、名前は以下のように使用されます。

名前が、プールにない LU の名前である場合、パーソナル・コミュニケーションズおよび Communications Serverはこの LU を使ってセッションを開始しようとします。

名前が LU プールの名前、またはプール内にある LU の名前である場合、パーソナル・コミュニケーションズおよび Communications Serverはプールから使用可能な最初の LU を使って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 JISCII 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 クライアント上のアプリケーションの場合、名前は構成された LUA セッション名と一致する必要があります。

以下の情報は、Communications Server Windows 95 および Windows NT SNA API クライアントに対してのみ適用されます。

それぞれのユーザーについてのデフォルトの LUA セッション名は、該当する構成ユーティリティー、すなわち INI 構成または LDAP のどちらか、を使用して割り当てることができます。

3270 のような LUA プログラムは、デフォルトの LUA セッション名を直接指定するのではなく、選択して使用することができます。LUA プログラム

が、 **lua_name** フィールドが 2 進ゼロまたは ASCII のブランクに設定されている **RUI_INIT** verb を / 行すると、 RUI API は構成済みのデフォルト LUA セッション名を使用します。

lua_post_handle
これは、 s 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、パーソナル・コミュニケーションズおよび Communications Server から RUI_INIT_STATUS 指 (をuけh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、236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。パーソナル・コミュニケーションズおよび Communications Server は！の 2 つの値をu 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は、
アプリケーション・プログラムによってB行されます。

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Aの暗号化および暗号解読ルーチンを (す 1～127のO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、パーソナル・コミュニケーションズおよび Communications Serverではサポートされていません)。

戻りパラメーター

！のパラメーターは常に戻されます。

lua_flag2
これは、verb が s 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 などのエラーを戻す場合以外は、 s 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。！の項を参照してください。

verb が正常にB行された場合は、LUA は！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 1 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。

！の戻りコードは、他の verb によって h 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 (します。

lua_prim_rc
LUA_CANCELLED

lua_sec_rc
LUA_TERMINATED

RUI_INIT が完了する前に、
RUI_TERM verb が出されました。

！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 (します。

lua_prim_rc
LUA_PARAMETER_CHECK

lua_sec_rc
可能な値は！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。 LU 名または LU プール名が *Personal Communications and Communications Server System Management Programming* API で定 A されているかどうかを調べてください。

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。

！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 / 行されたことを (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。

！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 (します。

lua_prim_rc
LUA_UNSUCCESSFUL

lua_sec_rc
可能な値は！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。

！に (す **lua_sec_rc** の値はパーソナル・コミュニケーションズおよび 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？ G します)。

X10020000
ACTPU が u 信されていません。 **RUI_INIT** は PU を活動化しません。

X10100000
ACTPU が u 信されていません。 **RUI_INIT** は PU を活動化します。

X10110000
ACTPU が u 信されました。 **ACTLU** は u 信されていません。 SSCP は、 + 己定 A 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 u 信されました。 **ACTLU** は u 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。

！の 1 ！戻りコードおよび 2 ！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 (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：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B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 / 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。

注 a

この verb は、セッションで / 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 / 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 / 行するその他のすべての verb は、！に (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 1 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u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！することになります。 **RUI_INIT** verb より前に **ACTLU** がu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可視ではありません。

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 / 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 h 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！状態になることがあります。 **RUI_PURGE** は、このような待！中の verb の制御を強制的に戻らせます（1 ！戻りコードは CANCELLED)。

指定パラメーター

アプリケーションは！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 JISCII 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、& 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 n する **RUI_READ** LUA_VERB_RECORD を指すポインター。

lua_post_handle
これは、 s 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

！のパラメーターは常に戻されます。

lua_flag2
これは、verb が s 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。！の項を参照してください。

verb が正常に完了した場合は、！のパラメーターが戻されます。

lua_prim_rc
LUA_OK

lua_sid
この verb を / 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。

！の戻りコードは、他の verb によって h 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 (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 / 行されました。

！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 (します。

lua_prim_rc
LUA_PARAMETER_CHECK

lua_sec_rc
可能な値は！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。

！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 / 行されたことを (します。

lua_prim_rc
LUA_STATE_CHECK

lua_sec_rc
可能な値は！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 / 生しました。

！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 (します。

lua_prim_rc
LUA_UNSUCCESSFUL

lua_sec_rc
可能な値は！のとおりです。

LUA_INVALID_PROCESS
この verb を / 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 / 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 / 行される前に **RUI_READ** verb が完了しました。

！の 1 ！戻りコードおよび 2 ！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 (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！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：敗しました。

LUA_RUI_LOGIC_ERROR
この戻りコードは！のいずれかを (します。

- ホスト・システムが SNA プロトコルに違？した。
- LUA で内部エラーが検出された。

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B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 / 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。

注 a

この verb が使用できるのは、**RUI_READ** が / 行されていて、その完了が保留状態にある（つまり 1 ！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B行中である場合は、この verb は / 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 (を / 行することはできません。 パーソナル・コミュニケーションズは、 **RUI_INIT** 処理時にこの指 (をアプリケーションに送信し、 LU-SSCP セッションの状態についての情報を提供します。 **RUI_INIT_STATUS** が送られるのは、アプリケーションが、 **RUI_INIT** を / 行するときに状況情報を要 a した場合だけです（227ページの『RUI_INIT』を参照）。

指定パラメーター

RUI_INIT_STATUS では ! 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（パーソナル・コミュニケーションズおよび 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。可能な値は ! 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 u 信されていないか、または **DACTPU** が u 信されました。

LUA_PU_ACTIVE
SSCP から **ACTPU** が u 信されました。

LUA_PU_REACTIVATED
PU が活動状態にあるときに **ACTPU(COLD)** が u 信されました。

LUA_LU_INACTIVE
ACTLU が q] されたか、または **DACTLU** が u 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
パーソナル・コミュニケーションズおよび 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、 s 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。 パーソナル・コミュニケーションズおよび 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 u け h ります。 データを読み h りたい特定のメッセージ・フロー（LU 通常、LU ^ 送、SSCP 通常、または SSCP ^ 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 / 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 ! 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 JISCII 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、& 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 u 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 u 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、 s 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、！の値のどれか 1 つ、またはいくつかを OR で k んだものです。

- 読み h り可能なデータの有無に関 8 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 / 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 h るかを指 (するために、！の 1 つまたは複数のフラグをセットします。

  LUA_SSCP_EXP
  SSCP ^ 送フロー

  LUA_LU_EXP
  LU ^ 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！のとおりです。

  1. SSCP ^ 送
  2. LU ^ 送
  3. SSCP 通常
  4. LU 通常

  どのフローからデータが読み h られたかを (すために、**lua_flag2** の中で同じフラグがセットされます（238ページの『戻りパラメーター』を参照）。

戻りパラメーター

！のパラメーターは常に戻されます。

lua_flag2
verb が s 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。！の項を参照してください。

verb が正常にB行された場合は、LUA は！のパラメーターも戻します。

lua_prim_rc
LUA_OK

！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、提供された **lua_data_length** パラメーターが小さすぎたために、切りNてが / 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 / 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
u 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 u 信したメッセージの伝送ヘッダー (TH) からの情報。

lua_rh u 信したメッセージの要 a / ~ 答ヘッダー (RH) からの情報。

lua_message_type
u 信したメッセージのメッセージ・タイプ。これは！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 パラメーター

これはどのメッセージ・フローからデータが読み h られたかを (すもので、！のいずれかの値にセットされます。

LUA_SSCP_EXP
SSCP ^ 送フロー

LUA_LU_EXP
LU ^ 送フロー

LUA_SSCP_NORM
SSCP 通常フロー

LUA_LU_NORM
LU 通常フロー

！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 h り消されたため、この verb が正常に完了しなかったことを (します。

lua_prim_rc
LUA_CANCELLED

lua_sec_rc
可能な値は！のとおりです。

LUA_PURGED
RUI_PURGE verb により **RUI_READ** verb が h 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 / 行されました。

！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 (します。

lua_prim_rc
LUA_PARAMETER_CHECK

lua_sec_rc
可能な値は！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！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 h るかを指 (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でした。(詳細は、243ページの『注 a 』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。

！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 / 行されたことを (します。

lua_prim_rc

LUA_STATE_CHECK

lua_sec_rc

LUA_NO_RUI_SESSION

RUI_INIT verb は、
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 セッション障害が / 生しました。

！の 1 ！戻りコードは、！の 2 つのケースのどちらかを (します。どちらが該当するかは 2 ！戻りコードに (されます。

- パーソナル・コミュニケーションズおよび Communications Serverがホストから u 信したデータにエラーを検出しました。パーソナル・コミュニケーションズおよび Communications Serverは、 u 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（そのメッセージがチェーンの一部である場合はチェーンの残りの部分）を破棄し、ホストに] 定 ~ 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、] 定 ~ 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、チェーンの途中にあるメッセージに対して] 定 ~ 答を送りました。パーソナル・コミュニケーションズは、この連鎖にある後続のメッセージを消 n し、連鎖のすべてのメッセージを u 信して消 n 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 ！戻りコードには、] 定 ~ 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、パーソナル・コミュニケーションズがホス

ト・データ中にエラーを検出して、ホストに] 定 ~ 答を送信したことを (しています。戻されるセンス・コード値の解 a の仕方は、 175ページの『SNA 層』を参照してください。

0 の 2 ！戻りコードは、チェーンの途中のメッセージに対する] 定 ~ 答の **RUI_WRITE** の後で、パーソナル・コミュニケーションズがこのチェーンからすべてのメッセージを u け h り、そして破棄したことを (します。

！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 (します。

lua_prim_rc

LUA_UNSUCCESSFUL

lua_sec_rc

可能な値は！のとおりです。

LUA_DATA_TRUNCATED

lua_data_length パラメーターが、 u 信したメッセージ・データの B 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 ！戻りコードを h 得した場合、追加のパラメーターも戻されます。

LUA_NO_DATA

データを待たず即時に戻すことを指 (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 / 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 / 行したインスタンスと同じではありません。

！の 1 ！戻りコードおよび 2 ！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 (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！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 : 敗しました。

LUA_RUI_LOGIC_ERROR

この戻りコードは！のいずれかを (します。

- ホスト・システムが SNA プロトコルに違 ? した。
- LUA で内部エラーが検出された。

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B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 / 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。

注 a

この verb を / 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 / 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、！のすべての条 o 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 / 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 - 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

u 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Nてられます。データの **lua_max_length** バイトだけが戻されます。 1 ！および 2 ！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 を使ってメッセージを読み h ると、それは u 信メッセージ待ち行列から除 n され、再びアクセスすることはできません。

注: RUI_BID verb は、データを待ち行列から除 n せずに読み h 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 ！から 1 ！のハーフ・セッションでペーシングを使って（ホスト構成で指定される）、パーソナル・コミュニケーションズおよび Communications Serverノードがメッセージでいっぱいにならないようにすることができます。 LUA アプリケーションのメッセージ読み h りの速度が低下すると、パーソナル・コミュニケーションズおよび Communications Server は、ホストへのペーシング ~ 答の速度を低下させるために、

RUI_READ

ペーシング ~ 答の送信を遅 d 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！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）。 これを (struct LUA_COMMON) のサイズにセットします。

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、& 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、 s 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

！のパラメーターは常に戻されます。

lua_flag2
これは、verb が s 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。！の項を参照してください。

verb が正常にB行された場合は、LUA は！のパラメーターも戻します。

lua_prim_rc
LUA_OK

以下に (す戻りコードは、指定パラメーターのどれかにエラーが生じたために verb が正常に完了しなかったことを (します。

lua_prim_rc
LUA_PARAMETER_CHECK

lua_sec_rc
可能な値は！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。

！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 / 行されたことを (します。

lua_prim_rc
LUA_STATE_CHECK

lua_sec_rc
LUA_NO_RUI_SESSION

RUI_INIT verb は、
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
セッション障害が / 生しました。

！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 (します。

lua_prim_rc
LUA_UNSUCCESSFUL

lua_sec_rc
可能な値は！のとおりです。

LUA_COMMAND_COUNT_ERROR
この verb を / 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 / 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 / 行したインスタンスと同じではありません。

！の 1 ！戻りコードおよび 2 ！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 (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！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：敗しました。

LUA_RUI_LOGIC_ERROR

この戻りコードは！のいずれかを (します。

- ホスト・システムが SNA プロトコルに違？した。
- LUA で内部エラーが検出された。

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B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 / 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。

注 a

この verb は、**RUI_INIT** verb を / 行した後であれば、それが完了しているかどうかに関 8 なく、いつでも / 行できます。 **RUI_TERM** を / 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 ！戻りコード LUA_CANCELLED を < 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 / 行できなくなります。

RUI_WRITE

RUI_WRITE verb は、SNA 要 a 単位または ~ 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 ! 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、& 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 u 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 ~ 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 V 号に基づいて ~ 答を完了します（**lua_th.snf** を参照）。 **BIND** または **STSN** に対する肯定 ~ 答の場合は、拡張 ~ 答が v されるので、0 以外の値を使用できます。

] 定 ~ 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 a の場合、またはデータを必要とする肯定 ~ 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。

] 定 ~ 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、 s 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
~ 答の送信の場合に限り必要です。これは、この ~ 答の対象となっている要 a の順序 V 号です。

lua_rh 要 a 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 a ヘッダー）に対 ~ 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。

~ 答を送信するときは、！の 2 つの **lua_rh** フラグだけがセットされます。

LUA_RRI
~ 答を (すためにセットされます。

LUA_RI
肯定 ~ 答の場合はセットされず、] 定 ~ 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 (すために、！のいずれかのフラグをセットします。

LUA_LU_EXP
LU ^ 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: パーソナル・コミュニケーションズおよび Communications Serverでは、アプリケーションが SSCP ^ 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

！のパラメーターは常に戻されます。

lua_flag2
これは、verb が s 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。！の項を参照してください。

verb が正常にB行された場合は、LUA は！のパラメーターも戻します。

lua_prim_rc
LUA_OK

lua_sid
この verb を / 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 - 入されたフィールドも含みます。ホストからの ~ 答との対 ~ 付けのために、**lua_th.snf**（順序V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 - 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 u 信されたかを (すために、！のいずれかの値にセットされます。

LUA_SSCP_EXP
SSCP ^ 送フロー

LUA_LU_EXP
LU ^ 送フロー

LUA_SSCP_NORM
SSCP 通常フロー

LUA_LU_NORM
LU 通常フロー

！の戻りコードは、他の verb によって h 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 (します。

lua_prim_rc
LUA_CANCELLED

lua_sec_rc
LUA_TERMINATED

RUI_TERM verb がこのセッションで出されなかったため、
verb は h り消されました。

！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 (します。

lua_prim_rc
LUA_PARAMETER_CHECK

lua_sec_rc
可能な値は！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 ^ 送フローで送信することを (します。パーソナル・コミュニケーションズおよび 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 (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 ! のいずれかの場合を (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。
- ~ 答を送信するために **RUI_WRITE** verb が使用されましたが、~ 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。

! 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 / 行されたことを (します。

lua_prim_rc
LUA_STATE_CHECK

lua_sec_rc
可能な値は ! 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 / 生しました。

！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 (します。

lua_prim_rc
LUA_UNSUCCESSFUL

lua_sec_rc
可能な値は！のとおりです。

LUA_FUNCTION_NOT_SUPPORTED
この戻りコードは！のいずれかの場合を (します。

- **lua_rh** が LUA_FI（フォーマット標 1）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 1 される要 a コードではありませんでした。
- **lua_rh** が LUA_RH_NC に送信されました（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た）。パーソナル・コミュニケーションズは、アプリケーションがこのカテゴリーで要 a を送信するのを v 可しません。

LUA_INVALID_PROCESS
この verb を / 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 / 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 u 信した **BIND** メッセージに対する肯定 ~ 答を送信しました。しかし、パーソナル・コミュニケーションズおよび Communications Server・ノードは指定された **BIND** パラメーターを u け入れることができず、] 定 ~ 答をホストへ送りました。パーソナル・コミュニケーションズおよび Communications Serverが u け入れる **BIND** プロファイルの詳細は、175 ページの『SNA 層』を参照してください。

LUA_RSP_CORRELATION_ERROR
RUI_WRITE を使用して ~ 答を送信するときに、 **lua_th.snf** パラメーター（これは ~ 答の対象となっている u 信メッセージの順序 V 号を (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 u 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 ！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 (す SNA センス・コードです。戻される SNA センス・コードの解 a の仕方については、175ページの『SNA 層』を参照してください。

！の 1 ！戻りコードおよび 2 ！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 (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！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：敗しました。

LUA_RUI_LOGIC_ERROR

この戻りコードは！のいずれかを (します。ホスト・システムが SNA プロトコルに違？した。 LUA で内部エラーが検出された。

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** パラメーターが無効でした。verb はB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 / 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。

注 a

この verb を / 行する前に、**RUI_INIT** verb が正常に / 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 / 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 / 行できます。 LU ^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u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i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 (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 u け入れたことを (すものではありません。 2 ！から 1 ！ハーフ・セッション上でペーシングを使用して（これは **BIND** 上で指定される）、LUA アプリケーションが、ローカルまたはリモート LU が、ハンドルできないほどのデータを送信することのないようにすることができます。その場合は、LUA は LU 通常フローでの **RUI_WRITE** を遅 d 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: パーソナル・コミュニケーションズおよび Communications Serverでは、アプリケーションが SSCP ^ 送フロー上でデータを送ること (LUA_SSCP_EXP) はできません。

第14章 SLI エントリー・ポイント

この章では、SLI 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。

SLI DLL は、以下のプロシージャー・エントリー・ポイントを定Aします。

SLI()

すべての **SLI** verb についてイベント通知を提供します。

構文

```
void WINAPI SLI (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。

戻り値

lua_flag2.async に戻される値は、 s 同期通知が生じるかどうかを (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、イベント・シグナルにより s 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 a は同期して完了します。 1 ！戻りコードおよび 2 ！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

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。

s 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 ！戻りコードおよび 2 ！戻りコードを調べて、エラー条 o の有無を確認してください。 257ページの『WinSLI』 も参照してください。

OS/2 および Windows 3.1 クライアントには SLI verb のみが提供されます。

WinSLI

すべての SLI verb について s 同期メッセージ通知を提供します。

構文

```
int WINAPI WinSLI (HWND hWnd,
                   LUA_VERB_RECORD* vcb);
```

パラメーター

説明

hwnd 完了メッセージを u け h 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

このファンクションは、SLI によって処理要 a が u け入れられたかどうかを (す値を戻します。 戻り値 0 は、要 a が u け入れられ処理されることを (します。 0 以外の値はエラーを (します。エラー・コードには ! 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、 s 同期通知が生じるかどうかを (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 s 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 a は同期して完了します。 1 ！戻りコードおよび 2 ！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

verb が完了すると、アプリケーションのウィンドウ *hWind* は、 **WinSLI** を入力ストリングとする **RegisterWindowMessage** から戻されたメッセージを u け h ります。 **lParam** 引き数には、完了したと通知される VCB のアドレスが入ります。 *wParam* 引き数は未定 A です。 処理要 a が u け入れられる可能性がありますが（ファンクション呼び出しが 0 を戻した場合）、後で q] され、VCB 内に 1 ！戻りコードおよび 2 ！戻りコードがセットされることもあります。 1 ！戻りコードおよび 2 ！戻りコードを調べて、エラーの有無を確認してください。

注: 256ページの『SLI()』 も参照してください。

WinSLICleanup()

アプリケーションを終了し SLI API からアプリケーションの登録を h り消します。

構文

```
BOOL WINAPI WinSLICleanup (void);
```

戻り値

戻り値は、登録 h り消しが成功したか : 敗したかを (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 h 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。

使用上の注意

WinSLICleanup は、SLI API の登録を h 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。

WinSLICleanup の使用は必須ではありません。

WinSLIStartup()

アプリケーションで、必要な SLI API のバージョンを指定し、 API の詳細情報を h り出すことができます。

構文

```
int WINAPI WinSLIStartup (WORD wVersionRequired,
                          LUADATA* luadata);
```

パラメーター

説明

wVersionRequired

必要な SLI 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Ｖ号（改訂Ｖ号）を (し、低位バイトはバージョンＶ号を (します。

luadata

SLI インプリメンテーションのバージョンを戻します。

戻り値

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、SLI API が指定のバージョンＶ号をサポートできるかどうかを (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！のいずれかです。

WLUAVERNOTSUPPORTED

要 a した SLI API サポートのバージョンは、この特定 SLI API では提供されていません。

WLUAINVALID

要 a したバージョンを = 別できませんでした。

使用上の注意

WinSLIStartup の使用は必須ではありません。

第15章 SLI verb

この章には、それぞれの SLI verb に関する以下の情報を } めてあります。

- verb の目的。
- SLI への指定パラメーターと SLI からの戻りパラメーター。 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 (されているパラメーターは、常に 0 にセットされます。

SLI_BID

この verb は SLI アプリケーション・プログラムに、メッセージが保留状態で SLI_RECEIVE による読み h りを待 ! していること、または、保留状態が表 (されていることを通知します。 SLI_BID は、保留状態のデータを事前表 (し、それによりアプリケーションがデータを u 信するための、ストラテジーを組み立てることができるようにするために使用します。 SLI アプリケーション・プログラム用のデータまたは状況が着信すると、適格な SLI_RECEIVE が s 活動状態にある場合には、SLI_BID にはそのことが通知されます。セッションが正常にオープンすると (または開始タイプが SSCP アクセスで始まる場合は SLI_OPEN 時に)、アプリケーション・プログラムは SLI_BID verb を / 行して、アプリケーション・プログラムがビッド・メカニズムを使用することを (します。

指定パラメーター

アプリケーションは ! のパラメーターを提供します。

lua_verb
LUA_VERB_SLI

LUA verb 用の verb コード標 1 。

lua_verb_length
verb 制御ブロックの長さ。この数値は、SLI_BID verb 用の SLI により予期される長さと、等しくなければなりません。

lua_opcode
LUA_OPCODE_SLI_BID

verb の操作コード。

lua_correlator
verb と他のユーザー提供の情報とをリンクさせる値。LUA インターフェースは、このパラメーターを使用しません。

lua_luname
ASCII 表 - のローカル LU 名。名前が 8 文字未満であれば、ブランク文字を埋め込んで補う必要があります。 LUA は、 **lua_sid** が 0 の場合にのみこのパラメーターを検査します。すべての verb に **lua_luname** パラメーターを使用すると、デバッグが一層簡単になり、特に複数の LU 構成の場合に効果的です。

lua_sid
使用すべきセッションを 1 別する SLI_OPEN が戻すセッション ID。このパラメーターが 0 である場合は、**lua_luname** パラメーターは 1 別用に使用されます。

lua_post_handle
これは、 s 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

verb が正常に完了した場合は、 ! のパラメーターが戻されます。

lua_prim_rc
verb ！能が設定する1 ！戻りコード。

lua_sec_rc
verb ！能が設定する2 ！戻りコード。

lua_data_length
u 信したピーク・データの長さ。

lua_peek_data
このパラメーターには、読み込まれる RU データの先頭から最大 12 バイトまでが入っています。このパラメーターに戻されるデータの長さは、**lua_data_length** パラメーターに入っています。

lua_th メッセージ用の SNA 伝送ヘッダー (TH) を } 容する 6 バイトのパラメーター。

lua_rh メッセージ用の SNA 要 a / ~ 答ヘッダー (RH) を } 容する 3 バイトのパラメーター。

lua_message_type
SNA データおよびコマンドのタイプ。有効なメッセージ・タイプは以下のとおりです。

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_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_SIGNAL
LUA_MESSAGE_TYPE_STSN

SLI は LUA インターフェース拡張ルーチンを使用して BIND および STSN 要 a を u 信し、~ 答します。

LU_DATA、LUSTAT_LU、LUSTAT_SSCP、および SSCP_DATA は SNA コマンドではありません。

lua_flag2
出力パラメーターとして使用するビットを } 容する 1 バイトのフラグ。 verb

の完了時には、値を - 述されていないすべてのビットは予約済みとなり、0 に設定される必要があります。高位のハーフバイトに入るフラグを以下に (します。

lua_flag2.async
この verb の s 同期完了を (すフラグ

下位のハーフバイトには、メッセージ・セッションおよび流れを - 述するフラグが } 容されます。以下のフラグの 1 つが戻されます。

lua_flag2.sscp_exp
SSCP ^ 送フローを指定します

lua_flag2.sscp_norm
SSCP 通常フローを指定します

lua_flag2.lu_exp
LU ^ 送フローを指定します

lua_flag2.lu_norm
LU 通常フローを指定します

lua_prim_rc
verb ！能が設定する 1 ！戻りコード。 詳細については、 369ページの『付録B. LUA Verb 戻りコード』 を参照してください。

lua_sec_rc
verb ！能が設定する 2 ！戻りコード。 詳細については、 369ページの『付録B. LUA Verb 戻りコード』 を参照してください。

使用上の注意

それぞれのセッションごとに、ただ 1 つの **SLI_BID** だけが、アクティブになることができます。 **SLI_BID** が再活動化されている場合は、アプリケーション・プログラムは、データが読み込まれていなくてもそれぞれのフローごとに 1 回ビッドを行うことができます。アプリケーション・プログラムがビッド・データを読み込まない場合、特定のフローについてのビッドは再度行われません。

SLI_BID verb を

- SSCP ^送
- LU ^送
- SSCP 通常
- LU 通常

2. **SLI_BID** の完了に引き続き、 LUA アプリケーションが複数の **lua_flag1** フロー・フラグが設定されている **SLI_RECEIVE** を / 行する場合は、データの読み h りは **SLI_BID** が戻すデータのフローとは異なったものになる場合があります。この事態は、より高い優先順位を持つデータが、 **SLI_BID** の完了時点と **SLI_RECEIVE** が / 行された時点との間に到着した場合に / こります。

   しかし、LUA アプリケーションは **SLI_RECEIVE** が、ビッドを u けたばかりのデータを読み h るのを保証することができます。この保証は、**SLI_RECEIVE** verb 用の制御ブロック中に、 **lua_flag1** フロー・フラグの中から 1 つだけを設定し、完了した **SLI_BID** の **lua_flag2** フィールドに戻されたのと同一のフローを指定することにより行われます。

SLI_BID は、RU が到着すると直ちに完了します。この RU はチェーン中の RU でも、また、複数 RU チェーンの先頭の RU でも構いません。 **SLI_BID** の完了時には、単一エレメント・チェーンは、完了チェーンがアプリケーションにビッドを行う唯一の時間です。

SLI_BID が複数 RU チェーンの先頭の RU で完了し、以降の **SLI_RECEIVE** には **lua_flag1.nowait** オプションが指定されている場合は、 **lua_flag1.nowait** オプションは無視されます。 **SLI_RECEIVE** verb は進行中に戻され、verb は、チェーン中のすべての RU の到着後に s 同期に完了します。

状況が選択可能であれば、アプリケーションはその状況を読み h る必要があります。アプリケーションが、 **SLI_BID** または **SLI_RECEIVE** を / 行して状況を読み h るまで、他のすべての操作はリジェクトされますが、以下については例外です。

- SSCP フロー上の **SLI_SEND** verb
- **SLI_CLOSE**

1 ！戻りコードが STATUS である場合は、戻される **SLI_BID** パラメーターは **lua_prim_rc**、**lua_sec_rc**、 および **lua_sid** です。状況が選択可能になった時に**SLI_BID** および **SLI_RECEIVE** が両方ともアクティブである場合は、**SLI_BID** のみが状況と共に通知されます。状況に関してアプリケーション・プログラムがビッドされると、すべての情報が表 (され、**SLI_RECEIVE** は不要になります。

1 ！戻りコードの値が STATUS である場合は、可能な 2 ！戻りコードの値は以下のとおりです。

- READY

  SLI セッションが、現在すべての追加コマンドの処理用に作動可能であることを (します。前に出された NOT_READY 状況の u 信後に、READY 状況が / 行されます。

- NOT_READY

  タイプ値 X'02' または X'01' を持つ CLEAR コマンドまたは UNBIND コマンドをホストから u 信したことを (します。 SLI セッションは中断状態になります。

- CLEAR が到着すると、セッションは SDT コマンドを u 信するまで中断状態となります。
- SNA UNBIND タイプ X'02' (BIND が予定されている UNBIND) が到着すると、セッションは、BIND、オプションの CRV と STSN、および SDT コマンドを u 信するまで中断状態になります。すべてのユーザー拡張ルーチンは、再入可能になっている必要があります。
- UNBIND タイプ X'01' (通常の UNBIND) が到着し、かつ、このセッション用の **SLI_OPEN** verb で LUA_SESSION_TYPE_DEDICATED の **lua_session_type** を指定していた場合は、セッションは BIND、オプションの CRV と STSN、および SDT コマンドを u 信するまで中断状態になります。これらのコマンドをプロセスするために提供されるユーザー拡張ルーチンは、再入可能となっている必要があります。

  CLEAR、UNBIND タイプ X'02'、または UNBIND タイプ X'01' の到着後はアプリケーションは、 NOT_READY 状況を読み h る前に SSCP データを送信することができ、また、 NOT_READY 状況の読み h り後は、SSCP データの送信および u 信の両方とも行うことができます。

- SESSION_END_REQUESTED

  ホストから SHUTD コマンドを u 信したことを (します。ホストは、SLI アプリケーションがセッションをできるだけ早く終了させることを要 a しています。

  アプリケーションがセッション終了の作動可能状態である場合は、 **SLI_OPEN** を / 行する必要があります。

- INIT_COMPLETE

  SLI_OPEN 処理時に、**RUI_INIT** verb が完了したことを (します。この状況は、**SLI_OPEN lua_init_type** パラメーターの値が LUA_INIT_TYPE_PRIM_SSCP である場合にのみ戻されます。

  この状況の u 信後は、アプリケーションは SSCP 通常フロー上のデータを送 u 信することができます。

ホスト・アプリケーションが送信する要 a 単位が例外要 a (EXR) に変換されている場合は、戻りコードに加えて追加 SNA センス・データを戻すことができます。以下の戻り verb パラメーター値を使用して **SLI_BID** を完了させると、EXR が (されます。

パラメーター
値

lua_prim_rc
OK (X'0000')

lua_sec_rc
OK (X'00000000')

lua_rh.rri
ビット・オフ (要 a 単位)

lua_rh.sdi
ビット・オン (センス・データを含む)

これらの条 o の下で要 a は EXR に変換されており、最大 7 バイトまでの情報が **lua_peek_data** verb パラメーターに戻されます。**lua_peek_data** パラメーター中の情報のフォーマットは以下のとおりです。

- バイト 0 - 3 には、検出されたエラーを定 A するセンス・データが入っています。 LUA が、要 a を EXR に変換した場合は、センス・データは以下の値のどれかをとります。

センス・データ	バイト 0 - 3 の値
LUA_MODE_INCONSISTENCY	X'08090000'
LUA_BRACKET_RACE_ERROR	X'080B0000'
LUA_BB_REJECT_NO_RTR	X'08130000'
LUA_RECEIVER_IN_TRANSMIT_MODE	X'081B0000'
LUA_CRYPTOGRAPHY_FUNCTION_INOP	X'08480000'
LUA_SYNC_EVENT_RESPONSE	X'10010000'
LUA_RU_DATA_ERROR	X'10020000'
LUA_RU_LENGTH_ERROR	X'10020000'
LUA_INCORRECT_SEQUENCE_NUMBER	X'20010000'

lua_peek_data のバイト 4 からバイト 6 に戻される情報には、オリジナル要 a 単位の先頭の 3 バイトまでが } 容されます。

SLI_CLOSE

この verb は SNA セッションをクローズします。SLI_CLOSE は、ホスト・アプリケーション・プログラムとの接続を終了させ、使用されたリソースを解放します。SLI_CLOSE の通知は、LU-LU コミュニケーションおよび SSCP-LU コミュニケーションが終了したことを表すものです。

指定パラメーター

アプリケーションは！のパラメーターを提供します。

lua_verb
LUA_VERB_SLI

LUA verb 用の verb コード標 1 。

lua_verb_length
verb 制御ブロックの長さ。この数値は、SLI_CLOSE verb 用の SLI により予期される長さと等しくなければなりません。

lua_opcode
LUA_OPCODE_SLI_CLOSE

この verb 用の操作コード。 SLI_CLOSE 用のもの。

lua_correlator
この verb を、プログラムが提供する他の情報に関連づけるために LUA アプリケーション・プログラムが提供できる値。 LUA インターフェースは、このパラメーターを使用しません。

lua_luname
ASCII 表 - のローカル LU 名。 名前が 8 文字未満であれば、ブランク文字を埋め込んで補う必要があります。 LUA は、このパラメーターを **lua_sid** が 0 の場合にのみ検査します。すべての verb に **lua_luname** パラメーターを使用すると、デバッグが一層簡単になり、特に複数の LU 構成の場合に効果的です。

lua_sid
正常に完了した SLI verb が戻す セッション ID で、使用するセッションを 1 別するもの。このパラメーターが 0 である場合は、**lua_luname** パラメーターは、 1 別用に使用されます。

lua_post_handle
これは、 s 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.close_abend
そのクローズが、即時クローズ (on) であるか通常のクローズ (off) であるかを指定します。

戻りパラメーター

verb が正常に完了した場合は、！のパラメーターが戻されます。

lua_flag2.async
この verb の s 同期完了を (すフラグ。

lua_prim_rc
verb ！能が設定する 1 ！戻りコード。 詳細については、 369ページの『付録B. LUA Verb 戻りコード』 を参照してください。

lua_sec_rc
verb ！能が設定する 2 ！戻りコード。 詳細については、 369ページの『付録B. LUA Verb 戻りコード』 を参照してください。

使用上の注意

SLI_CLOSE には 2 つのタイプがあります。正常クローズおよびアベンド・クローズです。

- 正常クローズ

  正常クローズは、**lua_flag1.close_abend** パラメーターが 0 に設定されることにより 1 別されます。クローズ順序は、2 ！開始または 1 ！開始のどちらかです。正常クローズは、 1 ！開始に対して SHUTD コマンドを使用します。正常クローズは SHUTD コマンドを 1 ！開始クローズに使用し、！に RSHUTD コマンドを 2 ！開始クローズに送ります。

  ホストが UNBIND タイプ X'02' (BIND が予定されているUNBIND) を、 1 ！開始または 2 ！開始された正常 **SLI_CLOSE** 時に送信すると、セッションはクローズされません。 **SLI_CLOSE** verb は、 CANCELED 1 ！戻りコードおよび RECEIVED_UNBIND_HOLD 2 ！戻りコードを戻して完了します。アプリケーション・プログラムは、STATUS を戻すために **SLI_BID** または **SLI_RECEIVE** verb を / 行する必要があります。

  ホストが UNBIND タイプ X'01' (通常の UNBIND) を、 1 ！開始または 2 ！開始された正常 **SLI_CLOSE** 時に送信し、かつ、このセッション用の **SLI_OPEN** verb が指定されていて、LUA_SESSION_TYPE_DEDICATED の **lua_session_type** が指定されていると、セッションはクローズされません。 **SLI_CLOSE** verb は、CANCELED 1 ！戻りコードおよび RECEIVED_UNBIND_NORMAL 2 ！戻りコードを戻して完了します。アプリケーション・プログラムは、STATUS を戻すために **SLI_BID** または **SLI_RECEIVE** を / 行する必要があります。

- アベンド・クローズ

  アベンド・クローズは、 **lua_flag.close_abend** パラメーターが 1 に設定されることにより 1 別されます。CLOSE_ABEND オプションは、 SLI に即時にセッションを終了するよう指 (します。

以下の SNA コマンドは、異なるタイプのクローズ処理時にも使用できます。

- 正常 **SLI_CLOSE**
  - 2 ！開始クローズ

    SLI アプリケーション・プログラムが、**lua_flag.close_abend** が 0 に設定された **SLI_CLOSE** verb を / 行した後で、 SLI は以下の処理を B 行します。

    RSHUTD コマンドを作成する
    RSHUTD コマンド ~ 答を読み h り、プロセスする
    CLEAR コマンド (必要な場合には) を読み h り、プロセスする

CLEAR コマンド~答 (必要な場合には) を作成する
UNBIND コマンドを読み h り、プロセスする
UNBIND コマンド~答を作成する
RUI セッションを停止させる

– 1 ！開始クローズ

SHUTD コマンドを読み h り、アプリケーションに SESSION_END_REQUESTED 状況を与えます。

SLI アプリケーション・プログラムが、**lua_flag.close_abend** が 0 に設定された **SLI_CLOSE** を / 行した後で、 SLI は以下の処理をB行します。

CHASE コマンドを作成する
CHASE コマンド~答を読み h り、プロセスする
シャットダウン完了 (SHUTC) コマンドを作成する
SHUTC コマンド~答を読み h り、プロセスする
CLEAR コマンド (必要な場合には) を読み h り、プロセスする
CLEAR コマンド~答 (必要な場合には) を作成する
UNBIND コマンドを読み h り、プロセスする
UNBIND コマンド~答を作成する
RUI セッションを停止させる

– アベンド **SLI_CLOSE**

- SLI アプリケーション・プログラムが、**lua_flag1.close_abend** が 1 に設定された **SLI_CLOSE** verb を / 行した後で、 SLI は RUI セッションを停止させます。

SLI_CLOSE verb の完了は、LU-LU セッションが未 k 合であることおよび、 SSCP が LU 用のセッション・キャパシティーを持たない旨を通知されたことを暗黙指定します。 **SLI_CLOSE** verb が正常に完了すると、別の **SLI_OPEN** 以外には、他の SLI コマンドを / 行することはできません。 **SLI_CLOSE** verb を u け h ると、すべての保留コマンドは終了します。

注:

1. RUI を使用して確立されたセッションのクローズに、このファンクションを使用しないでください。
2. 正常 **SLI_CLOSE** を / 行する前には、すべての未送~答がホストに送信されたことを確認してください。~答が未送である場合は、 SLI は+動的に CLOSE タイプを ABEND に変更します。

   LUA アプリケーション・プログラムがデータを無視する時も、 CLOSE タイプは +動的に ABEND への変更を行う場合があります。 **SLI_RECEIVE** verb を使用して、すべてのデータをホストから u 信するのは、良いプログラミングのやり方です。このようなやり方をしないと、データが例外要 a であった場合でも、 SLI は~答が未送であると想定し、CLOSE タイプを ABEND に変更します。

SLI_OPEN

この verb は、リンク上でセッション・レベルのコミュニケーションを要 a しているアプリケーション・プログラム用の SNA セッションをオープンします。セッション・レベルのファンクションは、セッションをオープンするアプリケーション・プログラムのために、 SNA コマンドを / 行します。 SLI ファンクションは、LU - LU セッションを確立するために、複数の RUI ファンクションをB行するため、LUA アプリケーション・プログラムは単純化されています。

指定パラメーター

このアプリケーションは、以下のパラメーターを提供します。

lua_verb

LUA_VERB_SLI

LUA verb 用の verb コード標 1 。

lua_verb_length

verb 制御ブロックの長さ。この数値は、SLI_OPEN verb 用の SLI により予期される長さと等しくなければなりません。

lua_opcode

LUA_OPCODE_SLI_OPEN

lua_correlator

この verb を、プログラムが提供する他の情報に関連づけるために LUA アプリケーション・プログラムが提供できる値。 Windows LUA インターフェースは、このパラメーターを使用しません。

lua_luname

ASCII 表 - のローカル LU 名。 名前が 8 文字未満であれば、ブランク文字を埋め込んで補う必要があります。

このパラメーターは **SLI_OPEN** に必須です。他の verb は、このパラメーターを **lua_sid** パラメーターがゼロである場合にのみ必要とします。しかし、**lua_luname** パラメーターをすべての verb に対して使用すると、デバッグが一層簡単になり、特に複数の LU 構成の場合に効果的です。

以下の情報は、Communications Server Windows 95 および Windows NT SNA API クライアントに対してのみ適用されます。

それぞれのユーザーについてのデフォルトの LUA セッション名は、該当する構成ユーティリティー、すなわち INI 構成または LDAP のどちらか、を使用して割り当てることができます。

3270 のような LUA プログラムは、デフォルトの LUA セッション名を直接指定するのではなく、選択して使用することができます。LUA プログラムが、 **lua_name** フィールドが 2 進ゼロまたは ASCII のブランクに設定されている、 **SLI_OPEN** verb を / 行すると、 SLI API は構成済みのデフォルト LUA セッション名を使用します。

lua_data_length

送信されている不定様 0 LOGON または INITSELF データの長さ。

lua_data_ptr

アプリケーションのデータ・バッファーを指すポインター。このバッファーは、データおよび SNA コマンドに使用されるため、バッファーの内容は通常は EBCDIC 表 (です。

このデータ・バッファーには以下のうちの 1 つが入ります。

- ユーザーの SNA INITSELF 要 a 単位 (RU)。lua_init_type パラメーターが INITSELF の 2 ！開始を指定している場合は、要 a されているすべてのアプリケーション・プログラム・データが - 入されています。 INITSELF には、モード名および PLU 名のようなユーザー情報が } 容されます。詳細については、*Systems Network Architecture Network Product Formats* を参照してください。
- **lua_init_type** パラメーターに不定様 0 LOGON メッセージの 2 ！開始が指定されている場合に、通常の SSCP フローに送信される LOGON メッセージ。
- セッションが 1 ！開始である場合は、このバッファーは使用されず、**lua_data_ptr** パラメーターは 0 になります。

lua_post_handle

イベントが s 同期通知を行う場合は、**lua_post_handle** にはシグナルを u けるイベントのハンドルが } 容されます。

lua_encr_decr_option

暗号化はサポートされません。

lua_init_type

Windows LUA インターフェースの LU-LU セッションの開始方法を定 A します。有効な値は以下のとおりです。

LUA_INIT_TYPE_SEC_IS

2 ！開始。OPEN のデータ・バッファーで提供される INITSELF コマンドを送信します。

LUA_INIT_TYPE_SEC_LOG

OPEN のデータ・バッファー中で指定される不定様 0 LOGON メッセージの 2 ！開始

LUA_INIT_TYPE_PRIM

1 ！開始。BIND を待！します

LUA_INIT_TYPE_PRIM_SSCP

SSCP アクセスの 1 ！開始

lua_session_type

SLI が UNBIND タイプ X'01'、正常 UNBIND をプロセスする方法を定 A する値。有効な値は、以下のとおりです。

LUA_SESSION_TYPE_NORMAL

1 ！論理装置から正常 UNBIND を u 信すると、SLI は肯定 ~ 答を送信し、 NOTIFY の SSCP へのフローを使用不可にする **RUI_TERM** を / 行します。SSCP-LU フローは使用不可になります。これはこのパラメーターのデフォルトの値です。

LUA_SESSION_TYPE_DEDICATED

正常 UNBIND を 1 ！論理装置から u けｈると、SLI は肯定～答を送信し、 SLI セッションは、新， BIND、オプションの CRV と STSN、および SDT コマンドを u 信するまで中断状態になります。この場合、SLI は **RUI_TERM** を / 行せず、使用不可の NOTIFY は SSCP へのフローを行えません。

LUA_SESSION_TYPE_DEDICATED は、SNA API クライアントではサポートされません。

lua_wait

ホストが以下のどれかのメッセージを送信した後で、SLI が INITSELF または LOGON メッセージの伝送の + 動的再試行の前に待！する秒数 (最大 65 535)。

- INITSELF または LOGON メッセージに対する] 定～答および 2 ！戻りコードが以下の値のどれか 1 つである場合。
  - RESOURCE_NOT_AVAILABLE (X'08010000')
  - SESSION_LIMIT_EXCEEDED (X'08050000')
  - SSCP_LU_SESS_NOT_ACTIVE (X'0857nnnn' この場合 *nnnn* は X'0002')
  - SESSION_SERVICE_PATH_ERROR (X'087Dnnnn' この場合 *nnnn* は X'0000')
- ネットワーク・サービス・プロシージャー・エラー (NSPE) メッセージ
- プロシージャー・エラーを (している NOTIFY コマンド

lua_wait の値が 0 である場合は、再試行は行われません。このパラメーターは、SLU が開始したセッションに対してのみ適用されます。PLU がセッションを開始すると、**lua_wait** は無視されます。

lua_extension_list_offset

verb 制御ブロックの開始からユーザー指定の拡張 DLL までのオフセットを指定します。この値はワード境界の先頭でなければなりません。拡張リストがない場合は、値はゼロに設定されます。

lua_routine_type

以下のモジュール名および j 続き名のルーチンのタイプ。有効なエントリーは、以下のとおりです。

lua_routine_type_bind
バインド・ルーチン

lua_routine_type_crv
暗号化ベクトル・ルーチン

注: 暗号化は現在はサポートされません。

lua_routine_type_sdt
開始データ・トラフィック (SDT) ルーチン

lua_routine_type_sdt は、SNA API クライアントではサポートされません。

lua_routine_type_stsn
設定およびテスト順序V号 (STSN) ルーチン

lua_routine_type_end
ルーチンのリストの終了区切り文字。

lua_module_name
ユーザー提供の ASCII モジュール名を提供します。このパラメーターは、最大 8 文字までの長さで、残りのバイトは X'00' に設定します。

lua_procedure_name
ユーザー提供の DLL プロシージャー名で、ASCII A 0 です。このパラメーターは、最大 32 文字までの長さで、残りのバイトは X'00' に設定します。

戻りパラメーター

verb が正常に完了した場合は、！のパラメーターが戻されます。

lua_flag2.async
この verb の s 同期完了を (すフラグ。

lua_sid
後続の verb が、使用するセッションを 1 別するためのセッション ID。このパラメーターの値は、1 ！戻りコードが OK または IN_PROGRESS である場合にのみ有効です。 IN_PROGRESS を戻した後に、 **SLI_OPEN** が : 敗すると、セッション ID はそれ以降は無効になります。

lua_prim_rc
verb ！能が設定する 1 ！戻りコード。 詳細については、 369ページの『付録B. LUA Verb 戻りコード』 を参照してください。

lua_sec_rc
verb ！能が設定する 2 ！戻りコード。 詳細については、 369ページの『付録B. LUA Verb 戻りコード』 を参照してください。

使用上の注意

SLI は、以下のセッション初期化タスクをB行することができます。

- RUI セッションの開始
- INITSELF または不定様 0 ログオン・メッセージ (2 ！初期化のみ) の作成
- INITSELF ～答の読み h りとプロセス、またはログオン・メッセージへの～答 (2 ！初期化のみ)。
- ホストから来た BIND コマンドの読み h りと検査
- BIND ～答の作成
- ホストにより送信される場合の UNBIND タイプ X'02'、または UNBIND タイプ X'01' の読み h りとプロセス。
- UNBIND ～答の作成、および後続の BIND の u 信準w。
- STSN コマンド (必要な場合) の読み h りとプロセス。
- STSN ～答 (必要な場合) の作成。
- SDT コマンドの読み h りとプロセス。

- SDT ~答の作成。
- アプリケーション・プログラムの **SLI_OPEN** verb 中に指定されている場合は、ユーザー・ルーチンに移動して行う BIND、STSN、および SDT コマンドのプロセス。

SLI_OPEN verb は、すべての SNA メッセージ・トラフィックを、 SDT コマンドへの~答を行うことによりハンドルします。

アプリケーション・プログラムは **SLI_OPEN** verb を / 行して、 **lua_luname** パラメーターに特別に定 A されている LUA LU を選択します。このフィールドは、 ASCII 文字列で、ブランクが埋め込まれている必要があります。

lua_init_type パラメーターは、SLI に LU セッションの確立方法を指 (します。以下のリストは、初期化オプションを - 述するものです。

- INITSELF による 2 ！初期化

  このオプション用に **lua_init_type** パラメーターを LUA_INIT_TYPE_SEC_IS に設定します。このオプションでは、アプリケーション・プログラムは **SLI_OPEN** verb 中に使用される INITSELF コマンドを提供する必要があります。なぜなら、INITSELF にはモード名や PLU 名のような、ホストが必要とするすべてのセッション固有の情報があるからです。 **lua_data_ptr** パラメーターは INITSELF のアドレスを提供し、また、**lua_data_length** パラメーターはその長さを提供します。
- 不定様 0 LOGON メッセージによる 2 ！初期化

  このオプション用に **lua_init_type** パラメーターを LUA_INIT_TYPE_SEC_LOG に設定します。不定様 0 LOGON メッセージによる、 2 ！初期化では、**lua_data_ptr** パラメーターは、 **lua_data_length** パラメーターで指定されている長さをもつ、ユーザーの EBCDIC LOGON メッセージのアドレスを含んでいます。
- 1 ！初期化

  このオプション用に **lua_init_type** パラメーターを LUA_INIT_TYPE_PRIM に設定します。 1 ！初期化では、SLU はホストとのセッションを開始するためには何もしません。 **SLI_OPEN** は、ホストが BIND コマンドおよび後続の SDT コマンドにより開始されるまで IN_PROGRESS のままです。
- SSCP による1 ！初期化

  このオプション用に **lua_init_type** パラメーターを LUA_INIT_TYPE_PRIM_SSCP に設定します。 SSCP アクセスによる 1 ！初期化では、SLI は、セッションを開始するためのホストへのコマンドの送信を行いません。その代わりに、SLI はアプリケーション・プログラムに **SLI_SEND** verb および **SLI_RECEIVE** verb を SSCP 通常フロー用に / 行させ、 INITSELF コマンドまたは LOGON メッセージを送信し、それらに対する ~ 答を u 信します。このオプションを使用すると、アプリケーション・プログラムは 2 ！初期化タイプであるため、 1 つの INITSELF または LOGON メッセージに限定されません。これは **SLI_OPEN** の完了前にアプリケーション・プログラムに SLI verb を / 行させる、唯一の **SLI_OPEN** タイプです。 **SLI_OPEN** verb が / 行された後は、アプリケーション・プログラムは、 **SLI_BID** または **SLI_RECEIVE** を / 行して、 INIT_COMPLETE 状況を入 j することができます。この状況は、アプリケーション・プログラムに SSCP 通常フロー・データ用に **SLI_SEND** verb および **SLI_RECEIVE** verb の / 行開始が可能であることを知らせます。

オプションの **lu_session_type** パラメーターは、 SLI に UNBIND タイプ X'01'、正常 UNBIND のプロセス方法を知らせます。このパラメーターは、 **SLI_OPEN** verb

が初期パラメーター検査をパスした後に効力が生じ、その効力は **SLI_CLOSE** アベンドが / 行されるか、または SLI が **RUI_TERM** を / 行するまで持続します。以下のリストは、標準 UNBIND および占有 UNBIND の処理を説明するものです。

- 正常 **SLI_CLOSE** を処理する標準の正常 UNBIND

  このオプション用に **lua_session_type** パラメーターを、LUA_SESSION_TYPE_NORMAL に設定します。これはデフォルトの値です。このオプションを使用すると、 SLI は肯定~答を 1 ！ LU が送信した正常 UNBIND に送信し、！に **RUI_TERM**を / 行します。これにより、NOTIFY の SSCP へのフローが使用不可になります。 これらのアクションにより、以下の処理が行われます。

  – LU-LU セッションを終了させる。

  – SSCP および PLU に、SLU は新 , BIND をプロセスできないことを指 (する。u 信した新 , BIND はリジェクトされる。

  – データが SSCP-LU セッションに流れ込むのを阻止する。

  SLI は、タイプ X'02' (BIND が予定されているUNBIND) 以外のすべての UNBIND を u けh った時に **RUI_TERM** を / 行します。

  – 占有正常 UNBIND 処理

  このオプション用に **lua_session_type** パラメーターを、LUA_SESSION_TYPE_DEDICATED に設定します。このオプションを使用すると、SLI は肯定~答を 1 ！論理装置が送信した正常 UNBIND に送信します。しかし、SLI は **RUI_TERM** を / 行しません。 SSCP-LU セッションの状況は変更されません (使用可能化されている)。 SLI セッションは、BIND、オプションの CRV と STSN、および SDT コマンドを u 信するまで中断状態になります。新 , BIND を待！している SLI セッションは、アベンド **SLI_CLOSE** を / 行することにより終了させることができます。

  SLI は、タイプ X'02' またはタイプ X'01' 以外のすべての UNBIND を u 信すると、 **RUI_TERM** を / 行します。

  このオプションは、1 ！ LU が、BIND が予定されている UNBIND を送信できないが、正常 UNBIND が送信される時にはこのタイプの動作を行える場合には、役にたちます。

アプリケーションが提供する BIND、SDT、または STSN ルーチン

- アプリケーション・プログラムが、BIND、SDT、または STSN ルーチンを提供する場合は、 DLL モジュール名およびプロシージャー・エントリー・ポイントは、**SLI_OPEN** 拡張ルーチン・リストに渡されます。対~する SNA 要 a が u 信されると、これらのルーチンは **SLI_OPEN** 時に呼び出されます。BIND ルーチンが提供されない場合は、 SLI は BIND 検査の限定された部分だけを必要に~じてB行します。 STSN ルーチンが提供されず、かつ STSN 要 a が u 信された場合は、 SLI は選択可能な情報が存在しないことを (すために肯定~答を / 行します。 SDT ルーチンが提供されず、かつ SDT 要 a が u 信された場合は、 SLI は肯定~答を / 行します。

通知

- **lua_prim_rc** パラメーターが OK になっている **SLI_OPEN** を通知することは、**SLI_OPEN** の正常な完了、および LU-LU データ・フロー・セッションの確立を意味します。このセッションが正常にオープンした後では、アプリケーション・プ

ログラムは **SLI_SEND**、**SLI_RECEIVE**、 **SLI_PURGE**、**SLI_BID**、または **SLI_CLOSE** verb を / 行することができます。

セッションの回復

- SLI は、アプリケーション・プログラムに限られたO囲のセッション回復を提供します。 SLI verb のどれかが **lua_prim_rc** パラメーターに、 SESSION_FAILURE を (して完了すると、アプリケーション・プログラムは **SLI_OPEN** を再 / 行しなければなりません。 この状態では、プログラムは新 , の **SLI_OPEN** verb を / 行する前に、**SLI_CLOSE** verb を / 行する必要はありません。

保留 SLI_OPEN の終了

- 保留されている **SLI_OPEN** を終了させるには、 **lua_flag1.close_abend** パラメーターを 1 に設定した **SLI_CLOSE** を / 行します。

SLI_PURGE

この verb は、未解 h の **SLI_RECEIVE** を除 n します。 **SLI_PURGE** は、WAIT オプションを持つ **SLI_RECEIVE** verb を使用するアプリケーション・プログラムが必要とするものです。たとえば、**SLI_RECEIVE** verb が指定された 時間間隔中に完了しない場合、アプリケーション・プログラムは **SLI_PURGE** を / 行することができます。アプリケーション・プログラムは、**lua_data_ptr** パラメーター中の、**SLI_RECEIVE** verb 制御ブロックのアドレスを提供して、どの **SLI_RECEIVE** を除 n すべきかを指定します。

指定パラメーター

このアプリケーションは、以下のパラメーターを提供します。

lua_verb

LUA_VERB_SLI

LUA verb 用の verb コード標 1 。

lua_verb_length

verb 制御ブロックの長さ。この数値は、**SLI_PURGE** verb 用の SLI により予期される長さと等しくなければなりません。

lua_opcode

LUA_OPCODE_SLI_PURGE

verb の操作コード。

lua_correlator

この verb を、プログラムが提供する他の情報に関連づけるために LUA アプリケーション・プログラムが提供できる値。 LUA インターフェースは、このパラメーターを無視します。

lua_luname

ASCII 表 - のローカル LU 名。 名前が 8 文字未満であれば、ブランク文字を埋め込んで補う必要があります。 LUA は、 **lua_sid** が 0 の場合にのみこのパラメーターを検査します。すべての verb に **lua_luname** パラメーターを使用すると、デバッグが一層簡単になり、特に複数の LU 構成の場合に効果的です。

lua_sid

SLI_OPEN が戻すセッション ID、使用するセッションを 1 別します。このパラメーターが 0 である場合は、**lua_luname** パラメーターは 1 別用に使用されます。

lua_data_ptr

除 n すべきアプリケーション・プログラム **SLI_RECEIVE** verb 制御ブロックを指すポインター。

lua_post_handle

イベントが s 同期通知を行う場合は、**lua_post_handle** にはシグナルを u けるイベントのハンドルが } 容されます。

戻りパラメーター

verb が正常に完了した場合は、！のパラメーターが戻されます。

lua_flag2.async
この verb の s 同期完了を (すフラグ。

lua_prim_rc
verb ！能が設定する 1 ！戻りコード。 詳細については、 369ページの『付録B. LUA Verb 戻りコード』 を参照してください。

lua_sec_rc
verb ！能が設定する 2 ！戻りコード。 詳細については、 369ページの『付録B. LUA Verb 戻りコード』 を参照してください。

使用上の注意

SLI_RECEIVE が正常に除 n されると、**SLI_RECEIVE** は CANCELED 1 ！戻りコードを戻して終了し、また **SLI_PURGE** は OK 1 ！戻りコードを戻して完了します。

SLI_RECEIVE

この verb は、データまたは状況コードをアプリケーション・プログラムに転送します。 SLI_RECEIVE はまた、セッションの現況を Windows LUA アプリケーションに提供します。

LU-LU セッション・フロー用の **SLI_RECEIVE** verb は、オープンされているセッション上でのみ / 行できます。**SLI_OPEN** 開始タイプが SSCP アクセスについては 1 ！である場合は、アプリケーション・プログラムは、 **SLI_OPEN** verb が保留状態であっても、 **SLI_RECEIVE** verb を SSCP-LU 通常フロー・データ用に / 行することができます。

指定パラメーター

このアプリケーションは、以下のパラメーターを提供します。

lua_verb
LUA_VERB_SLI

LUA verb 用の verb コード標 1 。

lua_verb_length
verb 制御ブロックの長さ。この数値は、SLI_RECEIVE verb 用の SLI により予期される長さと、等しくなければなりません。

lua_opcode
LUA_OPCODE_SLI_RECEIVE

lua_correlator
この verb を、プログラムが提供する他の情報に関連づけるために LUA アプリケーション・プログラムが提供できる値。 LUA インターフェースは、このパラメーターを無視します。

lua_luname
ASCII 表 - のローカル LU 名。 名前が 8 文字未満であれば、ブランク文字を埋め込んで補う必要があります。 LUA は、 **lua_sid** が 0 の場合にのみこのパラメーターを検査します。すべての verb に **lua_luname** パラメーターを使用すると、デバッグが一層簡単になり、特に複数の LU 構成の場合に効果的です。

lua_sid
SLI_OPEN が戻すセッション ID で、使用するセッションを 1 別します。このパラメーターが 0 である場合は、**lua_luname** パラメーターは 1 別用に使用されます。

lua_max_length
データの u 信に使用するバッファーの長さ。

lua_data_ptr
SLI がホスト・アプリケーションから u 信したデータを入れるバッファーを指すポインター。このバッファーは、データおよび SNA コマンドに使用されるため、バッファーの内容は通常は EBCDIC 表 (です。

lua_post_handle
Windows NT では、イベントが s 同期通知を行う場合は、**lua_post_handle** には、シグナルを u けるイベントのハンドルが } 容されます。

lua_flag1.bid_enable
LUA が、SLI_BID verb 制御ブロックを LUA アプリケーション・プログラム用に、再利用すべきかどうかを指定するフラグ。

lua_flag1.nowait
読み h るデータがない場合に、 SLI が戻りコード NO_DATA を SLI_RECEIVE verb に通知するように知らせるフラグ。複数 RU チェーンの最初の RU が到着し、かつ、**lua_flag1.nowait** オプションが選択されている場合は、**lua_flag1.nowait** オプションは無視されます。 SLI_RECEIVE verb は、チェーンのすべての RU が到着した後で IN_PROGRESS を戻し、 s 同期に完了します。チェーニングが使用できる場合は、**lua_flag1.nowait** オプションは使用できません。

lua_flag1 の下位のハーフバイトには、メッセージ・セッションおよび流れを - 述するフラグが } 容されます。このフロー・フラグは、LUA アプリケーション・プログラムがメッセージを u 領できるフローを - 述します。以下のフラグの中の少なくとも 1 つが設定されていなければならず、また、設定済みフラグは、別のアクティブ **SLI_RECEIVE** verb に設定されているフラグとオーバーラップできません。

lua_flag1.sscp_exp
SSCP ^ 送フローを指定するフラグ。

lua_flag1.sscp_norm
SSCP 通常フローを指定するフラグ。

lua_flag1.lu_exp
LU ^ 送フローを指定するフラグ。

lua_flag1.lu_norm
LU 通常フローを指定するフラグ。

戻りパラメーター

verb が正常に完了した場合は、！のパラメーターが戻されます。

lua_data_length
u 信するデータの長さ。

lua_th メッセージ用の SNA 伝送ヘッダー (TH) を } 容する 6 バイトのパラメーター。

lua_rh メッセージ用の SNA 要 a / ~ 答ヘッダー (RH) を } 容する 3 バイトのパラメーター。

lua_message_type
SNA データおよびコマンドのタイプ。SLI アプリケーション・プログラムが、データを送信しようとする時は、このパラメーターの設定が必要です。有効なメッセージ・タイプは以下のとおりです。

LUA_MESSAGE_TYPE_LU_DATA

LUA_MESSAGE_TYPE_SSCP_DATA

LUA_MESSAGE_TYPE_RSP
LUA_MESSAGE_TYPE_BID
LUA_MESSAGE_TYPE_BIS
LUA_MESSAGE_TYPE_CANCEL
LUA_MESSAGE_TYPE_CHASE
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_SIGNAL

LU_DATA、LUSTAT_LU、LUSTAT_SSCP、および SSCP_DATA は SNA コマンドではありません。

lua_flag2.async
この verb は s 同期に完了することを指 (するフラグ。

lua_flag2.sscp_exp
SSCP ^ 送フローを指定するフラグ。

lua_flag2.sscp_norm
SSCP 通常フローを指定するフラグ。

lua_flag2.lu_exp
LU ^ 送フローを指定するフラグ。

lua_flag2.lu_norm
LU 通常フローを指定するフラグ。

lua_prim_rc
verb ！能が設定する 1 ！戻りコード。 詳細については、 369ページの『付録B. LUA Verb 戻りコード』 を参照してください。

lua_sec_rc
verb ！能が設定する 2 ！戻りコード。 詳細については、 369ページの『付録B. LUA Verb 戻りコード』 を参照してください。

使用上の注意

SLI_RECEIVE は、ホストから ~ 答、SNA コマンド、および要 a 単位データを u 信します。**SLI_RECEIVE** はまた、セッションの状況を Windows LUA アプリケーションに提供します。**SLI_OPEN** 要 a は、**SLI_RECEIVE** の / 行以前に完了していることが必要です。しかし、lua_init_type が LUA_INIT_TYPE_PRIM_SSCP に設定されている **SLI_OPEN** が / 行されると、SSCP 通常フローの **SLI_RECEIVE** は、 **SLI_OPEN** が IN_PROGRESS を戻すと直ちに、 / 行されます。

データは、4 つのセッションの流れの 1 つに従いアプリケーションに u 信されます。4 つのセッションの流れを、優先順位の高い方から低い方へ、以下に s げておきます。

- SSCP ^送
- LU ^送
- SSCP 通常
- LU 通常

SLI_RECEIVE verb がプロセスするデータ流れタイプは、 **lua_flag1** 中に指定されます。 アプリケーションはまた、複数のデータ・フローのタイプを調べる必要があるかどうかも指定します。複数のフローのビットが設定されている場合は、もっとも高い優先順位を持つものが、最初に u 信されます。**SLI_RECEIVE** が処理を完了すると、 **lua_flag2** は、Windows LUA アプリケーションによりデータが u 信されている、特定のフローのタイプを指 (します。

SLI_RECEIVE が / 行される前に SLI_BID が正常に完了すると、 Windows LUA インターフェースには、最新の SLI_BID の verb 制御ブロックを再利用するよう指 (することができます。このためには、**lua_flag1.bid_enable** パラメーターを 1 に設定した **SLI_RECEIVE** を / 行します。

lua_flag1.bid_enable パラメーターを使用する場合は、**SLI_BID** - 憶域を解放してはいけません。なぜなら、最後の **SLI_BID** verb の verb 制御ブロックが使用されているからです。また、**lua_flag1.bid_enable** パラメーターを使用すると、**SLI_BID** の正常終了が通知されます。

選択可能な u 信が存在しない場合に、**lua_flag1.nowait** を持つ **SLI_RECEIVE** を / 行すると、Windows LUA インターフェース が設定する 2 ！戻りコードは LUA_NO_DATA です。

状況が選択可能であれば、アプリケーションはその状況を読み h る必要があります。アプリケーションが **SLI_BID** または **SLI_RECEIVE** を / 行して状況を読み h るまで、他のすべての操作はリジェクトされますが、以下については例外です。

- SSCP フロー上の **SLI_SEND** verb
- **SLI_CLOSE**

1 ！戻りコードが STATUS である場合は、戻される **SLI_RECEIVE** パラメーターは **lua_prim_rc**、**lua_sec_rc**、 および **lua_sid** です。アクティブな **SLI_BID** verb がない場合のみは、アクティブ **SLI_RECEIVE** verb に、STATUS 戻りコードを使用して、通知することができます。

1 ！戻りコードの値が STATUS である場合は、可能な 2 ！戻りコードの値は以下のとおりです。

- READY

  SLI セッションが、現在すべての追加コマンドの処理用に作動可能であることを (します。前に出された NOT_READY 状況の u 信後に、READY 状況が / 行されます。

- NOT_READY

  タイプ値 X'02' または X'01' を持つ CLEAR コマンドまたは UNBIND コマンドをホストから u 信したことを (します。 SLI セッションは中断状態になります。

  – CLEAR が到着すると、セッションは SDT コマンドを u 信するまで中断状態となります。

- UNBIND タイプ X'02' (BIND が予定されている UNBIND) が到着すると、セッションは、BIND、オプションの CRV と STSN、および SDT コマンドを u 信するまで中断状態になります。すべてのユーザー拡張ルーチンは、再入可能になっている必要があります。
- UNBIND タイプ X'01' (通常の UNBIND) が到着し、かつ、このセッション用の **SLI_OPEN** verb で LUA_SESSION_TYPE_DEDICATED の **lua_session_type** を指定していた場合は、セッションは BIND、オプションの CRV と STSN、および SDT コマンドを u 信するまで中断状態になります。 これらのコマンドをプロセスするために提供されるユーザー拡張ルーチンは、再入可能となっている必要があります。

  CLEAR、UNBIND タイプ X'02'、または UNBIND タイプ X'01' の到着後は、アプリケーションは NOT_READY 状況を読み h る前に SSCP データを送信することができ、また、 NOT_READY 状況の読み h り後は、SSCP データの送信および u 信の両方とも行うことができます。

- SESSION_END_REQUESTED

  ホストから SHUTD コマンドを u 信したことを (します。ホストは、SLI アプリケーションがセッションをできるだけ早く終了させることを要 a しています。

  アプリケーションがセッション終了作動可能である場合は、**SLI_CLOSE** または正常 **SLI_CLOSE** を / 行する必要があります。

- INIT_COMPLETE

  SLI_OPEN 処理時に、**RUI_INIT** verb が完了したことを (します。この状況は、**SLI_OPEN lua_init_type** パラメーターの値が LUA_INIT_TYPE_PRIM_SSCP である場合にのみ戻されます。

  この状況の u 信後は、アプリケーションは SSCP 通常フロー上のデータを送 u 信することができます。

ホスト・アプリケーションが送信する要 a 単位が例外要 a (EXR) に変換されている場合は、戻りコードに加えて追加 SNA センス・データを戻すことができます。以下の戻り verb パラメーター値を使用して **SLI_RECEIVE** を完了させると、 EXR が (されます。

パラメーター
値

lua_prim_rc
OK (X'0000')

lua_sec_rc
OK (X'00000000')

lua_rh.rri
ビット・オフ (要 a 単位)

lua_rh.sdi
ビット・オン (センス・データを含む)

これらの条 o の下で要 a は EXR に変換されており、最大 7 バイトまでの情報が、アプリケーション・バッファーに戻されます。データ・バッファー中の情報のフォーマットは、以下のとおりです。

- バイト 0−3 には、検出されたエラーを定Aするセンス・データが入っています。LUA が要 a を EXR に変換した場合は、センス・データは以下の値のどれかをとります。

センス・データ	バイト 0 - 3 の値
LUA_MODE_INCONSISTENCY	X'08090000'
LUA_BRACKET_RACE_ERROR	X'080B0000'
LUA_BB_REJECT_NO_RTR	X'08130000'
LUA_RECEIVER_IN_TRANSMIT_MODE	X'081B0000'
LUA_CRYPTOGRAPHY_FUNCTION_INOP	X'08480000'
LUA_SYNC_EVENT_RESPONSE	X'10010000'
LUA_RU_DATA_ERROR	X'10020000'
LUA_RU_LENGTH_ERROR	X'10020000'
LUA_INCORRECT_SEQUENCE_NUMBER	X'20010000'
LUA_LCC_NOT_SUPPORTED	X'20010000'

lua_peek_data のバイト 4 からバイト 6 に戻される情報には、オリジナル要 a 単位の先頭の 3 バイトまでが } 容されます。

SLI_SEND

この verb は、LUA アプリケーション・プログラムから通信リンクへ、ユーザー・データ、 SNA コマンド、または SNA ~答を転送します。LU-LU セッション・フロー用の **SLI_SEND** は、前もってオープンされているセッション上でのみ / 行することができます。 **SLI_OPEN** 開始タイプが SSCP アクセスについては 1 ！であり、INIT_COMPLETE 状況が完了している場合は、アプリケーション・プログラムは、**SLI_SEND** を SSCP-LU 通常フローにデータを転送するために / 行することができます。

LUA アプリケーションは、それぞれ定 A されている LUA LU に対して、 2 つのアクティブ **SLI_SEND** verb を並行して持つことができます。この 2 つの verb は、任意の 2 つの別個のフローに対 ~ できます。

指定パラメーター

このアプリケーションは、以下のパラメーターを提供します。

lua_verb
LUA_VERB_SLI

LUA verb 用の verb コード標 1 。

lua_verb_length
verb 制御ブロックの長さ。この数値は、**SLI_SEND** verb 用の SLI により予期される長さと等しくなければなりません。

lua_opcode
LUA_OPCODE_SLI_SEND

この verb 用の操作コード。

lua_correlator
この verb を、プログラムが提供する他の情報に関連づけるために LUA アプリケーション・プログラムが提供できる値。 SLI はこのパラメーターを無視します。

lua_luname
ASCII 表 - のローカル LU 名。 名前が 8 文字未満であれば、ブランク文字を埋め込んで補う必要があります。 LUA は、 **lua_sid** が 0 の場合にのみこのパラメーターを検査します。すべての verb に **lua_luname** パラメーターを使用すると、デバッグが一層簡単になり、特に複数の LU 構成の場合に効果的です。

lua_sid
使用すべきセッションを 1 別する **SLI_OPEN** が戻すセッション ID。このパラメーターが 0 である場合は、**lua_luname** パラメーターは 1 別用に使用されます。

lua_data_length
送信するデータの長さ。

lua_data_ptr
ホスト・アプリケーションに送信すべきアプリケーション・プログラム・デ

ータを指すポインター。このバッファーは、データおよび SNA コマンドに使用されるため、バッファーの内容は通常は EBCDIC 表 (です。

lua_post_handle
s 同期 verb の完了を通知するのに使用する 4 バイトのハンドル。

lua_th.snf
RU のシーケンスV号。

lua_rh メッセージ用の SNA 要a / ~答ヘッダー (RH) を } 容する 3 バイトのパラメーター。

lua_message_type
SNA データおよびコマンドのタイプ。SLI アプリケーション・プログラムが、データを送信しようとする時は、このパラメーターの設定が必要です。SNA コマンドの詳細については、*Systems Network Architecture Network Product Formats* を参照してください。有効なメッセージ・タイプは以下のとおりです。

LUA_MESSAGE_TYPE_BID
LUA_MESSAGE_TYPE_BIS
LUA_MESSAGE_TYPE_CANCEL
LUA_MESSAGE_TYPE_CHASE
LUA_MESSAGE_TYPE_LU_DATA
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_RSP
LUA_MESSAGE_TYPE_RTR
LUA_MESSAGE_TYPE_SBI
LUA_MESSAGE_TYPE_SSCP_DATA

lua_flag1.sscp_exp
SSCP ^送フローを指定します

lua_flag1.sscp_norm
SSCP 通常フローを指定します

lua_flag1.lu_exp
LU ^送フローを指定します

lua_flag1.lu_norm
LU 通常フローを指定します

戻りパラメーター

verb が正常にB行された場合は、LUA は！のパラメーターを戻します。

lua_data_length
u 信したピーク・データの長さ。

lua_th メッセージ用の SNA 伝送ヘッダー (TH) を } 容する 6 バイトのパラメーター。

lua_flag2.async
この verb の s 同期完了を (すフラグ。

lua_flag2.sscp_exp
SSCP ^ 送フローを指定します。

lua_flag2.sscp_norm
SSCP 通常フローを指定します。

lua_flag2.lu_exp
LU ^ 送フローを指定します。

lua_flag2.lu_norm
LU 通常フローを指定します。

lua_sequence_number
SLI_SEND verb 用の先頭チェーンまたは単独チェーン RU のシーケンスV号。このシーケンスV号はバイトU転されていません。

lua_prim_rc
verb ！能が設定する 1 ！戻りコード。 詳細については、 369ページの『付録B. LUA Verb 戻りコード』 を参照してください。

lua_sec_rc
verb ！能が設定する 2 ！戻りコード。 詳細については、 369ページの『付録B. LUA Verb 戻りコード』 を参照してください。

使用上の注意

SLI_SEND は、特 l な処理たとえば、RH ビットと TH ビットおよびフロー・フラグの設定を、 **lua_message_type** パラメーターに基づいてB行します。たとえば、アプリケーションが **lua_message_type** パラメーターを X'84' (CHASE) に設定すると、SLI 構成要素は + 動的に **lua_rh** パラメーターを X'4B8000' に設定します。 288ページの表 17 は、必要な場合でかつプログラムの現在の状態が与えられた場合に、アプリケーション・プログラムが設定すべきパラメーターを (しています。

表 17. メッセージ・タイプに基づくパラメーターの設定値

	lua_message_type パラメーターの値						
SLI_SEND パラメーター	**LU_DATA SSCP_DATA**	**RSP**	**BID, BIS, RTR**	**CHASE QC**	**QEC, RELQ, SBI, SIG**	**RQR**	**LUSTAT_LU LUSTAT_SSCP**
lua_rh	FI, DR1I, DR2I, RI, BBI, EBI, CDI, CSI, EDI	RI	SDI, QRI	SDI, QRI, EBI, CDI	SDI	0	SDI, QRI, DR1I, DR2I, RI, BBI, EBI, CDI
lua_th	0	SNF	0	0	0	0	0

表 17. メッセージ・タイプに基づくパラメーターの設定値 (続き)

	lua_message_type パラメーターの値						
SLI_SEND パラメーター	**LU_DATA SSCP_DATA**	**RSP**	**BID, BIS, RTR**	**CHASE QC**	**QEC, RELQ, SBI, SIG**	**RQR**	**LUSTAT_LU LUSTAT_SSCP**
lua_data_ptr	Required (0 if no data)	Required (0 if no data)	0	0	0	0	必須
lua_data_length	必須	必須 (0 データがない場合)	0	0	0	0	必須
lua_flag1 フロー・フラグ	0	必須 (1 を設定)	0	0	0	0	0

SLI_SEND verb は、データを **lua_data_ptr** パラメーターが指定する位置から、**lua_data_length** で指定する長さだけ転送します。 SLI は、必要に ~ じてデータをチェーンします。 **SLI_SEND** は同期または s 同期に完了することができます。アプリケーション・プログラムが SLI の呼び出しから戻る時は、 **lua_flag2.async** フラグが verb の完了方法を指 (します。 **lua_flag2.async** が ON に設定されていると、IN_PROGRESS という 1 ！戻りコードは verb が u 信され、進行中であることを (します。 OK という 1 ！戻りコードは、データまたはコマンドが RUI に書き込まれたことを (します。 アプリケーション・プログラムは、 SLI 呼び出しからの同期戻りを行う **RUI_WRITE** を使用して送信される、最終チェーン・エレメントのシーケンス V 号を正常に u 信します。すべてのチェーン・エレメントが書き込まれると、アプリケーション・プログラムは、 TH (伝送ヘッダー) 中の最終戻りコードおよび最終シーケンス V 号を u 信します。SLI が、チェーン

SLI_SEND

SLI は、センス・データの内容に従って要 a コードを - 入します。

SLI_BIND_ROUTINE

この verb は、SLI アプリケーション・プログラムに SNA BIND 要 a がホストから到着したこと、および、そのアプリケーション・プログラムがセッション・プロトコルの検査を v 可されたことを知らせます。 **SLI_BIND_ROUTINE** は、 **SLI_OPEN** 拡張リストのバインド・ルーチン・フィールドで指定された、プログラマーが指定する DLL に渡されます。

指定パラメーター

SLI_BIND_ROUTINE に関する以下のパラメーターは、SLI が提供します。

lua_verb
LUA_VERB_SLI

LUA verb 用の verb コード標 1 。

lua_verb_length
verb 制御ブロックの長さ。

lua_opcode
LUA_OPCODE_SLI_BIND_ROUTINE

このルーチン用の操作コード。

lua_luname
ASCII 表 - のローカル LU 名。

lua_sid
使用すべきセッションを 1 別する **SLI_OPEN** が戻すセッション ID。

lua_data_length
BIND RU の長さ。

lua_data_ptr
BIND RU を指すポインター。BIND RU には EBCDIC 文字、たとえば PLU 名、が含まれる場合があります。

lua_th
BIND のTH (伝送ヘッダー)。

lua_rh
BIND のRH (要 a ヘッダー)。

戻りパラメーター

verb が正常に完了した場合は、LUA は ! のパラメーターを戻します。

lua_prim_rc
LUA_OK

lua_data_length
送信中の BIND ~答 の長さ。

lua_prim_rc
verb ！能が設定する 1 ！戻りコード。 詳細については、 369ページの『付録B. LUA Verb 戻りコード』 を参照してください。

使用上の注意

この verb 制御ブロックは、SLI が割り振りする - 憶域中に作成されます。 **lua_th** および **lua_rh** パラメーターの内容は、 **SLI_BIND_ROUTINE** verb 制御ブロック中に置かれます。**lua_data_ptr** パラメーターには、BIND RU のアドレスが、また **lua_data_length** パラメーターには、 RU の長さが } 容されています。

SLI_BIND_ROUTINE verb 制御ブロック中に **lua_prim_rc** および **lua_data_length** パラメーターが設定されている拡張ルーチンが戻されると、 **SLI_BIND_ROUTINE** は完了します。BIND ~答は、BIND RU を上書きします。OK という1 ！戻りコードは、 BIND が u け入れられたことを (します。ルーチンが BIND をリジェクトする場合は、 1 ！戻りコードを NEGATIVE_RSP に設定し、] 定センス・コードを BIND バッファーに書き込みます。 **lua_data_ptr** パラメーターを変更してはいけません。

注: このルーチンが出す] 定 ~ 答は、**SLI_OPEN** verb を h り消します。 SLI は、1 ！戻りコード SESSION_FAILURE および 2 ！戻りコード NEG_RSP_FROM_BIND_ROUTINE を戻します。

SLI_STSN_ROUTINE

この verb は、SLI アプリケーション・プログラムに SNA STSN 要 a がホストから到着したこと、および、そのアプリケーション・プログラムが STSN RU の検査および~答の作成を v 可されたことを知らせます。 **SLI_STSN_ROUTINE** は、 **SLI_OPEN** 拡張リストのバインド・ルーチン・フィールドで指定された、プログラマーが指定する DLL に渡されます。

指定パラメーター

SLI_STSN_ROUTINE に関する以下のパラメーターは、SLI が提供します。

lua_verb
LUA_VERB_SLI

LUA verb 用の verb コード標 1 。

lua_verb_length
verb 制御ブロックの長さ。

lua_opcode LUA_OPCODE_SLI_STSN_ROUTINE
このルーチン用の操作コード。

lua_luname
ASCII 表 - のローカル LU 名。

lua_sid
使用すべきセッションを 1 別する **SLI_OPEN** が戻すセッション ID。

lua_data_length
STSN RU の長さ。

lua_data_ptr
STSN RU を指すポインター。

lua_th
STSN のTH (伝送ヘッダー)。

lua_rh
STSN のRH (要 a ヘッダー)。

戻りパラメーター

verb が正常にB行された場合は、LUA は！のパラメーターを戻します。

lua_prim_rc
LUA_OK

lua_data_length
送信中の STSN ~答 の長さ。

lua_prim_rc
verb ！能が設定する 1 ！戻りコード。 詳細については、 369ページの『付録B. LUA Verb 戻りコード』 を参照してください。

使用上の注意

この verb 制御ブロックは、SLI が割り振りする - 憶域中に作成されます。 **lua_th** および **lua_rh** パラメーターの内容は、 **SLI_STSN_ROUTINE** verb 制御ブロック中に置かれます。 **lua_data_ptr** パラメーターには、**STSN** RU のアドレスが、また **lua_data_length** パラメーターには、RU の長さが } 容されています。

SLI_STSN_ROUTINE verb 制御ブロック中に **lua_prim_rc** および **lua_data_length** パラメーターが設定されている拡張ルーチンが戻されると、 **SLI_STSN_ROUTINE** は完了します。 STSN ~答は STSN RU を上書きします。OK という 1 ！戻りコードは、 STSN が u け入れられたことを (します。ルーチンが STSN をリジェクトする場合は、 1 ！戻りコードを NEGATIVE_RSP に設定し、] 定センス・コードを STSN バッファーに書き込みます。 **lua_data_ptr** パラメーターを変更してはいけません。

注: このルーチンが出す] 定 ~ 答は、**SLI_OPEN** verb を h り消します。 SLI は、1 ！戻りコード SESSION_FAILURE および 2 ！戻りコード NEG_RSP_FROM_STSN_ROUTINE を戻します。

SLI_SDT_ROUTINE

この verb は、SLI アプリケーション・プログラムに SNA SDT 要 a がホストから到着したこと、および、そのアプリケーション・プログラムが SDT RU の検査および ~ 答の作成を v 可されたことを知らせます。 **SLI_SDT_ROUTINE** は、 **SLI_OPEN** 拡張リストのバインド・ルーチン・フィールドで指定された、プログラマーが指定する DLL に渡されます。

SLI_SDT_ROUTINE は、SNA API クライアントではサポートされません。

指定パラメーター

SLI_SDT_ROUTINE に関する以下のパラメーターは、SLI が提供します。

lua_verb
LUA_VERB_SLI

LUA verb 用の verb コード標 1 。

lua_verb_length
verb 制御ブロックの長さ。

lua_opcode
LUA_OPCODE_SLI_STSN_ROUTINE

このルーチン用の操作コード。

lua_luname
ASCII 表 - のローカル LU 名。

lua_sid
使用すべきセッションを 1 別する **SLI_OPEN** が戻すセッション ID。

lua_data_length
SDT RU の長さ。

lua_data_ptr
SDT RU を指すポインター。

lua_th
SDT のTH (伝送ヘッダー)。

lua_rh
SDT のRH (要 a ヘッダー)。

戻りパラメーター

拡張ルーチンが戻す **SLI_SDT_ROUTINE** 用のパラメーターのリストを以下に (します。

lua_prim_rc
LUA_OK

lua_data_length
送信中の SDT ~ 答 の長さ。

lua_prim_rc

verb ！能が設定する 1 ！戻りコード。 詳細については、 369ページの『付録B. LUA Verb 戻りコード』 を参照してください。

使用上の注意

この verb 制御ブロックは、SLI が割り振りする - 億域中に作成されます。 **lua_th** および **lua_rh** パラメーターの内容は、 **SLI_SDT_ROUTINE** verb 制御ブロック中に置かれます。 **lua_data_ptr** パラメーターには、SDT RU のアドレスが、また **lua_data_length** パラメーターには、RU の長さが } 容されています。

SLI_SDT_ROUTINE verb 制御ブロック中に **lua_prim_rc** および **lua_data_length** パラメーターが設定されている拡張ルーチンが戻されると、 **SLI_SDT_ROUTINE** は完了します。 SDT ~答は SDT RU を上書きします。 OK という 1 ！戻りコードは、SDT が u け入れられたことを (します。ルーチンが SDT をリジェクトする場合は、1 ！戻りコードを NEGATIVE_RSP に設定し、] 定センス・コードを STSN バッファーに書き込みます。 **lua_data_ptr** パラメーターを変更してはいけません。

注: このルーチンが出す] 定 ~ 答は、**SLI_OPEN** verb を h り消します。 SLI は、1 ！戻りコード SESSION_FAILURE および 2 ！戻りコード NEG_RSP_FROM_SDT_ROUTINE を戻します。

第3部 共通サービス API

第16章 共通サービス・エントリー・ポイント

パーソナル・コミュニケーションズおよび 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、パーソナル・コミュニケーションズおよび 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。

どのパーソナル・コミュニケーションズおよび Communications Serverアプリケーション・プログラムも、これらの共通サービス verb を使って、以下の作業の 1 つまたは複数個を行うことができます。

- 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**)。

注: 本書第 3 部の諸章には、以下のシステムが提供する共通サービス API に関する情報が含まれています。

- Windows NT 上でB行される Communications Server
- 通信サーバー/NT 製品と一緒に納入される OS/2、Windows NT、Windows 95、および Windows 3.1 用の SNA API クライアント
- Windows 95 および Windows NT 用の パーソナル・コミュニケーションズ

これらのシステムが提供するサポート間で相違がある場合は、注 - します。

共通サービス・プログラムの作成

以下の表では、提供されたヘッダー・ファイルのソース・モジュール使用法と、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ライブラリーを (しています。

表 18. オペレーティング・システムのヘッダー・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

*WINNT = Windows NT
*WIN95 = Windows 95
*WIN3.1 = Windows 3.1

ここでは、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、すべての CSV verb 用の同期エントリー・ポイントです。 パーソナル・コミュニケーションズおよび 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 ！戻りコードおよび 2 ！戻りコードを調べてください。

これは SNA API OS/2 クライアント用にサポートされる唯一の Communications Server エントリー・ポイントです。

WinCSV()

このファンクションは、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 (します。

注: 303ページの『WinAsyncCSV()』 ページの **WinAsyncCSV()** も参照してください。

WinCSVCleanup()

このファンクションは、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 h り消します。

構文

```
BOOL WINAPI WinCSVCleanup(void);
```

戻り値

戻り値は、登録の h り消しが成功したかどうかを (します。 値が 0 以外であれば、パーソナル・コミュニケーションズおよび Communications Serverはアプリケーションの登録を正常に h り消しています。値が 0 の場合は、パーソナル・コミュニケーションズおよび Communications Server はアプリケーションの登録を h り消しています。

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 h 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。

WinAsyncCSV()

このファンクションは、**TRANSFER_MS_DATA** 専用の s 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 にこのファンクションを使用しても、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 u け h るウィンドウ・ハンドル。

vcb verb 制御ブロックを指すポインター。

戻り値

戻り値は、verb 要 a が正常に行われたかどうかを (します。このファンクションが成功した場合は、B 際の戻り値は s 同期タスク・ハンドルです。 ファンクションが成功しなかった場合は、パーソナル・コミュニケーションズおよび Communications Serverは `0` を戻します。

使用上の注意

s 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して、 **RegisterWindowMessage** が戻すメッセージを u け h ります。 *wParam* 引き数には、元のファンクション・コールから戻された s 同期のタスク・ハンドルが入っています。 *lParam* 引き数には元の 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 = 別できます。

このファンクションが正常に戻った場合は、パーソナル・コミュニケーションズおよび Communications Serverは、操作が完了したとき、または会話が h 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()

このファンクションを使用すると、アプリケーションは要 a 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。この呼び出しは必須ではありませんが、これを使用する場合は、**WinCSVCleanup** 呼び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 a 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V号（改訂V号）を (し、低位バイトはバージョンV号を (します。

lpwCSVDATA

基礎となる 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V号をサポートするかどうかを (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、！のいずれかの値が戻されます。

WCSVVERNOTSUPPORTED

この CSV API は、要 a 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 a されているバージョンを = 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。

！の構造は、B際の CSV API ！能の詳細を (しています。

```
typedef struct tagWCSVDATA { WORD wVersion;
                char szDescription[WCSVDESCRIPTION_LEN+l];
                } WCSVDATA, FAR *LPWCSVDATA;
```

アプリケーションは、最後の CSV API 呼び出しの後で、**WinCSVCleanup()** を呼び出します。

GetCsvReturnCode()

このエントリー・ポイントを使用して、verb 内の 1 ！戻りコードおよび 2 ！戻りコードを、印刷可能ストリングに変換します。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 (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 ? 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。このファンクションは、指定した **verb** からの読み h り、または指定したバッファーへの書き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

このファンクションは、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。パーソナル・コミュニケーションズおよび Communications Serverは、**TRNSDT.DLL** ファイル **TrnsDt** を提供します。**TransDt** は DBCS セッションでのみ使用できます。

構文

```
TrnsDt (PASSSTRUCT *passparm);
```

ファンクション

このファンクションは、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。以下の表では、『✓』 は、パーソナル・コミュニケーションズおよび Communications Serverがコード・ページの対の間の変換をサポートしていることを、 “- ”（ハイフン）は、どちらのプログラムもその変換をサポートしていないことを (しています。

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、リンクにはどちらかのプログラムの 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。

戻りコード

passparm A 0 は！のとおりです。

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！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！のものがあります。

4 DBCS で終了

0 s DBCS で終了

注:

1. ビット 8 およびビット 0 は！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** はデフォルト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L 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 1 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。

戻りコード

0 NO_ERROR

2 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

メモリー割り振りエラー。

使用上の注意

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 , 模なデータ変換を h 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、 u 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 k 果に ~ じて操作を進めます。

たとえば、変換の k 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 A してから、 ! の呼び出しを行います。

VCB 構造

! 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     = Created by ActiveSystems. 02/11/97. Entity not defined[0];
passaparm.out_length  = 20;
passaparm.out_addr    = Created by ActiveSystems. 02/11/97. Entity not defined[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17章 共通サービス verb (CSV)

パーソナル・コミュニケーションズ および Communications Server は、共通サービス API 用として！の verb を提供しています。

GET_CP_CONVERT_TABLE
CONVERT

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。

```
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;
```

source_code_page

置換文字が抽出されるコード・ページV号。 コード・ページのV号は、以下の数字のいずれかです。

- ASCII コード・ページ（10 進数）
  - 437 米国 (US) IBM PC
  - 737 ギリシャ
  - 813 ギリシャ
  - 819 ANSI (米国 , 格協会) 標準
  - 850 マルチリンガル
  - 852 チェコ／スロバキア／ハンガリー／ポーランド／ l ユーゴスラビア
  - 855 キリル語
  - 857 トルコ
  - 858 マルチリンガル
  - 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 北欧ゲルマンO言語
  - 866 キリル語
  - 874 タイ
  - 912 ラテン語 2
  - 915 キリル語
  - 916 ヘブライ語

  - 920 トルコ
  - 921 ラトビア、リトアニア
  - 922 エストニア
  - 923 ANSI (米国 , 格協会) 標準
  - 1008 アラビア語
  - 1089 アラビア語
  - 1124 ウクライナ
  - 1125 ウクライナ
  - 1127 アラビア語/フランス語
  - 1129 ベトナム
  - 1131 ベラルーシ
  - 1133 ラオ語
  - 1250 ラテン語 2
  - 1251 キリル語
  - 1252 ラテン語 1
  - 1253 ギリシャ
  - 1254 トルコ
  - 1255 ヘブライ語
  - 1256 アラビア語
  - 1257 バルト語 (ラトビア、リトアニア、エストニア)
  - 1258 ベトナム
- 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
  - 275 ブラジル
  - 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 Q国
  - 297 フランス
  - 420 アラビア語
  - 424 ヘブライ語
  - 500 ベルギー／スイス・フランス語／スイス・ドイツ語
  - 803 ヘブライ語
  - 870 チェコ／スロバキア／ハンガリー／ポーランド／ l ユーゴスラビア
  - 871 アイスランド
  - 875 ギリシャ
  - 924 ラテン語 1
  - 1025 キリル語

- 1026 トルコ
- 1047 ラテン語 1
- 1112 ラトビア、リトアニア
- 1122 エストニア
- 1123 ウクライナ
- 1130 ベトナム
- 1132 ラオ語
- 1140 米国/カナダ/オランダ/ポルトガル/ブラジル/オーストラリア/ニュージーランド
- 1141 ドイツ/オーストリア
- 1142 デンマーク/ノルウェー
- 1143 フィンランド/スウェーデン
- 1144 イタリア
- 1145 ラテン・アメリカ／スペイン
- 1146 Q国
- 1147 フランス
- 1148 ベルギー/スイス
- 1149 アイスランド

• ユーザー定Aコード・ページ
  - 65280 〜 65534
  - ユーザー定Aコード・ページを使用するときは、まず、以下のように パーソナル・コミュニケーションズ 用に、ユーザーが定Aした CPT ファイルへのパスを使用してレジストリー・エントリーを定Aする必要があります。

    ```
    HKEY_LOCAL_MACHINE/SOFTWARE/IBM/Personal
    Communications /CurrentVersion/COMCPT
    ```

    Communications Server 用には、ユーザーが定Aした CPT ファイルへのパスを使用して、以下のようにレジストリー・エントリーを定A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W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V号。このV号は、**source_code_page** の項に (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u け入れるバッファーのアドレス。このバッファーは、読みh り/書き込みセグメント内に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。以下のいずれかの値を指定します。

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？転するA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U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B行されたことを (します。

戻りコードが OK のときは、！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 に、そして EBCDIC 文字ストリングを ASCII に変換します。

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。

```
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;
```

return_code OK エラー・コード

指示 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 v されている文字セット。 **CONVERT** verb で使用するために、SV_A、SV_AE、および SV_G の 3 つのタイプの ASCII/EBCDIC 変換テーブルを指定することができます。タイプ A とタイプ AE テーブルはパーソナル・コミュニケーションズおよび Communications Server内で定 A されています。

変換テーブルのフォーマットは、それぞれ 32 文字の行 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 " 改行文字から成っています。 前 > の 16 行は、 ASCII から EBCDIC への変換のための情報を提供します。 後 > 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 行のすべてが含まれていることが必要です。

変換を行うときに、パーソナル・コミュニケーションズおよび 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 / 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、テーブル中の 48 V 目の位置には、X'F0' の値が入っ

ていると仮定します。 この場合、パーソナル・コミュニケーションズおよび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 l 文字 $、#、@ を変換します。ソース・ストリングの最初の 1 文字は、Q字の大文字か、または 3 つの特 l 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 ！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 文字に変換されます。

後書きブランク（ソース・ストリングの末 x 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。対照的に、組み込みブランクは 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Q数字 (A から Z、a から z、0 から 9)、特 l 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 x 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。対照的に、組み込みブランクは 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（ASCII 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 > を使用するには **CONVERT** verb で ASCII_TO_EBCDIC を指定し、後 > を使用するには EBCDIC_TO_ASCII を指定する必要があります。

パーソナル・コミュニケーションズは！のレジストリーを調べます。

```
HKEY_LOCAL_MACHINE/SOFTWARE/IBM/Personal Communications /
                                  CurrentVersion/COMTBLG
```

この項目から、G テーブルの完全パス名を入 j します。 Communications Server は！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 j します。 32-bit Windows NT クライアントの場合は、レジストリー中のテーブル G のパスの位置は、以下のとおりです。

```
HKEY_LOCAL_MACHINE/SOFTWARE/IBM/Comm.Server for NT SNA/Client/
                                  CurrentVersion/COMTBLG
```

CONVERT verb のその他の指定パラメーターには、！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 u け h 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。

OK 戻りコードは、**CONVERT** verb が正常に B 行されたことを (します。

！に、**CONVERT** verb に関連した 1 ！および 2 ！エラー戻りコードと、戻りコードの説明がある場所を (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18章 EHNAPPC アプリケーション・プログラム・インターフェース

これは、Communications Server SNA API クライアントだけで使用することができます。

EHNAPPC 通信 API は、パーソナル・コンピューターと AS/400 システムとの間の連H処理アプリケーションを作成するj段を提供します。これにより、プログラマーは低水準の通信プログラミングやハードウェア接続性タイプといった問題から解放されます。アプリケーション・プログラマーは、この API を使うとき、 AS/400 と PC プログラムの両方を書く必要があります。ホスト・アプリケーションがアクセスできるものはほとんどすべて、パートナー PC アプリケーションに拡張することができます。この API は、パフォーマンス要oの厳しい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 の Communications Server SNA API クライアント用の 32 ビット EHNAPPC API を構成するルーチン、データ構造、および戻りコードを説明しています。これらの！能のほとんどは、Windows 3.1 の 16 ビット API でも利用できます。

EHNAPPC プログラムの作成

以下の表では、提供されたヘッダー・ファイルのソース・モジュール使用法と、EHNAPPC プログラムをコンパイルしリンクするのに必要なライブラリーを(している ます。

表 19. オペレーティング・システムのヘッダー・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

*WINNT = Windows NT
*WIN95 = Windows 95
*WIN3.1 = Windows 3.1

EHNAPPC ルーチン

各クライアント・ウィンドウ APIﾁﾌﾞﾊﾞﾄﾞ 、！な が
ます。

- 戻りコード

EHNAPPC_Allocate

目的

この関数は、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。

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 1 別します。

bType は、割り当てる会話のタイプを 1 別します。可能な値は！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・プログラムとの間の同期レベルを 1 別します。可能な値は！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 で終了する文字列を指します。このポインターが NULL に設定される場合、デフォルト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 で終了する文字列を指します。最初の文字が 0x40 より小さい場合、 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 1 別します。この変数が 0 の場合、送信される PIP データはありません。

lpPipData は PIP データを指します。 PIP データは GDS フォーマットおよび EBCDIC A 0 でなければなりません。

lpdwConversation は、後続の呼び出し時に使用するハンドルを戻すために使用されるダブルワード変数を指します。ハンドルは、それぞれの会話に固有な値です。

戻りコード

戻りコードについては、340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

この関数は、これまでに送信されたすべてのデータがパートナーによって u 信されたことを確認するよう要 a 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 1 別します。

dwConversation は、EHNAPPC_Allocate または EHNAPPC_ExtendedAllocate によって戻される会話ハンドルを 1 別します。

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 を出したかどうかを保管するのに使用される変数を指します。 TRUE 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 を出したことを (しています。

戻りコード

戻りコードについては、340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

この関数は、確認を要 a したパートナーに ~ 答して、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 1 別します。

dwConversation は、EHNAPPC_Allocate または EHNAPPC_ExtendedAllocate によって戻される会話ハンドルを 1 別します。

戻りコード

戻りコードについては、340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

この関数は、割り当てられた会話を割り当て解除します。

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 1 別します。

dwConversation は、EHNAPPC_Allocate または EHNAPPC_ExtendedAllocate によって戻される会話ハンドルを 1 別します。

bType は、クライアントがB行する割り当て解除のタイプを 1 別します。可能な値は！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、340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

この関数は、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、セキュリティーまたはモードの指定をオーバーライドします。

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 1 別します。

bType は、割り当てる会話のタイプを 1 別します。可能な値は！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・プログラムとの間の同期レベルを 1 別します。可能な値は！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 で終了する文字列を指します。このポインターが NULL に設定される場合、デフォルト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 で終了する文字列を指します。最初の文字が X'40' より小さい場合、 ASCII から EBCDIC への変換は行われません。

lpszModeName。モードの命名 , 則を以下に (します。

モード名 は、1〜8 文字の長さです。それぞれの部分の最初の文字は大文字のQ字 (A〜Z) か、特 l 文字 (@、#、$) です。残りの文字は大文字のQ字 (A〜Z)、数字 (0〜9)、または特 l 文字 (@、#、$) です。

bSecurityType は、使用するセキュリティー・タイプを 1 別します。可能な値は！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 で終了する文字列を指します。最大長は 10 文字です。

lpszPassword は、パスワードを含む、NULL で終了する文字列を指します。最大長は 10 文字です。

nPipLength は、PIP データの長さを 1 別します。この変数が 0 の場合、送信される PIP データはありません。

lpPipData は PIP データを指します。 PIP データは GDS フォーマットおよび EBCDIC A 0 でなければなりません。

lpdwConversation は、後続の呼び出し時に使用するハンドルを戻すために使用される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、340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

この関数により、クライアントはバッファー内にあるすべてのデータを送信します。

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 1 別します。

dwConversation は、EHNAPPC_Allocate または EHNAPPC_ExtendedAllocate によって戻される会話ハンドルを 1 別します。

戻りコード

戻りコードについては、340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。これには、ローカルおよびパートナー・トランザクション・プログラムの LU 名、同期処理のレベル、およびセキュリティーのために指定されるユーザー 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 1 別します。

dwConversation は、EHNAPPC_Allocate または EHNAPPC_Extended 割り当てによって戻される会話ハンドルを 1 別します。

lpbSyncLevel は、同期レベルを戻すために使用されるバイト変数を指します。

lpszModeName は、8 文字のモード名を戻すために使用される、 NULL で終了する文字列を指します。

lpszLuName は、ローカル・トランザクション・アクション・プログラムの LU を戻すために使用される、 NULL で終了する文字列を指します。

lpszPluName は、パートナー LU の名前を戻すために使用される、 NULL で終了する文字列を指します。

lpszUserId は、この接続を確立するために使用されるユーザー ID を戻すのに使用される、NULL で終了する文字列を指します。

戻りコード

戻りコードについては、340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

この関数は、現在ロードされているクライアントの能力を (すデータ構造に、データを入れます。

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 1 別します。

lpList は、能力情報を検索するために使用される！能リストを指します。！能リストは、ヘッダーと、その後に続く可変数の！能構造から成っています。入力時には、照会する！能をリストで指定します。出力時には、リストには！能情報が入っています。

注: 付加的な構造情報については、338ページの『appcrtrcap_hdr』、339ページの『appcrtrcap_mult』、および 339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、340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

この関数は、クライアントが接続されるデフォルト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 1 別します。

lpszDefSysName は、デフォルト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、NULL で終了する文字列としてこのバッファーに保管されます。

戻りコード

戻りコードについては、340ページの『EHNAPPC API の戻りコード』を参照してください。

EHNAPPC_IsRouterLoaded

目的

この関数は、クライアント・ルーターがメモリーにロードされているかどうかを = 別します。

プロシージャー宣言

```
#include <WINDOWS.H>
#include "E32APPC.H"
extern bool EHNAPPC_IsRouterLoaded(
     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 1 別します。

戻りコード

Communications Server SNA クライアント・ルーターがロードされていない場合、戻りコードは FALSE (0) です。 その他の場合は、戻り値は TRUE (1) です。

EHNAPPC_PrepareToReceive

目的

この関数は、データを u 信するプログラムを用意します。この関数に続けて EHNAPPC_ReceiveImmediate を使うと、EHNAPPC_ReceiveAndWait を使った場合と同じになります。

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 1 別します。

dwConversation は、EHNAPPC_Allocate または EHNAPPC_ExtendedAllocate によって戻される会話ハンドルを 1 別します。

戻りコード

戻りコードについては、340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

この関数は、通信サーバー上で構成され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 1 別します。

lpSysCount は、接続されたシステムの数を戻すために使用される整変数を指します。

lpSys は、システムの名前を戻すために使用される AS400_Sys 構造を指します。デフォルト・システムは、構造の中の最初のシステムです。 AS400_Sys 構造の説明については、338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、340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

この関数は、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 1 別します。

dwConversation は、EHNAPPC_Allocate または EHNAPPC_ExtendedAllocate によって戻される会話ハンドルを 1 別します。

戻りコード

戻り値は、会話の現在の状態を (しています。可能な値は！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

この関数は、クライアントが接続され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 1 別します。

lpSysCount は、接続されたシステムの数を戻すために使用される整変数を指します。

lpSys は、システムの名前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、340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

この関数は、指定されたシステムに接続されるユーザー 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 118 27 0 TD 0 Tc (d 0 Tc6 /F118 H7 0 TD * /F106 1 Tf 6 /F118TD (k)T

hWnd は、アプリケーションの現行ウィンドウを 1 別します。

lpszLocationName は、照会するシステム名を含む、 NULL で終了する文字列を指
します ルヨルコ ヤK ピャエジョピリピリクテケサゴ ケ ピユケギゴスゴサコジャエ
ジ NULL文 を L P S T R l 1 6 8 . 3 9 9 9 F 1 1 8 ,

EHNAPPC_QuerySystems

目的

この関数は、クライアントが接続され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 1 別します。

lpSysCount は、接続されたシステムの数を戻すために使用される整変数を指します。

lpSys は、システムの名前を戻すために使用される AS400_Sys 構造を指します。デフォルト・システムは、構造の中の最初のシステムです。 AS400_Sys 構造の説明については、338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、340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

この関数は、会話時に到着する情報を待！してから、情報を u 信します。

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 1 別します。

dwConversation は、EHNAPPC_Allocate または EHNAPPC_ExtendedAllocate によって戻される会話ハンドルを 1 別します。

bFill は、プログラムがデータを u 信する際の様 0 を (しています。可能な値は ! のとおりです。

EHNAPPC_BUFFER (0)（バッファーを満たす）

EHNAPPC_LL (1)（完全なまたは切り N てられた論理レコードを u 信する）

nMaxLength は、 u け入れることができる最大データ量を (しています。

lpReceiveData は、データを u 信するバッファーを指しています。

lpWhatReceived は、クライアントが u 信したものを (しています。可能な値は ! 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 を出したかどうかを保管するために使用される変数を指します。 TRUE (1) 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 を出したことを (しています。

lpReceiveDataLength は、クライアントが u 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、340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

この関数は、 u 信したものがあるかどうかを調べます。あるなら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 1 別します。

dwConversation は、EHNAPPC_Allocate または EHNAPPC_ExtendedAllocate によって戻される会話ハンドルを 1 別します。

bFill は、プログラムがデータを u 信する際の様 0 を (しています。可能な値は ! のとおりです。

EHNAPPC_BUFFER (0)（バッファーを満たす）

EHNAPPC_LL (1)（完全なまたは切り N てられた論理レコードを u 信する）

nMaxLength は、 u け入れることができる最大データ量を (しています。

lpReceiveData は、データを u 信するバッファーを指しています。

lpWhatReceived は、クライアントが u 信したものを 1 別します。可能な値は ! 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 を出したかどうかを保管するのに使用される変数を指します。 TRUE (1) 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 を出したことを (しています。

lpReceiveDataLength は、クライアントが u 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、340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

この関数により、Windows アプリケーションはリモート AS/400 システム上でプログラムを / 動することができます。

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 1 別します。

lpszHostSystemName は、リモート・システムの名前が含まれている、 NULL で終了する文字列を指します。この文字列の最大長は 8 文字です。このポインターが NULL の場合、デフォルトのシステム名が使用されます。

lpszHostProgramName は、 / 動されるホスト・プログラムの名前が含まれている、NULL で終了する文字列を指します。

lpszHostLibraryName は、ホスト・プログラムのライブラリー・パスが含まれている、NULL で終了する文字列を指します。このポインターが NULL の場合、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、送信される PIP データはありません。

wPipDataLength には、PIP データの長さが含まれています。

戻りコード

戻りコードについては、340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

この関数は、会話の制御をパートナーが放棄するよう要 a します。ローカル・トランザクション・プログラムが、続いてパートナー・トランザクション・プログラムから、 u 信 verb の lpWhatReceived パラメーターに入っている EHNAPPC_RECEIVEDSEND (6) を u 信すると、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 1 別します。

dwConversation は、EHNAPPC_Allocate または EHNAPPC_ExtendedAllocate によって戻される会話ハンドルを 1 別します。

戻りコード

戻りコードについては、340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

この関数は、パートナー・トランザクション・プログラムにデータを送信します。

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 1 別します。

dwConversation は、EHNAPPC_Allocate または EHNAPPC_ExtendedAllocate によって戻される会話ハンドルを 1 別します。

nSendDataLength は、送信バッファーのデータの長さを 1 別します。

lpSendDataBuffer は、送信バッファーのアドレスを 1 別します。

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 を出したかどうかを保管するために使用される変数を指します。 TRUE 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 を出したことを (しています。

戻りコード

戻りコードについては、340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

この関数は、何らかのエラーが見つかったことをパートナー・トランザクション・プログラムに知らせます。この関数を使用した後に、ローカル・プログラムは u 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 1 別します。

dwConversation は、EHNAPPC_Allocate または EHNAPPC_ExtendedAllocate によって戻される会話ハンドルを 1 別します。

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 を出したかどうかを保管するために使用される変数を指します。 TRUE 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 を出したことを (しています。

戻りコード

戻りコードについては、340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

この関数により、Windows アプリケーションはリモート AS/400 システム上でプログラムを / 動することができます。その際、会話をアクティブのままにしておくことによって、アプリケーションはホスト・プログラムが B 行中であることを確認できます。アプリケーションは EHNAPPC_Deallocate 関数を使って会話を終了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 1 別します。

lpszHostSystemName は、リモート・システムの名前が含まれている、 NULL で終了する文字列を指します。この文字列の最大長は 8 文字です。このポインターが、ト

lpszHostProgramName は、 / 動されるホスト・プログラムの名前が含まれている、NULL で終了する文字列を指します。

lpszHostLibraryName は、ホスト・プログラムのライブラリー・パスが含まれている、NULL で終了する文字列を指します。このポインターが NULL の場合、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、送信される PIP データはありません。

wPipDataLength には、PIP データの長さが含まれています。

戻りコード

戻りコードについては、340ページの『EHNAPPC API の戻りコード』を参照してください。

EHNAPPC 構造体

AS400_SYS

目的

この構造体を使って、クライアントが接続されるシステムの名前を保管します。

プロシージャー宣言

```
struct AS400_sys
     (
unsigned char EHNAPPC_SysName$EHNAPPC_MAX_SYSTEMS|
     $EHNAPPC_SYSNAME_SYSNAME_LENGTH|;
     );
```

パラメーター

EHNAPPC_SysName を使って、接続されたシステムの名前を保管します。システム名は、NULL で終了する文字列として戻されます。配列に戻される最初のシステムはデフォルト・システムです
(EHNAPPC_MAX_SYSTEMS = 32 および EHNAPPC_SYSNAME_SYSNAME_LENGTH = 10)。

appcrtrcap_hdr

目的

これは、クライアント！能リスト・ヘッダーの構造体です。

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 を使って、！能要 a の全体的な戻りコードを保管します。

opcode は、！能要 a h 得のシグナルです。その値は、 EHNAPPC_OC_CAPABILITIES (0x17) でなければなりません。

length は、！能リスト全体の長さを 1 別します。その長さは、ヘッダーのサイズと、それぞれの！能構造体のサイズを足した長さです。

appcrtrcap_mult

目的

これは、最適通信バッファー乗数を = 別するために使用される！能構造体です。

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！能構造体の長さを (しています。

identifier は、最適通信バッファー乗数のシグナルです。その値は、EHNAPPC_CAP_OPTIMAL_COM_SIZE (X'02') でなければなりません。

rc を使って、この！能要 a の戻りコードを保管します。

data を使って、最適通信バッファー乗数を戻します。

appcrtrcap_query

目的

これは、指定された！能をクライアントがサポートしているかどうかを照会するために使用される！能構造体です。

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！能構造体の長さです。

identifier は、照会する関数を (しています。可能な値は！のとおりです。

EHNAPPC_CAP_QUERY_CONV_STATE (3)

EHNAPPC_CAP_EXT_ALLOCATE (4)

rc を使って、この！能要 a の戻りコードを保管します。

data を使って、指定された関数がサポートされているかどうかを戻します。

EHNAPPC API の戻りコード

クライアント・ウィンドウ API の関数では、E32APPC.H で定 A される以下の戻りコード定数を使用します。

表 20. 戻りコード

戻りコード	16 進値	説明
EHNAPPC_OK	0	コマンドが正常に完了した
ENHAPPC_DEALLOCNORMAL	1	正常に割り当て解除された
ENHAPPC_PROGRAMMERNOTRUNCATION	2	プログラム・エラー、切り N てなし
ENHAPPC_PROGRAMMERTRUNCATION	3	プログラム・エラー、切り N て
ENHAPPC_PROGRAMMERPURGING	4	プログラム・エラー、消 n
ENHAPPC_RESOURCEFAILURETRY	5	リソース障害、再試行
ENHAPPC_RESOURCEFAILURENORETRY	6	リソース障害、再試行なし
ENHAPPC_UNSUCCESSFUL	7	：敗
ENHAPPC_APPCBUSY	8	APPC 使用中
ENHAPPC_PARMCHKINVALIDVERB	14	パラメーター・チェック、間違った verb
ENHAPPC_PARMCHKINVALIDCONVERID	15	パラメーター・チェック、間違った会話 ID
ENHAPPC_PARMCHKBUFFERCROSSEG	16	パラメーター・チェック、バッファーにまたがったセグメント
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、間違った会話タイプ
ENHAPPC_PARMCHKBADSYNCLVLALLOC	19	パラメーター・チェック、不正な同期レベル割り当て
ENHAPPC_PARMCHKBADRETURNCNTRL	1A	パラメーター・チェック、不正な戻り制御

表 20. 戻りコード (続き)

戻りコード	16 進値	説明
ENHAPPC_ ENHAPPC_PARMCHKPIPTOOLONG	1B	パラメーター・チェック、PIP データが長すぎる
ENHAPPC_PARMCHKBADPARTNERNAME	1C	パラメーター・チェック、不正なパートナー名
ENHAPPC_PARMCHKCONFNOTALLOWED	1D	パラメーター・チェック、確認が v 可されていない
ENHAPPC_PARMCHKBADDEALLOCTYPE	1E	パラメーター・チェック、不正な割り当て解除タイプ
ENHAPPC_PARMCHKPREPTORCVTYPE	1F	パラメーター・チェック、 u 信準 w のタイプ
ENHAPPC_PARMCHKBADFILLTYPE	20	パラメーター・チェック、不正な入力タイプ
ENHAPPC_PARMCHKRECMAXLEN	21	パラメーター・チェック、 u 信最大長
ENHAPPC_PARMCHKUNKNOWNSECTYPE	22	パラメーター・チェック、予約フィールドがゼロでない
ENHAPPC_PARMCHKRESFLDNOTZERO	23	パラメーター・チェック、予約フィールドがゼロでない
ENHAPPC_STATECHKNOTINCONFSTAT	28	状態チェック、確認状態にない
ENHAPPC_STATECHKNOTINRECEIVE	29	状態チェック、 u 信状態にない
ENAHAPPC_STATECHKREQSNDBADSTATN	2A	状態チェック、不正な状態を送信させる要 a
ENHAPPC_STATECHKSNDINBADSTATE	2B	状態チェック、不正な状態での送信
ENHAPPC_STATECHKSNDERRBADSTAT	2C	状態チェック、送信エラーの不正な状態
ENHAPPC_ALLOCERRNORETRY	32	割り振りエラー、再試行なし
ENHAPPC_ALLOCERRRETRY	33	割り振りエラー、再試行
ENHAPPC_ALLOCERROGMNOTAVAILNR	34	割り振りエラー、プログラムを利用できず、再試行なし
ENHAPPC_ALLOCERRTPNNOTRECOG	35	割り振りエラー、トランザクション・プログラム名が認 1 されない
ENHAPPC_ALLOCERRPGMNOTAVAILR	36	割り振りエラー、プログラムを利用できず、再試行
ENHAPPC_ALLOCERRSECNOTVALID	37	割り振りエラー、セキュリティーが無効
ENHAPPC_ALLOCERRCONVTYP	38	割り振りエラー、会話タイプが不適合
ENHAPPC_ALLOCERRPIPNOTALLOWED	39	割り振りエラー、PIP データが v 可されていない
ENHAPPC_ALLOCERRPIPNOTCORRECT	3A	割り振りエラー、PIP データが正しくない
ENHAPPC_ALLOCERRSYNCHLEVEL	3B	割り振りエラー、同期レベルがサポートされていない
ENHAPPC_DEALLOCABENDPROGRAM	46	割り当て解除でのプログラム異常終了
ENHAPPC_INSUFFICIENTMEMORY	47	メモリーが不十分
ENHAPPC_MEMORYALLOCERROR	47	メモリー割り振りエラー
ENHAPPC_MEMORYALLCERROR	48	メモリー割り振りエラー
ENHAPPC_TOOMANYCONVERSATIONS	4A	会話が多すぎる
ENHAPPC_CONVTABLEFULL	4B	変換テーブルが満杯

表 20. 戻りコード (続き)

戻りコード	16 進値	説明
ENHAPPC_CLIENTNOTINSTALLED	4C	クライアントが未導入
ENHAPPC_CLIENTWRONGLEVEL	4C	クライアントが間違ったレベルにある
ENHAPPC_PCSWINNOTLOADED	4D	PSWIN がロードされていない
ENHAPPC_PCSWINOUTOFMEMORY	4E	PCSWIN がメモリーを使い果たした
ENHAPPC_INVALIDUSERIDLEN	4F	間違ったユーザー ID 長
ENHAPPC_INVALIDPASSWORDLEN	50	間違ったパスワード長
ENHAPPC_INVALIDUNAME	51	間違った LU 長
ENHAPPC_UNDEFINED	63	未定 A

Windows 95 および Windows NT 上での 16 ビット EHNAPPC プログラムの B 行

Communications Server SNA API の Windows 95 および Windows NT クライアントには、 Windows 95 および Windows NT 上で既存の 16 ビット EHNAPPC プログラムを B 行する！能があります。それを B 行するには、16 ビット EHNAPPC アプリケーションのいずれかを開始する前に、 Communications Server SNA API クライアント・サブディレクトリーからプログラム EHNAPPCD を開始します。このプログラムには、32 ビット E32APPC.DLL へ必要とされるサンクが含まれています。

第19章 データ変換 Windows アプリケーション・プログラム・インターフェース

これは、Communications Server SNA API クライアントだけで使用することができます。

データ変換 API には、AS/400 と PC フォーマット間でデータを変換する！能があります。 AS/400 との間でデータを送 u 信するときに変換が必要になることがあります。データ変換 API は、テキストおよび各 o の数値フォーマットの変換をサポートしています。

この章では、データ変換 API を構成する個々のルーチンと戻りコードを説明します。

データ変換 Windows API ルーチン

各データ変換 API ルーチンについて、！のような詳細が説明されています。

- 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

この関数は、Windows ANSI コード・ページから EBCDIC に文字列を変換します。このルーチンが ASCII/EBCDIC 間変換テーブルにアクセスできるようにするため、ルーターをロードする必要があります。

ターゲット・ストリングが、変換後のストリングが入る十分な大きさでない場合、変換はターゲット・ストリングの終わりで停止します。ターゲット・ストリングが必要以上に大きい場合、文字列の終わりにブランクが入れられます。

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 1 別します。

lpsSource は、変換するソース (ANSI) 文字列を指します。

lpsTarget は、ターゲット（変換後）文字列を指します。 **wSource** は、ソース・ストリングの長さをバイト単位で表したものです。

lpwTarget は、ターゲット・バッファーのサイズが含まれているワード変数を指します。この変数は、ターゲット・バッファー内の変換後の文字の合W数で更新されます。

戻りコード

この関数が成功すると、EHNDT_SUCCESS (X'0000') が戻されます。ルーターがロードされていない場合、EHNDT_A2E_TABLE_NOT_FOUND (X'FFFC') が戻されます。一時バッファーを割り振ろうとしている間にエラーが生じた場合、EHNDT_MEMALLOC (X'FFFF') が戻されます。変換中に間違ったデータが見つかった場合、戻りコードは、変換されていない最初の文字の位置に 1 を加えたものです。

EHNDT_ASCIIToEBCDIC

目的

この関数は、ASCII から EBCDIC に文字列を変換します。このルーチンが ASCII/EBCDIC 間変換テーブルにアクセスできるようにするため、ルーターをロードする必要があります。ターゲット・ストリングが、変換後のストリングが入る十分な大きさでない場合、変換はターゲット・ストリングの終わりで停止します。ターゲット・ストリングが必要以上に大きい場合、文字列の終わりにブランクが入れられます。

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 1 別します。

lpsTarget は、ターゲット（変換後）文字列を指します。

lpsSource は、変換するソース (ASCII) 文字列を指します。

wSource は、ソース・ストリングの長さをバイト単位で表したものです。

lpwTarget は、ターゲット・バッファーのサイズが含まれているワード変数を指します。この変数は、ターゲット・バッファー内の変換後の文字の合W数で更新されます。

戻りコード

この関数が成功すると、EHNDT_SUCCESS (X'0000') が戻されます。ルーターがロードされていない場合、EHNDT_A2E_TABLE_NOT_FOUND (X'FFFC') が戻されます。

変換中に間違ったデータが見つかった場合、戻りコードは、変換されていない最初の文字の位置に 1 を加えたものです。

EHNDT_EBCDICToANSI

目的

この関数は、EBCDIC から Windows ANSI コード・ページに文字列を変換します。このルーチンが ASCII/EBCDIC 間変換テーブルにアクセスできるようにするため、ルーターをロードする必要があります。

ターゲット・ストリングが、変換後のストリングが入る十分な大きさでない場合、変換はターゲット・ストリングの終わりで停止します。ターゲット・ストリングが必要以上に大きい場合、文字列の終わりにブランクが入れられます。

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 1 別します。

lpsTarget は、ターゲット（変換後の）文字列を指します。

lpsSource は、変換するソース (EBCDIC) 文字列を指します。

wSource は、ソース・ストリングの長さをバイト単位で表したものです。

lpwTarget は、ターゲット・バッファーのサイズが含まれているワード変数を指します。この変数は、ターゲット・バッファー内の変換後の文字の合W数で更新されます。

戻りコード

この関数が成功すると、EHNDT_SUCCESS ('0000') が戻されます。ルーターがロードされていない場合、EHNDT_E2A_TABLE_.NOT_FOUND ('FFFC') が戻されます。変換中に間違ったデータが見つかった場合、戻りコードは、変換されていない最初の文字の位置に 1 を加えたものです。

EHNDT_EBCDICToASCII

目的

この関数は、EBCDIC から ASCII に文字列を変換します。このルーチンが ASCII/EBCDIC 間変換テーブルにアクセスできるようにするため、ルーターをロードする必要があります。

ターゲット・ストリングが、変換後のストリングが入る十分な大きさでない場合、変換はターゲット・ストリングの終わりで停止します。ターゲット・ストリングが必要以上に大きい場合、文字列の終わりにブランクが入れられます。

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 1 別します。

lpsTarget は、ターゲット（変換後）文字列を指します。

lpsSource は、変換するソース (EBCDIC) 文字列を指します。

wSource は、ソース・ストリングの長さをバイト単位で表したものです。

lpwTarget は、ターゲット・バッファーのサイズが含まれているワード変数を指します。この変数は、ターゲット・バッファー内の変換後の文字の合W数で更新されます。

戻りコード

この関数が成功すると、EHNDT_SUCCESS ('0000') が戻されます。ルーターがロードされていない場合、EHNDT_.E2A_TABLE_NOT_FOUND ('FFFC') が戻されます。変換中に間違ったデータが見つかった場合、戻りコードは、変換されていない最初の文字の位置に 1 を加えたものです。

第5部 Java プログラミング・インターフェース

第20章 Java 用ホスト・アクセス・クラス・ライブラリー の概要

本章では、IBM Java 用 eNetwork ホスト・アクセス・クラス・ライブラリー (ECL) について、 3270 および 5250 アプリケーションに関 8 するO囲で説明します。以下の事項が含まれます。

- Java 用 ECL の構造の概要
- ECL のためにインストールされるもの
- 選択可能なサンプルとその動作

注: eNetwork ホスト・アクセス・クラス・ライブラリーの説明では、HACL という頭字語も使用されます。本書では、ECL という頭字語を優先して使用します。

ECL とは?

Java 用 ECL は、アプリケーション・プログラマーが 3270 および 5250 データ・ストリーム・レベルでホスト・アプリケーションに簡単かつ迅速にアクセスできるようにする、クラスおよびメソッドのセットです。 ECL は、コアのホスト・アクセス関数を完全なクラス・モデルのA でB 装します。このクラス・モデルはグラフィック表 (とは独立していて、操作のためには Java 対 ~ のブラウザーまたはそれに相当する Java 環境だけを必要とします。

クラス・ライブラリーは以下のものを含む、ホスト接続の完全なオブジェクト指向抽象を表します。

- ホスト表 (空間 (画面) の読み h りと書き込み
- 表 (空間でのフィールドの列 s
- 操作員情報域 (OIA) からの状況に関する情報の読み h り
- ファイル転送
- 有効なイベントの s 同期通知

アプリケーション・プログラマーが作成した Java アプレットは、 (3270 や 5250 などの) データ・ストリーム表 (空間から得られたデータを、それらのマシンに常駐しないで操作することができます。表 (空間は、ホスト・アプリケーションによって提供されるデータと関連属性の両方を含む、仮想的な端末画面を表します。アプレットは、ある対話が完了してから、別のタスクに切り換えることも、そのままセッションをクローズすることもできます。トランザクションは、ホスト画面を一切表 (しないで完了させることができます。

ECL Java アプレットは、以下のことをB 行できます。

- ホストとのセッションをオープンする。
- 着信ホスト・データを待つ。
- 仮想的な画面から特定の文字列を獲得する。
- 文字列の関連属性を獲得する。
- 新 , 文字列の値を設定する。
- データ・ストリーム・ファンクション・キーをホストに戻す。

- ！のホスト・セッションを待つ。

ECL は、 EHLLAPI などの、クライアント特定の画面切り h りアプリケーション・プログラミング・インターフェースに f べて、以下のような著しい長所があります。

- ECL はプラットフォーム独立です。
- ECL は、解 a されたエミュレーター画面ではなく、データ・ストリームを直接操作します。これにより、ビジュアル・ウィンドウでデータ・ストリームを解 a および表 (するためのオーバーヘッドを除 n することができます。
- ECL は、ローカル・ワークステーションでB行されるエミュレーター・ソフトウェアを必要としないため、プラットフォーム固有の画面A 0 とキーボードのレイアウトへの依存性を低下させることができます。
- ECL は、標準の Web および Java テクノロジーを使用してクライアント・ワークステーションにダウンロードし、B行することができます。これにより、保 i およびリソースが大幅に節約できます。

ECL の概念

以下のセクションでは、ECL のいくつかの基本概念について説明します。これらの概念を理解しておくと、ライブラリーを効果的に利用できるようになります。

セッション

ECL のコンテキストでは、セッション・オブジェクト (ECLSession) が、ホストへの接続とその接続の特性をカプセル化します。セッション・オブジェクトは、その他のセッション固有オブジェクト、すなわち ECLPS (表 (空間)、 ECLOIA (操作員情報域)、および ECLXfer (ファイル転送) のコンテナーとしても！能します。

セッション・オブジェクトに関連したグラフィカル・ユーザー・インターフェース (GUI) はありません。つまり、ECLSession のインスタンスを作成してもエミュレーター画面は表 (されません。

コンテナー・オブジェクト

ECL クラスの中には、他のオブジェクトのコンテナーとして！能するものがあります。 ECLSession オブジェクトには、 ECLPS、ECLOIA、および ECLXfer オブジェクトのインスタンスが含まれています。コンテナーは、含まれるオブジェクトを指すポインターを戻す j 段として使用できます。 ECLSesison オブジェクトには、OIA オブジェクトへのポインターを戻す GetOIA メソッドが含まれます。含まれるオブジェクトは、コンテナーのクラスの共用メンバーとしてはB装されず、 ECL メソッドを介してのみアクセスすることができます。

リスト・オブジェクト

ECL クラスの中には、リスト？復！能をwえているものがあります。たとえば、ECLConnList クラスは接続のリストを管理します。 ECL リスト・クラスは、リスト内容の変更を？Gさせるために s 同期更新されることはありません。アプリケーションが明 (的に Refresh メソッドを呼び出して、リストの内容を更新しなければなり

ません。これによりアプリケーションは、 ? 復中にリストが変更される可能性について考慮せずにリストを + り返すことができます。

イベント

ECL は、特定のイベントの s 同期通知を行う ! 能を w えています。アプリケーションは、特定イベントが / 生したときに通知を u けるように選択することができます。たとえば、アプリケーションは、ホストへの接続の状況が変更されたときに通知を u けることができます。現在、ECL は以下のイベントに関する通知をサポートしています。

表 21. APPC 用ヘッダー・ファイルおよびライブラリー

イベント	イベントを取り込むために使用されるインターフェース
通信の接続と切断	ECLLCommNotify
表 (空間の更新	ECLPSNotify
操作員情報域 (OIA) の更新	ECLOIANotify

イベント通知は、該当の ECL Notify インターフェースによって定 A されています。イベント・タイプごとに別個のインターフェースがあります。アプリケーションは、イベントの通知を u けるために、通知が必要なイベント・タイプに対 ~ するインターフェースを B 装するオブジェクトを定 A し、作成しなければなりません。そのオブジェクトは、適切な ECL 登録ファンクションを呼び出して登録しなければなりません。アプリケーション・オブジェクトが登録されると、イベントが / 生したときにそのオブジェクトの NotifyEvent メソッドが呼び出されます。

注: アプリケーションの NotifyEvent メソッドは、別個の B 行スレッドで s 同期的に呼び出されます。したがって、NotifyEvent メソッドには再入する必要があります。アプリケーション・リソースがアクセスされた場合には、適切なロックまたは同期を使用するようにしてください。

エラー処理

一 L に ECL は、ECLErr オブジェクトを送出してアプリケーションにエラーを知らせます。エラーを捕そくするために、アプリケーションで ! のように ECL オブジェクトの呼び出しを try/catch ブロックに組み込んでください。

```
try {
       int pos = ps.ConvertRowColToPos(row, col);

       //...possibly more references to ECL objects...

   } catch (ECLErr err) {
      System.out.println("ECL Error!  " + err.GetMsgText());
   }
```

ECL エラーが検出されると、アプリケーションは ECLErr オブジェクトのメソッドを呼び出して、エラーの正確な原因を = 別することができます。 ECLErr オブジェクトは、完全な言語依存のエラー・メッセージを構成するために呼び出すこともできます。

アドレス指定 (行、カラム、位置)

ECL には、ホスト表 (空間内の点 (文字位置) をアドレス指定するための方法が 2 つあります。アプリケーションは、文字を行/カラムＶ号でアドレス指定することも、単一のリニア位置の値でアドレス指定することもできます。表 (空間のアドレス指定は、アドレス指定体Ｏと無関 8 に、常に (0 ベースではなく) 1 ベースで行われます。

行とカラムによるアドレス指定体Ｏは、ホスト・データの物理画面表 (に直接関連するアプリケーションで使用すると便利です。 (左上隅に行 1、カラム 1 がある) 直交座標Ｏは、画面上の点をアドレス指定するための + 然な方法です。 (左上隅が位置 1 で、左から＆への方向で上から下に向かって大きくなる) リニア定位置アドレス指定方 0 は、表 (空間全体をデータ要素の単一配列で表 (するアプリケーション、または EHLLAPI インターフェースから移植されたアプリケーションで使用すると便利です。

一Ｌに、同じメソッドに関して異なるシグニチャーを呼び出すと、異なるアドレス指定体Ｏが選択されます。たとえば、ホスト・カーソルを特定の画面座標に移動したい場合、アプリケーションは！の 2 つのいずれかのシグニチャーで ECLPS::SetCursorPos メソッドを呼び出すことができます。

```
ps.SetCusorPos(81);
ps.SetCursorPos(2, 1);
```

ホスト画面が行当たり 80 カラムで構成されている場合には、これらのステートメントの効果は同じになります。この例は、アドレス指定体Ｏ間の y 妙な相違も表しています。リニア位置方 0 を使用すると、アプリケーションが表 (空間の行当たり文字数として特定の数を想定している場合、予期しない k 果が生じることがあります。たとえば表 (空間が 132 カラムで構成されている場合、この例のコードの最初の行は、行 1、カラム 81 にカーソルを移動します。コードの 2 Ｖ目の行は、表 (空間構成とは無関 8 に、行 2、カラム 1 にカーソルを移動します。

Communications Server for NT サーバーへの ECL のインストール

これは、Communications Server for Windows NT だけで使用することができます。

Communications Server for Windows NT 4.0 CD-ROM を挿入してインターフェースのステップに従うと、「**セットアップ**」をクリックして InstallShield** ウィザードのインストールを開始するようにプロンプト指 (されます。このウィザードをインストールすると、残りのインストール j 順がこのウィザードによって案内されます。ウィザードのインストールが完了すると、「IBM Communications Server へようこそ」ウィンドウが表 (されます。「**次へ**」をクリックして続行してください。！の一連のパネルでは、セットアップ・タイプ、Communications Server をインストールしたいドライブとディレクトリー、 IBM Files On-Demand に匿名アクセスするための FTP ディレクトリー、および ECL クラス・ファイルをインストールしたいドライブとディレクトリーを選択するようにプロンプト指 (されます。

このインストールにより、サーバーにあるアプレットから ECL Java クラス・ファイルにアクセスしたり、サーバーにある Java アプリケーション、ECL コード・ページ・コンバーター、ECL の文書、および Java アプレットと Java アプリケーションのサンプルから ECL Java クラス・ファイルにアクセスしたりできるようになります。 (クライアントで Java アプリケーションをB行するために ECL をサーバーにインストールする必要はありません。)

以下に、ECL パーツとその定Aを (します。

IBMCS¥ecl101.jar	このファイルは、サーバーから ECL Java のアプレットとアプリケーションを実行するために使用されます。
IBMCS¥*.class	これらのファイルは、サーバー・アクセスまたは Web ベースのアクセスのために、ユーザー定義のディレクトリーにインストールすることができます。
IBMCS¥ECL¥ZIP*.zip	zip 形式の ECL クラス・ファイル。これらのファイルは、すべての ECL クラス・ファイルにアクセスしやすくするためにインストールされます。後で説明するコード・ページ・コンバーターは、ecl101n.zip というファイルに入っています。
¥IBMCS¥SDK¥JAVA¥ECL¥DOC¥*.*	HTML 形式によるオンラインの ECL 文書。この文書は、Web ブラウザーでアクセスされることを想定して形式化されています。"ECLReference.html" というファイルから開始することをお勧めします。
¥IBMCS¥SDK¥JAVA¥ECL¥SAMPLES¥*.*	サンプル・プログラム。これについては、後で説明します。
¥IBMCS¥jre¥*.*	サーバーにインストールされた ECL ファイルに相当する Java 実行時環境。

Communications Server for NT 32 ビット Windows クライアントへの ECL のインストール

これは、Communications Server Windows NT および Windows 95 の SNA API クライアントだけで使用することができます。

ECL が Typical または Custom クライアント・インストール・オプションでクライアントにインストールされている場合、 Java B行時環境 (jre) とともに ecl101.jar が CSNT クライアント・ディレクトリー (たとえば、 CNSTAPI) にインストールされます。これにより、ECL Java アプリケーションは、 ecl101.jar ファイルに入っている ECL Java クラスにアクセスできるようになります。 ECL は、それだけでは完全なアプリケーションではありません。希望するファンクションのセットをB行するための ECL Java クラスを使用する、 Java アプリケーションを作成する必要があります。 ECL のクライアント・インストールは、ユーザー作成の ECL Java アプリケーションをB行するために必要なレベルの！能をwえています。サーバーに追加の ECL コードをインストールする必要はありません。

サイズの制約があるため、ecl101.jar ファイルにはQ語のコード・ページだけしか含まれていません。その他のコード・ページ・コンバーター・クラスは、サーバーにインストールされている zip ファイル ecl101n.zipから入 j することができます。完全な ECL 文書、サンプルの Java アプレットと Java アプリケーション、および ECL で Java アプレットをB行する！能もサーバーにインストールすることができます。

Classpath の設定

Java アプリケーションまたは Java アプレットをB行する場合には、環境変数 **classpath** を、そのアプリケーションまたはアプレットのB行に必要な Java クラスのロケーションのフル・パス名と同じ値に設定してください。たとえば、ECL Java アプリケーションを作成して SNA API クライアント・サブディレクトリー (たとえば、C:¥CSNTAPI) にコピーする場合には、

- **classpath** を！のように設定する必要があります。

  C:¥CSNTAPI;C:¥CSNTAPI¥ecl101.jar
- コマンド行には！のように入力する必要があります。

  set classpath=C:¥CSNTAPI;C:¥CSNTAPI¥ecl101.jar

Java B行時環境 (JRE) を使用している場合には、 **classpath** 環境変数は使用されませんが、 JRE を呼び出すときに **cp** オプションで Java クラスへのパスを指定することができます。

ECL コード・ページ・コンバーター

ECL コード・ページ・コンバーターは複数の言語をサポートします。サイズの制約があるため、 ecl101.jar ファイルと ecl101.zip ファイルにはQ語のコード・ページだけしか含まれていません。その他のコード・ページ・コンバーター・クラスは、サーバーにインストールされているファイル ecl101n.zipから入 j することができます。このファイルを unzip すると、コード・ページ・コンバーター・ファイルを見つけることができます。これらのファイルは、Java アプリケーションをB行しているマシンにコピーすることも、 ecl101.jar/zip A 0 のファイルと k 合させて新 , jar/zip ファイルを作成することもできます。ファイルが置かれる Classpath (com¥ibm...) を必ず保管してください。

ECL アプリケーションまたはアプレットのサイズを減らすために、アプリケーションまたはアプレットに必要なコンバーター・クラス・ファイルだけをコピーするようにしてください。コード・ページ・コンバーター・クラスのB装に関する情報は、ECL 文書に - 載されています。

ECL サンプル

以下のセクションでは、 **IBMCS¥SDK¥JAVA¥ECL¥SAMPLES** サブディレクトリーにインストールされている Host Launchpad サンプルおよび各 o サンプルについて説明します。

Host Launchpad サンプル

IBMCS¥SDK¥JAVA¥ECL¥SAMPLES¥LAUNCHPAD にインストールされている Host Launchpad サンプルは、ホストへの接続、ログオン、ホストのアプリケーションおよび情報へのアクセス、およびログオフのために ECL を使用します。このサンプルは、アプレットとしてB行することも、アプリケーションとしてB行することもできます。アプレットは HostLP.htm という HTML ファイルからB行することができます。そのために、HostLP.htm に！の行を追加する必要があります。

```
<APPLET archive=ecl101.jar  code=HostLaunchPad.class
    id=HostLP
    width=400
    height=10 >
</APPLET>
```

また、Win32 システムでは classpath を！のように設定する必要があります。

```
  set classpath=%classpath%;x:¥pathto¥ecl101.jar;
```

以下のソース・ファイルはこのサンプルを構成するものであり、使用する前にコンパイルしておかなければなりません。

CFilelData.java
CInitData.java
CLogonData.java
FileLaunchApp.java
FileResultsApp.java
HostLPConstants.java
HostLaunchPad.java
LaunchBase.java
LogonLaunchApp.java
SAFilel.java
SAFilelResults
SALogon.java
SALogoutLaunch.java
SAMore.java
SAReady.java
SAVMLogin.java
SAMsg10.java
ScreenAction.java
ScreenState.java

メイン・クラスは HostLaunchPad です。

このサンプル・コードを理解するために、HostLaunchPad.java から開始し、さらに LaunchBase.java、LogonLaunchApp.java、SAMsg10.java、SALogon.java、SAVMLogin、SAMore、SAReady.java、および ScreenAction.java へと進んでください。

このサンプルは、ホスト・アクセス・クラス・ライブラリー (ECL) を使用してホスト・アクセス権を獲得するためにほとんどのアプレットで使用される、 g 要な概念を導入しています。 Launchpad サンプルは以下のタスクをB施します。

1. ホスト・セッションを確立します。これには、 ECLSession のインスタンスの組み立て、および StartCommunication() メソッドの呼び出しが含まれます。このロジックは LogonLaunchApp.java に入っています。
2. 表 (空間 (PS) イベントを登録します。 PS イベントは、画面が更新されたときに生成されます。登録は RegisterPSEvent() メソッドを使用して行われます。このロジックは SALogon.java に入っています。
3. PS イベントに~答します。これには、画面を認 1 すること、およびキーストローク、 (Enter キーやファンクション・キーなどの) ホスト・ファンクション、およびカーソル移動によって画面をナビゲートすることが含まれます。画面処理論理は 、
   NotifyEvent()、SAReady.java、SAMore.java、SAMsg10.java、SAFilelResults.java、および ScreenAction.java に入っています。
4. 単純な VM コマンド 'filel'をB行してそのk果をダイアログに表 (するための論理をB装します。ファイル・リスト・コマンドをB装してそのk果を表 (するためのファイルは、 FileLaunchApp.java、SAFilel.java、SAFileResults.java、ScreenAction.java、および FilelResultsApp.java に入っています。

注: このサンプルはいくつかの画面を処理しますが、それらの画面はシステムごとに異なる可能性があります。おそらく、コードにc干の変更を加えて、ユーザーが使用している特定システムに合わせてカスタマイズする必要があります。画面認 1 およびキーストローク論理のメソッドは、SAVMLogin.java、SAReady.java、 SAMore.java、SAMsg10.java、および SAFilelResults.java の各ファイルに入っています。これらのファイルは、それぞれ異なる画面の処理を扱います。

このサンプルをデバッグするためには、B際に画面で行われることを調べる必要があります。最初のパネルにあるチェックボックスを使用して PS デバッガーを使用可能にすると、各o のホスト画面でアプレットが進行しているときに画面の内容を表 (することができます。

各o サンプル

追加のサンプルが **IBMCS¥SDK¥JAVA¥ECL¥SAMPLES¥MISC** サブディレクトリーにインストールされています。これらのサンプルは、ECL の多くのクラスおよびメソッドの基本的な使用法を (しています。これらのサンプルはすべて ECLAppletInterfaceをB装しているため、 Run Applet ！能を使用して Host On-Demand からB行することができます。これらのサンプルを Host On-Demand なしでB行するためには、ECLSession オブジェクトを設定する必要があります。これは、 Host Launchpad サンプルで ECLSession オブジェクトを設定したときと同じ方法で行うことができます。

第21章 Java 用 CPI-C の使用

これは、Communications Server Windows 95 および Windows NT の SNA API クライアントだけで使用することができます。

本章では、Java 用共通プログラミング・インターフェース通信 (CPI-C) API およびその使用法について説明します。以下の内容が含まれています。

- Java 用 CPI-C の概要
- Java 用 CPI-C のためにインストールされるもの
- 選択可能なサンプルとその動作

Java 用 CPI-C の概要

Java 用 CPI-C は、開 / T が Java 言語で 共通プログラミング・インターフェース通信 (CPI-C) API を使用できるようにする、プログラミング・ツールキットです。CPI-C は、SNA LU 6.2 用のオープン API です。 CPI-C API の詳細については、IBM eNetwork Communications for Windows NT バージョン 6.0 に含まれている、 *共通プログラミング・インターフェース コミュニケーション・インターフェース (CPI-C) 解説書* (SC88-7217) を参照してください。

このツールキットの g 要な目標は、従来の C を Java に変換することです。したがって、このツールキットの呼び出しは C におけるツールキット呼び出しとよく似ています。 Java 用 CPI-C は、CPI-C のネイティブ API よりも上の層として提供されていて、 CPI-C を使用するためには、このネイティブ・コードをインストールしなければなりません。

このツールキットには、ツールキット内のすべてのクラス、メソッド、および変数に関する、プログラマー向けの参照文書が w わっています。この文書は HTML A 0 になっていて、使いやすい相互参照も提供されています。

このプログラミング・ツールキットには、 CPI-C パラメーターを } めるオブジェクトを含む Java クラスのセットと、 C の CPI-C 関数にマップされるメソッドを定 A する **CPIC** クラスも用意されています。このツールキットに含まれているサンプル・アプリケーション (JPing.class) を B 行することも、独 + のアプリケーションを作成することもできます。

Java 用 CPI-C のバインドを利用すると、Java アプリケーションで SNA ネットワークを使用したり、 CPI-C をネットワーキング API として使用したりできます。これらの Java アプリケーションは、 ! のパートナーに接続することができます。

- 新 , の Java 用 CPI-C アプリケーション
- 新 , または既存の、Java 以外の CPI-C アプリケーション
- 新 , または既存の APPC アプリケーション

Java 用 CPI-C のインストール

Java 用 CPI-C ツールキットとともに、以下のものがインストールされます。

- CPICJAVA.JAR には、Java 用 CPI-C プログラムの作成に使用される Java クラスが含まれています。この JAR ファイルはユーザーの CLASSPATH 環境変数に組み込むか、あるいは Java 用 CPI-C アプリケーションを呼び出すときに明 (的に指定するかしてください。このファイルは、他の API クライアント・ファイルとともにユーザーのワークステーションにインストールされます。 JAR ファイルには、サンプル・アプリケーション JPing.class も含まれています。
- CPICJAVA.DLL はプラットフォーム固有の DLL であり、 Java 用 CPI-C クラスとユーザーのにインストールされたネイティブ LU 6.2 サポートの間のリンケージを含んでいます。このファイルは、他の API クライアント DLL とともにワークステーションにインストールされます。
- Jcpic001.htm は、 Java 用 CPI-C の各クラス、メソッド、および変数が (されているプログラマー向けの参照文書のルートです。このファイルは、Java 用 eNetwork ホスト・アクセス・クラス・ライブラリー (ECL) のインストール時に、**IBMCS¥SDK¥JAVA¥CPIC¥DOC** サブディレクトリーにインストールされます。この文書は、カスタム・アプリケーションの開 / に使用されます。
- CPICJAVA.HTM は、ツールキットおよびサンプル・アプリケーションに関する簡単な紹介です。この HTML A 0 ファイルは、他の API クライアント・ファイルとともにユーザーのワークステーションにインストールされます。
- JPing.java は、JPing.class サンプル・アプリケーションのソース・ファイルです。このファイル内のコメントは、このツールキットを使用してプログラミングを行うためのヒントになります。 JPing.java ファイルは、Java 用 ECL のインストール時にサブディレクトリーにインストールされます。

Java 用 CPI-C のサンプル

以下のセクションでは、Java 用 CPI-C のクライアントおよびサーバーのサンプルについて説明します。

クライアントのサンプル

このツールキットに含まれているサンプルは、 APING クライアント・ユーティリティーと同じ働きをします。つまり、サーバー・プロセスにデータが送信され、サーバー・プロセスによってそのデータが APING ユーティリティーにエコーされます。このサンプル・クライアントは、コンパイルされて CPICJAVA.JAR ファイルに置かれています。ソース・ファイル (JPing.java) は、 Java 用 ECL のインストール時に **IBMCS¥SDK¥JAVA¥CPIC¥SAMPLES** サブディレクトリーにインストールされます。

この API は、COM.ibm.eNetwork.cpic という Java パッケージで提供されています。！のサンプルのコードの最初の行は、ツールキットで提供されたクラスのアクセスするために必要です。 **CPIC** クラスは、ネイティブ CPI-C コードへのメイン・インターフェースです。 **CPIC** クラスには、会話 ID などの CPI-C 内で定 A された多くの定数、およびネイティブ CPI-C 呼び出しにパススルーされるメソッドが含まれています。

宣言する必要のある **CPIC** オブジェクトは、クラスごとに 1 つだけです。 Java は、**CPIC** オブジェクトがインスタンス化されるときに、ネイティブ・メソッドを含むダイナミック・リンク・ライブラリー (DLL) (CPICJAVA.DLL) をロードします。

！のサンプルは、CPI-C パイプラインを表しています。 JPing.java ソース・ファイル内の情報は複 L されません。

注: ！のサンプルのコードには、注 a が含まれています。

```
/*-------------------------------------------------------------
 * Pipeline transaction, client side.
 *-------------------------------------------------------------*/
import COM.ibm.eNetwork.cpic.*;
public class Pipe extends Object {
   public static void main(String args[]) {

   // Make a CPIC object
   CPIC cpic_obj = new CPIC();
```

それぞれのタイプのパラメーターには固有のクラスがあり、それらの各クラスに関連した定数がクラス変数として定 A されています。たとえば、CPICReturnCode クラスには成功戻りコード CM_OK が定 A されています。

パラメーターのタイプごとにクラスが h められている理由は、2 つあります。まず、Java はすべてのパラメーターを値によって渡すため、整数のような単純なタイプのデータを戻す方法がありません。あるオブジェクトをパラメーターとしてメソッドに渡すと、メソッドがオブジェクト内で変数を設定することができるため、 / 呼 T にデータが戻されます。 ！に、オブジェクトを使用することによって、定数を認 1 するオブジェクト内でそれらの定数がカプセル化されます。これは、情報を # すための標準的な ; 法です。

```
   // Return Code
   CPICReturnCode cpic_return_code =
      new CPICReturnCode(CPICReturnCode.CM_OK);

   // Request to send received?
   CPICControlInformationReceived rts_received =
      new CPICControlInformationReceived(
         CPICControlInformationReceived.CM_NO_CONTROL_INFO_RECEIVED);
```

CPI-C 送信ファンクションは C 言語バッファー、すなわち特定タイプをもたない割り振りスペースを予想しています。 Java には、C とは異なり、タイプのないメモリーを割り振る！能はありません。 Java では、プリミティブ以外のすべてのものはオブジェクトです。プログラムが送信するものはすべて、そのオブジェクト・タイプから C スタイルのバイト配列に変換されなければなりません。

Java には、これらの変換を容易にするためのメソッドが w わっています。たとえば、Java は文字列を Java のバイト配列に変換することができます。バイト配列は Java のオブジェクトですが、 Java ではバイト配列からネイティブ・メソッドによってデータを抽出することができます。

```
 // String to Send
   String sendThis = "Test of the PipeLine Transaction";

   // Length of String to send
   CPICLength send_length = new CPICLength(sendThis.length());
```

```
   // Convert String to send to a Java array of bytes
   byte[] stringBytes = new byte[ send_length.intValue()];
   sendThis.getBytes(0,send_length.intValue(),stringBytes,0);
```

バッファー処理のように、 CPI-C ネイティブ呼び出しは - 号宛先名が Java String ではなく C 文字列であることを予期しています。ツールキットは必要に ~ じて、それらを Java 文字列から C 文字列に + 動的に変換します。一 L に、ツールキットが特定の Java を予期している場合には + 動変換が可能です。

会話 ID は Java のバイト配列であり、ツールキットによって単純なバイトのブロックからなる C 配列に + 動的に変換されます。

```
   // this hardcoded sym_dest_name must
   // be 8 chars long & blank padded
   String sym_dest_name = "PIPE    ";

   // Space to hold a conversation ID
   // (which is just a bunch of bytes)
   byte[] conversation_ID = new byte[CPIC.CM_CID_SIZE];
```

このプログラムが開始する CPI-C 呼び出しは、C で使用される CPI-C 呼び出しに s 常によく似ています。ただし、メソッド呼び出しには CPI-C オブジェクトの名前が接頭部として使用されていて、パラメーターには参照による引き渡し (&) シンボルが接頭部として付いて**いません**。

```
  //
   // Initialize CPI-C
   //
   cpic_obj.cminit(             /* Initialize_Conversation        */
             conversation_ID,   /*  O: returned conversation ID   */
             sym_dest_name,     /*  I: symbolic destination name  */
             cpic_return_code); /*  O: return code from this call */
   //
   // ALLOCATE
   //
   cpic_obj.cmallc(             /* Allocate Conversation          */
             conversation_ID,   /*  I: conversation ID            */
             cpic_return_code); /*  O: return code from this call */
   //
   // SEND
   //
   cpic_obj.cmsend(             /* Send_Data                      */
             conversation_ID,   /*  I: conversation ID            */
             stringBytes,       /*  I: send this buffer           */
             send_length,       /*  I: length to send             */
             rts_received,      /*  O: was RTS received?          */
             cpic_return_code); /*  O: return code from this call */
   //
   // DEALLOCATE
   //
   cpic_obj.cmdeal(             /* Deallocate                     */
             conversation_ID,   /*  I: conversation ID            */
             cpic_return_code); /*  O: return code from this call */
   } // end main method
} // end the class
```

サーバーのサンプル

サーバーはそれ + 体を初期化し、会話を u け入れ、データを u 信し、診断情報を印刷します。クライアントの場合と同じように、CPI-C パラメーターを保持するクラスをインスタンス化します。これらの多くは、インスタンス・データとして整数だけを含みます。オブジェクトを使用することにより、参照による呼び出しを模倣することができます。また、 u 信したデータを保持するバイト配列の割り振りも行います。

注: ！のサンプルのコードには、注 a が含まれています。

```
/*-------------------------------------------------------------
 * Pipeline transaction, server side.
 *-------------------------------------------------------------*/
import COM.ibm.eNetwork.cpic.*;
import Java.io.IOException;

public class PipeServer extends Object {
   public static void main(String args[]) {

    CPIC cpic_obj = new CPIC();

    // Space to hold the received data
    byte[] data_buffer;
    data_buffer =  new byte[101];

    CPICLength requested_length = new CPICLength(101);
    CPICDataReceivedType data_received =
       new CPICDataReceivedType(0);
    CPICLength received_length = new CPICLength(0);
    CPICStatusReceived status_received =
       new CPICStatusReceived(0);
    CPICControlInformationReceived rts_received =
       new CPICControlInformationReceived(0);
    CPICReturnCode cpic_return_code =
       new CPICReturnCode(0);

    // Space to hold a conversation ID -- a bunch of bytes
    // The first line declares conversation_ID to be a reference to
    // a byte array object.  The second line creates such an object,
    // and assigns the reference to the byte array object.
    byte[] conversation_ID;
    conversation_ID = new byte[cpic_obj.CM_CID_SIZE];
```

CPI-C u 信呼び出し (cmrcv) は Java のバイト配列を戻しますが、パイプ・トランザクションは文字列を予期しています。プログラマーは、引き数としてバイト配列を使用する文字列クラス・コンストラクターを使用して、バイト配列を文字列に変換することができます。

```
    //
    // ACCEPT
    //
    cpic_obj.cmaccp(         /* Accept_Conversation            */
        conversation_ID,     /*  O: returned conversation ID   */
        cpic_return_code);   /*  O: return code                */
    //
    // RECEIVE
    //
    cpic_obj.cmrcv(          /* Receive                        */
        conversation_ID,     /*  I: conversation ID            */
        data_buffer,         /*  I: where to put received data */
```

```
        requested_length,    /*  I: maximum length to receive  */
        data_received,       /*  O: data complete or not?      */
        received_length,     /*  O: length of received data    */
        status_received,     /*  O: has status changed?        */
        rts_received,        /*  O: was RTS received?          */
        cpic_return_code);   /*  O: return code from this call */
    //
    // Do some return code processing
    //
    System.out.println("  Data from Receive:");
    System.out.println("    cpic_return_code        = " +
                       cpic_return_code.intValue());
    System.out.println("    cpic_data_received      = " +
                       data_received.intValue());
    System.out.println("    cpic_received_length    = " +
                       received_length.intValue());
    System.out.println("    cpic_rts_received       = " +
                       rts_received.intValue());
    System.out.println("    cpic_status_received    = " +
                       status_received.intValue());
    // Create a Java String from the array of bytes that you received
    // and print it out.
    String receivedString = new String(data_buffer,0);
    System.out.println(
       "    Recevied string                 = "
       + receivedString );

    //
    // BLOCK so that the Server Window doesn't disappear
    //
    try{
        System.out.println("Press any key to continue");
        System.in.read();
       }
    catch
       (IOException e){ e.printStackTrace(); }
    }
}
```

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 ！戻りコード（および、該当する場合は 2 ！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
パーソナル・コミュニケーションズおよび Communications Server は会話の割り振りに : 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 / 行された verb から戻されます。関連する 2 ！コードは ! 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 J 続条 o が原因で、会話の割り振りができませんでした。エラーを = 別するには、システム管理 T 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、割り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 o が原因で、会話の割り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 - 録されています。できればタイムアウトによりこの条 o が解消されてから、割り振りを再試行してください。

AP_SECURITY_NOT_VALID
割り振り要 a 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 u け入れませんでした。

AP_TRANS_PGM_NOT_AVAIL_RETRY
リモート LU が、要 a 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、割り振りを q] しました。タイムアウトなどの一時的条 o が原因で、要 a 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 - 録されている場合があります。この条 o は、オペレーターの介入なしで + 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 o 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 a 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、割り振りを q] しました。 J 続条 o または > J 続条 o が原因で、要 a 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 - 録されている場合があります。この条 o は、オペレーターの介入なしで + 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
割り振り要 a 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 1 されません。

AP_PIP_NOT_ALLOWED
要 a 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 u 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 (しています。

AP_PIP_NOT_SPECIFIED_CORRECTLY
要 a 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 u 信できますが、提供された PIP にエラーを / 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 (しています。

AP_CONVERSATION_TYPE_MISMATCH
要 a したトランザクション・プログラムは、割り振り要 a に指定した会話のタイプ（基本またはマップ 0 ）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 (しています。

AP_SYNC_LEVEL_NOT_SUPPORTED
要 a したトランザクション・プログラムは、割り振り要 a 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 (しています。

AP_CANCELLED
会話が h 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 / 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 J 続条 o が原因で、会話が終了しました。システム管理 T は、システム・エラー・ログを調べてエラーの原因を = 別する必要があります。エラー条 o 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 / するかどうかを確認してください。再 / する場合は、システム管理 T はエラー・ログを調べて、エラーの原因を = 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 / 行しました。

AP_DEALLOC_ABEND
！のいずれかの理由で会話が割り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 / 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 a を送信した。

AP_DEALLOC_ABEND_PROG

！のいずれかの理由で会話が割り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 / 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 a 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 / 行したため、会話が割り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 / 行したため、会話が割り振り解除されました。

AP_DEALLOC_NORMAL

この戻りコ　されまし

AP_ABEND_SVC にセットして **DEALLOCATE** verb を / 行したため、会話が割り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 / 行したため、会話が割り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り振り解除されましたが、ローカル LU はその理由を認 1 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 s ブロッキング verb を / 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！のいずれかの verb を / 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 N 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、！のいずれかの verb を / 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 u 信されていないデータが除 n 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 / 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 N ての生じた論理レコードを u 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 / 行しました。データは切り N 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 / 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 n 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 / 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Nての生じた論理レコードをすでに u けh っている場合があります。

AP_TP_BUSY

パーソナル・コミュニケーションズおよび Communications Server が同じ会話の別の verb を処理している間に、ローカル・トランザクション・プログラムがブロッキング verb を パーソナル・コミュニケーションズおよび Communications Server に出しました。

AP_UNEXPECTED_SYSTEM_ERROR

パーソナル・コミュニケーションズおよび 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 o 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 a 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. LUA Verb 戻りコード

この付録では、いくつかの SLI verb に共通の 1 ！戻りコード (および、該当する場合は 2 ！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。

1 ！戻りコード

このセクションでは、LUA 1 ！戻りコードについて説明します。

LUA_OK
LUA verb が正常に完了しました。

LUA_PARAMETER_CHECK
LUA フィーチャーが誤ったパラメーターを検出しました。

LUA_STATE_CHECK
セッションが、 / 行された verb には不適切な状態になっています。

LUA_SESSION_FAILURE

セッションがダウンしました。 2 ！戻りコードに特定の理由が (されています。

LUA_UNSUCCESSFUL
verb が正常に完了しませんでした。

LUA_NEGATIVE_RESPONSE
！のいずれかの条 o が / 生しています。

- LUA アプリケーション・プログラムによって] 定 ~ 答されたチェーンに関して、チェーン終了に達した。 2 ！戻りコードは設定されていない。
- LUA が 1 ！ LU からのメッセージにエラーがあることを検出し、] 定 ~ 答を送信した。このエラーは、1 ！ LU からチェーン終了を u 信したときに戻されます。 2 ！戻りコードには、] 定 ~ 答とともに送られたセンス・データが入っています。

LUA_CANCELED
この verb は、2 ！戻りコードで指定された理由によって h り消されました。

LUA_IN_PROGRESS

この同期コードは、 s 同期コマンドが u 信されて、完了しなかった場合に戻されます。

LUA_STATUS
SLI は、2 ！戻りコードでアプリケーションの状況に関する情報を提供します。

LUA_COMM_SUBSYSTEM_ABENDED

Communications Server が異常終了しました。

LUA_COMM_SUBSYSTEM_NOT_LOADED

Communications Server がロードされていません。

LUA_INVALID_VERB_SEGMENT

データ・セグメントに verb 制御ブロック全体が含まれていないため、 LUA が verb を処理できませんでした。 verb 制御ブロックの終わりのアドレスが、セグメントの終わりを超えています。

LUA_UNEXPECTED_DOS_ERROR

Communications Server がシステム呼び出しを / 行し、この verb が 1 ！戻りコード UNEXPECTED_DOS_ERROR で通知された後で、予期しないシステム・エラーが / 生しました。 2 ！戻りコードに予期しないシステム・エラーが含まれています。

LUA_STACK_TOO_SMALL

LUA アプリケーション・スタックが小さすぎるため、LUA が要 a を処理できません。

LUA_INVALID_VERB

LUA は、 u 信した verb 制御ブロック内の verb コードまたは verb 操作コード (またはその両方) を認 1 しません。

2 ！戻りコード

このセクションでは、LUA 2 ！戻りコードについて説明します。

LUA_SEC_OK
2 ！戻りコードに関連した1 ！戻りコードについて、追加情報が得られます。

LUA_INVALID_LUNAME
verb に無効な **lua_name** が指定されています。

LUA_BAD_SESSION_ID
verb 制御ブロックで、**lua_sid** パラメーターに誤った値が指定されています。

LUA_DATA_TRUNCATED
(**lua_max_length** で指定された) バッファー長が不十分なため u 信したデータが } 容できず、データが切り N てられました。

LUA_BAD_DATA_PTR
コマンドはデータの提供または戻りを必要としますが、 **lua_data_ptr** パラメーターに無効なポインターが含まれているか、あるいは読み h り/書き込みセグメントを指していません。

LUA_DATA_SEG_LENGTH_ERROR
！のいずれかの条 o が / 生しています。

- **RUI_READ** または **SLI_RECEIVE** verb で提供されたデータ・セグメントが、 **lua_max_length** パラメーターで指定された長さよりも短くなっている。
- **RUI_WRITE** または **SLI_SEND** verb で提供されたデータ・セグメントが、**lua_data_length** パラメーターで指定された長さよりも短くなっている。

- **RUI_READ**、**RUI_WRITE**、 **SLI_RECEIVE**、または **SLI_SEND** verb で提供されたデータ・セグメントが、読み h り/書き込みデータ・セグメントではない。

LUA_RESERVED_FIELD_NOT_ZERO
直前に / 行されたコマンドに、ゼロ以外の予約パラメーターが指定されていました。

LUA_INVALID_POST_HANDLE
LUA verb 制御ブロックで有効なセマフォーが指定されていませんでした。LUA verb が同期完了しない場合には、verb の完了を通知するためにセマフォーが必要です。

LUA_PURGED
RUI_PURGE または **SLI_PURGE** が / 行されたため、 **RUI_READ** または **SLI_RECEIVE** verb が h り消されました。

LUA_BID_VERB_SEG_ERROR
lua_flag1.bid_enable として 1 が設定された **SLI_RECEIVE** が / 行される前に、 **SLI_BID** verb 制御ブロックが指定されたバッファーが解放されました。

LUA_NO_PREVIOUS_BID_ENABLED
lua_flag1.bid_enable が指定された **RUI_READ** または **SLI_RECEIVE** verb が / 行される前に、 **RUI_BID** または **SLI_BID** verb が / 行されませんでした。

LUA_NO_DATA
NO_WAIT パラメーターを指定した **RUI_READ** または **SLI_RECEIVE** verb が / 行されましたが、読み h ることのできるデータがありませんでした。

LUA_BID_ALREADY_ENABLED
lua_flag1.bid_enable が指定された **RUI_READ** または **SLI_RECEIVE** verb が / 行されたときに、 **RUI_BID** または **SLI_BID** verb がアクティブになっていました。

LUA_VERB_RECORD_SPANS_SEGMENTS
LUA verb 制御ブロックに含まれている長さパラメーターがセグメントのオフセットに追加されると、セグメントの終わりを超えてしまいます。

LUA_INVALID_FLOW
lua_flag1 流れフラグの設定に誤りのある LUA verb が / 行されました。正しい数の **lua_flag1** 流れフラグが！のように設定されていることを確認してください。

- **RUI_READ** または **SLI_RECEIVE** の場合には、少なくとも 1 つ。
- **RUI_WRITE** の場合には 1 つだけ。
- **SLI_SEND** の場合、 SNA ～答の送信時に **lua_flag1** 流れフラグが 1 つだけ設定されていなければなりません。

LUA_NOT_ACTIVE
アプリケーション・プログラムが LUA verb を / 行したときに、その LUA が Communications Server 内でアクティブになっていませんでした。

LUA_VERB_LENGTH_INVALID
誤った **lua_verb_length** パラメーターが指定された verb が / 行されました。指定された長さは、LUA が予期した長さと異なっています。

LUA_REQUIRED_FIELD_MISSING
/ 行された **RUI_WRITE** verb にデータ・ポインターが組み込まれていない (データ・カウントがゼロではない場合) か、あるいは **lua_flag1** 流れフラグが組み込まれていませんでした。

LUA_READY
SLI セッションが追加コマンドを処理できるようになっています。この状況は、前の NOT_READY 状況が u 信された後で、または **SLI_CLOSE** verb が完了して 1 ！戻りコード CANCELED および 2 ！戻りコード RECEIVE_UNBIND_HOLD または RECEIVED_UNBIND_NORMAL が戻された後で / 行されます。

LUA_NOT_READY
！のいずれかの理由により、SLI セッションが一時的に中断しています。

- CLEAR コマンドが u 信された。 SDT コマンドを u 信すると SLI セッションが再開します。
- UNBIND コマンドが u 信された。このセッションは、BIND、任意選択の STSN および SDT コマンドが u 信されるまで中断状態になります。オリジナルの **SLI_OPEN** verb によって提供されたユーザー拡張ルーチンは再び呼び出されるため、これらのルーチンは再入可能でなければなりません。SLI が SDT コマンドを処理した後で、SLI セッションが再開します。UNBIND コマンドには、！の 2 つのタイプがあります。
  - UNBIND タイプ X'02' は、新 , BIND が行われようとしていることを意味します。
  - UNBIND タイプ X'01' は、このセッションを開始した **SLI_OPEN** verb で、アプリケーションが **lua_session_type** として LUA_SESSION_TYPE_DEDICATED を指定していたことを意味します。

LUA_INIT_COMPLETE
SLI_OPEN の処理中に LUA インターフェースがセッションを初期化しました。この状況は、 **LUA_INIT_TYPE_PRIM_SSCP** パラメーターを指定して **SLI_OPEN** を / 行した LUA アプリケーションの、 **SLI_RECEIVE** または **SLI_BID** verb で戻されます。

LUA_SESSION_END_REQUESTED
SLI がホストから SHUTD コマンドを u 信しました。これは、ホストがセッションをシャットダウンする準 w ができていることを意味します。

LUA_NO_SLI_SESSION
セッションがオープンしていなかったとき、または **SLI_CLOSE** verb またはセッション障害が原因でセッションがダウンしていたときに、コマンドが / 行されました。以下の場合には、**SLI_OPEN** verb の処理中に **SLI_RECEIVE** または **SLI_SEND** verb を / 行すると、このコードが戻されます。

- **SLI_OPEN lua_init_type** パラメーターが LUA_INIT_TYPE_PRIM_SSCP に設定されていない。このような状況では、SLI_BID verb もこのコードを戻します。

- **SLI_RECEIVE** または **SLI_SEND** の **lua_flag1** パラメーターで **lua_flag1.sscp_norm** が指定されていない。

SLI 構成要素は、 UNBIND タイプ X'02' コマンドまたは UNBIND タイプ X'01' (LUA_SESSION_TYPE_DEDICATED) が u 信された後で、 SDT コマンドが処理されるまで、**SLI_OPEN** 処理中になっています。 UNBIND タイプ X'02' は、新 , BIND が行われようとしていることを意味します。

LUA_SESSION_ALREADY_OPEN
すでにセッションがオープンしている LU 名に関して、 **SLI_OPEN** verb が / 行されました。

LUA_INVALID_OPEN_INIT_TYPE
SLI_OPEN verb の lua_init_type パラメーターに誤った値が指定されています。

LUA_INVALID_OPEN_DATA
INITSELF (LUA_INIT_TYPE_SEC_IS) による 2 ！初期化用に lua_init_type パラメーターを指定された **SLI_OPEN** verb が / 行されましたが、データ・バッファーに有効な INITSELF コマンドが含まれていません。

LUA_UNEXPECTED_SNA_SEQUENCE
SLI_OPEN 処理中にホストから予期しないコマンドまたはデータを u 信しました。

LUA_NEG_RSP_FROM_BIND_ROUTINE
ユーザーが提供した **SLI_BIND** ルーチンが、 BIND に対して] 定 ~ 答を生成しました。 **SLI_OPEN** verb は正常に終了しませんでした。

LUA_NEG_RSP_FROM_CRV_ROUTINE
ユーザーが提供した **SLI_BIND** ルーチンが、 BIND に対して] 定 ~ 答を生成しました。 **SLI_OPEN** verb は正常に終了しませんでした。

LUA_NEG_RSP_FROM_STSN_ROUTINE
ユーザーが提供した SLI STSN ルーチンが STSN に対して] 定 ~ 答しました。 **SLI_OPEN** が正常に終了しませんでした。

LUA_CRV_ROUTINE_REQUIRED
ユーザーが SLI CRV ルーチンを提供しませんでしたが、ホストから CRV を u 信しました。 SLI が CRV に対して] 定 ~ 答を / 行し、 **SLI_OPEN** verb がその時点で : 敗して終了します。

LUA_NEG_RSP_FROM_SDT_ROUTINE
ユーザーが提供した SLI SDT ルーチンが、SDT に対して] 定 ~ 答を生成しました。この条 o が原因で **SLI_OPEN** verb が終了します。

LUA_INVALID_OPEN_ROUTINE_TYPE
SLI_OPEN 拡張ルーチン・リストで、 **lua_open_routine_type** パラメーターが無効になっています。

LUA_MAX_NUMBER_OF_SENDS
アプリケーション・プログラムが、1 つの **SLI_SEND** verb が完了する前に、2 つ以上の **SLI_SEND** verb を / 行しました。

LUA_SEND_ON_FLOW_PENDING
すでに未解 h の **SLI_SEND** verb がある SNA 流れ (SSCP ^ 送、SSCP 通常、LU ^ 送、LU 通常) に関して、アプリケーションが **SLI_SEND** verb を / 行しました。

LUA_INVALID_MESSAGE_TYPE
SLI が **lua_message_type** パラメーターを認 1 しません。

LUA_RECEIVE_ON_FLOW_PENDING
すでに未解 h の **SLI_RECEIVE** verb がある SNA 流れに関して、 SLI アプリケーションが **SLI_RECEIVE** verb を / 行しました。

LUA_DATA_LENGTH_ERROR
アプリケーション・プログラムによって提供されないユーザー・データを必要とする、 **SLI_OPEN** コマンドが / 行されました。 2 V 目に開始された **SLI_OPEN** verb にはデータが必要で、アプリケーションが LUSTAT コマンド対 ~ して **SLI_SEND** verb を / 行する場合には 4 バイトの状況が必要です。

LUA_CLOSE_PENDING
！のいずれかが / 生しています。

- CLOSE_NORMAL または CLOSE_ABEND の保留中に CLOSE_NORMAL が / 行されました。
- CLOSE_ABEND の保留中に別の CLOSE_ABEND が / 行されました。別の CLOSE_ABEND の / 行が有効なのは、CLOSE_NORMAL が保留されているときだけです。

LUA_NEGATIVE_RSP_CHASE
SLI_CLOSE 処理中に、 SLI がホストからの CHASE コマンドに対する] 定 ~ 答を u 信しました。セッションは、**SLI_CLOSE** の要 a どおりに停止します。

LUA_NEGATIVE_RSP_SHUTC
SLI_CLOSE 処理中に、 SLI がホストからの SHUTC コマンドに対する] 定 ~ 答を u 信しました。セッションは、**SLI_CLOSE** の要 a どおりに停止します。

LUA_NEGATIVE_RSP_SHUTD
SLI_CLOSE 処理中に、 SLI がホストからの SHUTD コマンドに対する] 定 ~ 答を u 信しました。セッションは、**SLI_CLOSE** の要 a どおりに停止します。

LUA_NO_RECEIVE_TO_PURGE
未解 h の **SLI_RECEIVE** verb がないときに **SLI_PURGE** verb が / 行されました。！の 2 つの原因が考えられます。

- **lua_data_ptr** パラメーターで指定されたアドレスが、パージ対象の未解 h **SLI_RECEIVE** verb を指していなかった。
- **SLI_PURGE** verb の処理中に **SLI_RECEIVE** verb が完了していた。これはエラー条 o ではありません。この状態を処理できるようにアプリケーション・プログラムをコーディングしてください。

LUA_CANCEL_COMMAND_RECEIVED
SLI_RECEIVE verb の処理中に、ホストが、 u 信しているデータのチェーンを h り消すために CANCEL コマンドを送信しました。

LUA_RUI_WRITE_FAILURE
RUI_WRITE verb が、予期しないエラーを SLI に通知しました。

LUA_INVALID_SESSION_TYPE
SLI_OPEN verb の **lua_session_type** に、無効な値が含まれています。

LUA_SLI_BID_PENDING
前に / 行した **SLI_BID** がアクティブになっているときに、 SLI verb が / 行されました。アクティブにできる **SLI_BID** は、一度に 1 つだけです。

LUA_PURGE _PENDING
前に / 行した **SLI_PURGE** がアクティブになっているときに、 **SLI_PURGE** verb が / 行されました。アクティブにできる **SLI_PURGE** は、一度に 1 つだけです。

LUA_PROCEDURE_ERROR
ホスト・プロシージャー・エラーが / こっていることを (す、NSPE または NOTIFY メッセージを u 信しました。 (**SLI_OPEN** verb 再試行オプションが使用されていない限り) **SLI_OPEN** にはこの戻りコードが通知されます。 **lua_wait** が s ゼロ値に設定されている場合には、ホスト・プロシージャーが使用可能になるかあるいはアプリケーションが **SLI_CLOSE** を / 行するまで、 INITSELF または LOGON メッセージが再試行されます。

LUA_INVALID_SLI_ENCR_OPTION
SLI_OPEN verb の **lua_encr_decr_option** パラメーターが 128 に設定されていました。 SLI は、暗号化または暗号化解除処理オプションとして 128 をサポートしません。

LUA_RECEIVED_UNBIND
アクティブな SLI があるときに、SLI が 1 ！ LU から UNBIND コマンドを u 信しました。この SLI セッションは停止します。

LUA_RECEIVED_UNBIND_HOLD
1 ！または 2 ！開始の **SLI_CLOSE** 通常処理中に、 SLI が UNBIND タイプ X'02' を u 信しました。タイプ X'02' は、新 , BIND が行われようとしていることを意味します。このセッションは、BIND、任意選択の CRV および STSN、および SDT コマンドが u 信されるまで中断状態になります。オリジナルの **SLI_OPEN** verb によって提供されたユーザー拡張ルーチンは再び呼び出されます。これらのルーチンは再入可能でなければなりません。 SLI が SDT コマンドを処理した後で、SLI セッションが再開します。

LUA_RECEIVED_UNBIND_NORMAL
lua_session_type として LUA_SESSION_TYPE_DEDICATED が指定された **SLI_OPEN** verb で開始されたセッションに関する、 1 ！または 2 ！開始の **SLI_CLOSE** 通常処理中に、 SLI が UNBIND タイプ X'01' を u 信しました。このセッションは、BIND、任意選択の STSN および SDT コマンドが u 信されるまで中断状態になります。オリジナルの **SLI_OPEN** verb によって提供されたユーザー拡張ルーチンは再び呼び出されます。これらのルーチンは再入可能でなければなりません。 SLI が SDT コマンドを処理した後で、SLI セッションが再開します。

LUA_SLI_LOGIC_ERROR
SLI が内部論理エラーを検出しました。

LUA_TERMINATED

SLI_CLOSE または **RUI_TERM** verb の / 行時に保留されていた verb が h り消されました。

LUA_NO_RUI_SESSION
(**RUI_INIT** によって) 開始されていないセッションに関して RUI verb が / 行されたか、あるいはそのセッションに関する **RUI_INIT** verb の進行中に、**RUI_TERM** 以外の verb が / 行されました。

この戻りコードは、未解 h のアクティブな RUI verb がない時点でセッション障害が / こったときに生成されることがあります。！に verb を / 行すると、この戻りコードを u け h ります。アプリケーション・プログラムはこの戻りコードを SESSION_FAILURE と同じように扱います。

LUA_DUPLICATE_RUI_INIT
すでに初期化されているセッションまたは **RUI_INIT** verb が進行中のセッションに関して、アプリケーション・プログラムが **RUI_INIT** verb を / 行しました。

LUA_INVALID_PROCESS
すでに別のプロセスに所有されているセッションに関して RUI verb が / 行されました。

LUA_API_MODE_CHANGE
SLI によって設定されたセッションで s SLI 要 a が / 行されました。

LUA_COMMAND_COUNT_ERROR
最大数を超える数の **RUI_READ** または **RUI_WRITE** verb が / 行されたか、あるいは前に / 行された **RUI_BID** または **RUI_TERM** verb がまだ進行しているときに **RUI_BID** または **RUI_TERM** verb が / 行されました。

LUA_NO_READ_TO_PURGE
未解 h の **RUI_READ** verb がないときに **RUI_PURGE** verb が / 行されました。！の 2 つの原因が考えられます。

- **lua_data_ptr** パラメーターで指定されたアドレスが、パージ対象の未解 h **RUI_READ** verb を指していない。
- **RUI_PURGE** verb の処理中に **RUI_READ** verb が完了した。これはエラー条 o ではありません。この状態を処理できるようにアプリケーション・プログラムをコーディングしてください。

LUA_MULTPLE_WRITE_FLOWS
RUI_WRITE verb に対して / 行された FLAG1 で、複数の流れフラグがオンになっていました。

LUA_DUPLICATE_READ_FLOW
すでに **RUI_READ** が保留されている流れに関して、アプリケーション・プログラムが **RUI_READ** を / 行しました。

LUA_DUPLICATE_WRITE_FLOW
/ 行された **RUI_WRITE** verb に、前に / 行されてまだ完了していない **RUI_WRITE** verb に関するセッション流れを (す FLAG1 流れフラグが含まれていました。

LUA_LINK_NOT_STARTED
LUA がセッション初期化中にデータ・リンクを開始できませんでした。

LUA_INVALID_ADAPTER
DLC アダプターの構成が誤っているか、あるいは構成ファイルが損傷を u けています。

LUA_ENCR_DECR_LOAD_ERROR
ユーザーが提供した暗号化または暗号化解除ダイナミック・リンク・ライブラリーをロードしようとしたときに、予期しないエラーを u 信しました。

LUA_ENCR_DECR_PROC_ERROR
ユーザーが提供した暗号化または暗号化解除ダイナミック・リンク・ライブラリーを獲得しようとしたときに、予期しないエラーを u 信しました。

LUA_LINK_NOT_STARTED_RETRY
リンクをアクティブにすることができないため、 **RUI_INIT** または **SLI_OPEN** verb が : 敗しました。この戻りコードは、パートナー・ロケーションまたは 2 つのマシン間の接続に何らかの障害があることを暗 (しています。

LUA_NEG_NOTIFY_RSP
RUI_INIT が / 行され、 SLU をセッションに参加させられるようになったことを (す通知要 a が SSCP に送信されました。 SSCP は、この通知要 a に対して] 定 ~ 答しました。予定されていたハーフセッション構成要素はサポートされる要 a を認 1 しましたが、その要 a を処理しませんでした。

LUA_RUI_LOGIC_ERROR
RUI 内部論理エラーが / 生しました。

LUA_LU_INOPERATIVE
SLI がセッションを停止させようとしていたときに、重大エラーが / 生しました。この LU は、ホストから ACTLU を u 信するまでは LUA 要 a の処理には使用できません。

LUA_RESOURCE_NOT_AVAILABLE
RU で指定された LU、PU、リンク・ステーション、またはリンクが使用できません。 **SLI_OPEN** 再試行オプションが使用されていない限り、 **SLI_OPEN** にはこの戻りコードが通知されることがあります。 **lua_wait** が s ゼロ値に設定されている場合には、ホスト・プロシージャーが使用可能になるかあるいはアプリケーションが **SLI_CLOSE** verb を / 行するまで、 INITSELF または LOGON メッセージが再試行されます。

LUA_SESSION_LIMIT_EXCEEDED
ネットワーク・アドレス単位 (NAU) のうちの 1 つがセッション限度 (たとえば、 LU-LU セッション限度または LU モード・セッション限度) に達しているため、要 a されたセッションをアクティブにすることができません。このセンス・コードは ACTCDRM、INIT、BID、および CINIT 要 a に適用されます。

SLI_OPEN verb 再試行オプションが使用されていない限り、 **SLI_OPEN** にはこの戻りコードが通知されることがあります。 **lua_wait** が s ゼロ値に設定されている場合には、ホスト・プロシージャーが使用可能になるかあるいはアプリケーションが **SLI_CLOSE** verb を / 行するまで、 INITSELF または LOGON メッセージが再試行されます。

LUA_SLU_SESSION_LIMIT_EXCEEDED
この要 a が u け入れられると SLU セッション限度を超えます。

LUA_MODE_INCONSISTENCY
現在の状況では、このファンクションを B 行することは v 可されません。予定されていたハーフセッション構成要素はサポートされる要 a を認 1 しましたが、その要 a を処理しませんでした。このコードは、EXR でセンス・コードとして表 (されることもあります。

LUA_INSUFFICIENT_RESOURCES
リソースが一時的に不足しているため、 u 信側が要 a に対 ~ できません。予定されていたハーフセッション構成要素はサポートされる要 a を認 1 しましたが、その要 a を処理しませんでした。

LUA_RECEIVER_IN_TRANSMIT_MODE
競争条 o が存在します。 > 二重コンテンション状態が「 s u 信」になっているとき、または通常流れデータの処理に必要な (バッファーなどの) リソースが使用できないときに、通常流れ要 a を u 信しました。

このコードは、例外要 a でセンス・コードとして表 (されることもあります。

LUA_LU_COMPONENT_DISCONNECTED
電源オフその他の切断状態になっているために、LU 構成要素が使用できません。

LUA_NEGOTIABLE_BIND_ERROR
交渉可能 BIND を u 信しました。 **SLI_OPEN** verb を介してユーザーが **SLI_BIND** ルーチンを提供していない限り、 SLI は交渉可能 BIND を v 可しません。

LUA_BIND_FM_PROFILE_ERROR
サポートされない FM プロファイルが BIND で検出されました。 SLI は FM プロファイル 3 および 4 だけをサポートします。

LUA_BIND_TS_PROFILE_ERROR
サポートされない TS プロファイルが BIND で検出されました。 SLI は TS プロファイル 3 および 4 だけをサポートします。

LUA_BIND_LU_TYPE_ERROR
サポートされない LU タイプが検出されました。 LUA は LU 0、LU 1、LU 2 および LU 3 だけをサポートします。

LUA_SSCP_LU_SESSION_NOT_ACTIVE
要 a を処理するために必要な SSCP-LU セッションがアクティブになっていません。たとえば、INITSELF 要 a の処理を行っている場合、 INITSELF で指定されたターゲット LU と SSCP の間にアクティブ・セッションがありません。

バイト 2 および 3 にセンス・コード特定情報が含まれています。以下の設定が可能です。

0000 特定コードは適用されません。

0001 SSCP-SLU セッションの再活動化中です。

0002 SSCP-PLU セッションが s アクティブになっています。 **SLI_OPEN** 再試行オプションが使用されていない限り、 **SLI_OPEN** にはこの戻りコードが通知されることがあります。 **lua_wait** が s ゼロ値に設定されている場合には、ホスト・プロシージャーが使用可能になるかあるいはアプリケーションが **SLI_CLOSE** verb を / 行するまで、INITSELF または LOGON メッセージが再試行されます。

0003 SSCP-SLU セッションが s アクティブになっています。

0004 SSCP-SLU セッションの再活動化中です。

LUA_REC_CORR_TABLE_FULL

要 a された流れに関するセッション u 信相関テーブルが、容量の限界に達しました。

LUA_SEND_CORR_TABLE_FULL

要 a された流れに関する送信相関テーブルが、容量の限界に達しました。

LUA_SESSION_SERVICES_PATH_ERROR

セッション・サービス要 a を SSCP-SSCP セッションのパス P 由で転送できません。たとえば、ネットワーク間 LU-LU セッションを設定するために、この！能が必要です。

バイト 2 および 3 にセンス・コード特定情報が含まれています。以下の設定が可能です。

0000 特定コードは適用されません。 **SLI_OPEN** 再試行オプションが使用されていない限り、 **SLI_OPEN** にはこの戻りコードが通知できません。 **lua_wait** が s ゼロ値に設定されている場合には、ホスト・プロシージャーが使用可能になるかあるいはアプリケーションが **SLI_CLOSE** を / 行するまで、 INITSELF または LOGON メッセージが再試行されます。

0001 SSCP が 1 つまたは複数の隣接 SSCP P 由でセッション・サービス要 a を宛先に転送しようとしましたが、：敗しました。この値は、ゲートウェイ SSCP が試行錯誤再 P 路指定を使い尽くしたときに、ゲートウェイ SSCP によって送信されます。

宛先 SSCP への再 P 路指定が完全に：敗しました。 SSCP が特定の SSCP への再 P 路指定を試みましたが、：敗しました。たとえば、再 P 路指定を試みていたノードで再 P 路指定障害に関する情報が表 (された場合には、このコードは特定の SSCP と関連しています。

0002

必要なルーティング・テーブルが利用不能なため (つまり、リソース ID 制御ベクトル内の転送キーに対～する隣接 SSCP テーブルがないため)、 SSCP がセッション・サービス要 a を転送できません。

0003

この SSCP では、LU に関する事前定Aが行われていませんが、隣接 SSCP がパートナー SSCP 内のダイナミック定Aをサポートしません。したがって、この SSCP は、LU を動的に定Aすることも、その隣接 SSCP に転送することもできません。

0005

再試行されました。

0006

再試行されました。

0008

隣接 SSCP が要 a された CDINIT ファンクション (たとえば、リソース可用性または XRF の通知) をサポートしません。

000A

セッション・サービス要 a が同じ SSCP を 2 回通っているため、SSCP がその要 a を転送できません。

000B

CDINIT で指定された DLU が u 信側 SSCP で認 1 されないため、 u 信側 SSCP が CDINIT を転送できません。

LUA_RU_LENGTH_ERROR

要 a された RU が長すぎるか、あるいは短すぎます。予定されたハーフセッション構成要素に RU が送達されましたが、それを解 a または処理することができませんでした。この条 o は、ハーフセッション能力のミスマッチを表しています。

このコードは、EXR でセンス・コードとして表 (されることもあります。

LUA_FUNCTION_NOT_SUPPORTED

要 a されたファンクションが LUA でサポートされません。このファンクションは、定様 0 要 a コード、RU 内のパラメーター、または制御文字で指定されたものです。

センス・コードの後のバイト 2 および 3 は、ユーザー定Aデータの場合には使用されません。これらのバイトには、センス・コード特定情報が含まれています。 ! の設定が可能です。

0000 要 a されたファンクションが LUA でサポートされません。

予定されたハーフセッション構成要素に RU が送達されましたが、それを解 a または処理することができませんでした。この条 o は、ハーフセッション能力のミスマッチを表しています。

LUA_HDX_BRACKET_STATE_ERROR

プロトコル・マシンが、既存の状態エラーのもとでは現在の要 a を送信できないことを = 別しました。

LUA_RESPONSE_ALREADY_SENT

プロトコル・マシンが、チェーンに関する ~ 答がすでに送信されているために、現在の要 a を送信できないことを = 別しました。

LUA_EXR_SENSE_INCORRECT

アプリケーションが、すでに u 信された例外要 a に関して] 定 ~ 答を / 行しました。その ~ 答のセンス・コードは u け入れ不能でした。

例外要 a 内のセンス・コードは X'0813000' であり、] 定 ~ 答内のセンス・コードは X'08130000' または X'08140000' のいずれかです。それ以外のすべての場合、] 定 ~ 答内のセンス・コードは例外要 a 内のセンス・コードと同じでなければなりません。

LUA_RESPONSE_OUT_OF_ORDER

プロトコル・マシンが、最も古い要 a に対して現在の ~ 答が送信されないことを = 別しました。

LUA_CHASE_RESPONSE_REQUIRED

プロトコル・マシンが、最も古い未解 h の CHASE 要 a で現在の要 a が試みられていることを = 別しました。

LUA_CATEGORY_NOT_SUPPORTED

DFC、SC、NC、または FMD 要 a が、そのようなカテゴリーの要 a をサポートしないハーフセッションによって u 信されたか、ネットワーク・サービス (NS) 要 a のバイト 0 が、定 A された値に設定されていなかったか、あるいはバイト 1 が、 u 信側によって NS カテゴリーに設定されていませんでした。

LUA_CHAINING_ERROR

連鎖標 1 設定値の順序に、first、middle、first などのような誤りがありました。 u 信側の現行セッション制御状態またはデータ・フロー制御状態では v 可されない要 a ヘッダーまたは要 a 単位が検出されました。このエラーにより、意図されたハーフセッション構成要素に要 a が送達できませんでした。

LUA_BRACKET

送信側が、セッションのブラケット , 則を適用しませんでした。 u 信側の現行セッション制御状態またはデータ・フロー制御状態では v 可されない要 a ヘッダーまたは要 a 単位が検出されました。このエラーにより、意図されたハーフセッション構成要素に要 a が送達できませんでした。

LUA_DIRECTION

＞二重フリップフロップ状態が NOT_RECEIVE になっているときに、通常流れ要 a を u 信しました。 u 信側の現行セッション制御状態またはデータ・フロー制御状態では v 可されない要 a ヘッダーまたは要 a 単位が検出されました。このエラーにより、意図されたハーフセッション構成要素に要 a が送達できませんでした。

LUA_DATA_TRAFFIC_RESET

FMD または通常流れ DFC 要 a がハーフセッションによって u 信されました。このハーフセッションのセッション活動化状態はアクティブですが、データ・トラフィック状態はアクティブになっていませんでした。 u 信側の現行セッション制御状態またはデータ・フロー制御状態では v 可されない要 a ヘッダーまたは要 a 単位が検出されました。このエラーにより、意図されたハーフセッション構成要素に要 a が送達できませんでした。

LUA_DATA_TRAFFIC_QUIESCED

QC コマンドまたは SHUTC コマンドを送信したハーフセッションから u 信した FMD または DFC 要 a が、 RELQ コマンドに対して ~ 答していませんでした。 u 信側の現行セッション制御状態またはデータ・フロー制御状態で

は v 可されない ~ 答ヘッダーまたは要 a 単位が検出されました。このエラーにより、意図されたハーフセッション構成要素に要 a が送達できませんでした。

LUA_DATA_TRAFFIC_NOT_RESET
データ通信量状態がリセットになっていないときに、セッション制御要 a を u 信しました。 u 信側の現行セッション制御状態またはデータ・フロー制御状態では v 可されない要 a ヘッダーまたは要 a 単位が検出されました。このエラーにより、意図されたハーフセッション構成要素に要 a が送達できませんでした。

LUA_NO_BEGIN_BRACKET
u 信側が、BIS コマンドに対して肯定 ~ 答を送信した後で、 BBI=BB が指定された BID または FMD 要 a を u 信しました。 u 信側の現行セッション制御状態またはデータ・フロー制御状態では v 可されない要 a ヘッダーまたは要 a 単位が検出されました。このエラーにより、意図されたハーフセッション構成要素に要 a が送達できませんでした。

LUA_SC_PROTOCOL_VIOLATION
SC プロトコルに対する違 ? がありました。 SC 要 a およびそれに関連した肯定 ~ 答が正常に交換された後でなければ v 可されない要 a が、正常な交換が行われる前に u 信されました。センス・データのバイト 4 に要 a コードが含まれています。このセンス・コードに関連したユーザー・データはありません。 u 信側の現行セッション制御状態またはデータ・フロー制御状態では v 可されない要 a ヘッダーまたは要 a 単位が検出されました。このエラーにより、意図されたハーフセッション構成要素に要 a が送達できませんでした。

LUA_IMMEDIATE_REQ_MODE_ERROR
この要 a が即時要 a モード・プロトコルに違 ? しました。 u 信側の現行セッション制御状態またはデータ・フロー制御状態では v 可されない RH または RU が検出されました。このエラーにより、意図されたハーフセッション構成要素に要 a が送達できませんでした。

LUA_QUEUED_RESPONSE_ERROR
要 a が待ち行列 ~ 答プロトコルに違 ? しました。たとえば、未解 h の要 a で QRI=QR が指定されているときに QRI=QR になっていませんでした。 u 信側の現行セッション制御状態またはデータ・フロー制御状態では v 可されない RH または RU が検出されました。このエラーにより、意図されたハーフセッション構成要素に要 a が送達できませんでした。

LUA_ERP_SYNC_EVENT_ERROR
ERP 同期イベント・プロトコルに対する違 ? がありました。 u 信側の現行セッション制御状態またはデータ・フロー制御状態では v 可されない RH または RU が検出されました。このエラーにより、意図されたハーフセッション構成要素に要 a が送達できませんでした。

LUA_RSP_BEFORE_SENDING_REQ
前に u 信した要 a に対する ~ 答がまだ送信されていないときに、 > 二重 (フリップフロップまたはコンテンション) 送 u 信モードで通常流れ要 a を送信しようとしました。 u 信側の現行セッション制御状態またはデータ・フロー制御状態では v 可されない RH または RU が検出されました。このエラーにより、意図されたハーフセッション構成要素に要 a が送達できませんでした。

LUA_RSP_CORRELATION_ERROR

前に送信された要 a と相関できない ~ 答が u 信されたか、あるいは前に u 信された要 a と相関できない ~ 答が送信されました。

LUA_RSP_PROTOCOL_ERROR

！のような、~ 答プロトコルに違 ? する ~ 答を 1 ！側ハーフセッションから u 信しました。

- RQE チェーンに関する肯定 ~ 答 (+RSP) を u 信した。
- 1 つのチェーンに関して 2 つの ~ 答を u 信した。

LUA_INVALID_SC_OR_NC_RH

セッション制御 (SC) またはネットワーク制御 (NC) 要 a の RH が無効でした。たとえば、ペーシング要 a 標 1 が 1 に設定された SC RH は無効です。RH 内のパラメーターまたはパラメーターの組み合わせの値が、すでに選択された LOGON オプションのアーキテクチャー , 則に違 ? していました。これらのエラーにより、意図されたハーフセッション構成要素に要 a が送達できませんでした。これらのエラーはセッションの現在の状態とは独立しており、送信側がセッション , 則を適用できなかったことが原因で生成されることがあります。

LUA_BB_NOT_ALLOWED

ブラケット開始標 1 (BB) が、 (たとえば BCI=BC ではない場合の BBI=BB のように) 誤って指定されていました。 RH 内のパラメーターまたはパラメーターの組み合わせの値が、すでに選択された LOGON オプションのアーキテクチャー , 則に違 ? していました。これらのエラーにより、意図されたハーフセッション構成要素に要 a が送達できませんでした。これらのエラーはセッションの現在の状態とは独立しており、送信側がセッション , 則を適用できなかったことが原因で生成されることがあります。

LUA_EB_NOT_ALLOWED

ブラケット終了標 1 (EB) が (たとえば、BCI=BC ではない場合の EBI=EB により、 2 ！側ハーフセッションだけが EB を送信できる場合の 1 ！側ハーフセッションにより、あるいは 1 ！側ハーフセッションだけが EB を送信できる場合の 2 ！側ハーフセッションにより)、誤って指定されていました。RH 内のパラメーターまたはパラメーターの組み合わせの値が、すでに選択された LOGON オプションのアーキテクチャー , 則に違 ? していました。これらのエラーにより、意図されたハーフセッション構成要素に要 a が送達できませんでした。これらのエラーはセッションの現在の状態とは独立しており、送信側がセッション , 則を適用できなかったことが原因で生成されることがあります。

LUA_EXCEPTION_RSP_NOT_ALLOWED

v 可されない場合に例外時 ~ 答が要 a されました。 RH 内のパラメーターまたはパラメーターの組み合わせの値が、すでに選択された LOGON オプションのアーキテクチャー , 則に違 ? していました。これらのエラーにより、意図されたハーフセッション構成要素に要 a が送達できませんでした。これらのエラーはセッションの現在の状態とは独立しており、送信側がセッション , 則を適用できなかったことが原因で生成されることがあります。

LUA_DEFINITE_RSP_NOT_ALLOWED

v 可されない場合に確定 ~ 答が要 a されました。 RH 内のパラメーターまた

はパラメーターの組み合わせの値が、すでに選択された LOGON オプションのアーキテクチャー , 則に違 ? していました。これらのエラーにより、意図されたハーフセッション構成要素に要 a が送達できませんでした。これらのエラーはセッションの現在の状態とは独立しており、送信側がセッション , 則を適用できなかったことが原因で生成されることがあります。

LUA_PACING_NOT_SUPPORTED

要 a でペーシング標 1 が設定されていましたが、 u 信側ハーフセッションまたは境界 ! 能ハーフセッションがこのセッションについてはペーシングをサポートしません。 RH 内のパラメーターまたはパラメーターの組み合わせの値が、すでに選択された LOGON オプションのアーキテクチャー , 則に違 ? していました。これらのエラーにより、意図されたハーフセッション構成要素に要 a が送達できませんでした。これらのエラーはセッションの現在の状態とは独立しており、送信側がセッション , 則を適用できなかったことが原因で生成されることがあります。

LUA_CD_NOT_ALLOWED

方向転換標 1 (CD) が、たとえば、ECI=EC ではない場合の CDI=CD や EBI=EB の場合の CDI=CD のように、誤って指定されていました。 RH 内のパラメーターまたはパラメーターの組み合わせの値が、すでに選択された LOGON オプションのアーキテクチャー , 則に違 ? していました。これらのエラーにより、意図されたハーフセッション構成要素に要 a が送達できませんでした。これらのエラーはセッションの現在の状態とは独立しており、送信側がセッション , 則を適用できなかったことが原因で生成されることがあります。

LUA_NO_RESPONSE_NOT_ALLOWED

v 可されない場合に要 a で無 ~ 答が指定されました。無 ~ 答は EXR だけで使用されます。 RH 内のパラメーターまたはパラメーターの組み合わせの値が、すでに選択された LOGON オプションのアーキテクチャー , 則に違 ? していました。これらのエラーにより、意図されたハーフセッション構成要素に要 a が送達できませんでした。これらのエラーはセッションの現在の状態とは独立しており、送信側がセッション , 則を適用できなかったことが原因で生成されることがあります。

LUA_CHAINING_NOT_SUPPORTED

連鎖標 1 (BCI と ECI) が誤って指定されていました (たとえば、このセッションまたは要 a ヘッダーで指定されたカテゴリーには複数要 a チェーンがサポートされていないにもかかわらず、 BCI=BC および ECI=EC 以外のチェーニング・ビットが表 (されていました)。 RH 内のパラメーターまたはパラメーターの組み合わせの値が、すでに選択された LOGON オプションのアーキテクチャー , 則に違 ? していました。これらのエラーにより、意図されたハーフセッション構成要素に要 a が送達できませんでした。これらのエラーはセッションの現在の状態とは独立しており、送信側がセッション , 則を適用できなかったことが原因で生成されることがあります。

LUA_BRACKETS_NOT_SUPPORTED

ブラケット標 1 (BBI および EBI) が誤って指定されていました (たとえば、このセッションではブラケットが使用されないにもかかわらず、ブラケット標 1 (BBI=BB または EBI=EB) が設定されていました)。 RH 内のパラメーターまたはパラメーターの組み合わせの値が、すでに選択された LOGON オ

プションのアーキテクチャー , 則に違 ? していました。これらのエラーにより、意図されたハーフセッション構成要素に要 a が送達できませんでした。これらのエラーはセッションの現在の状態とは独立しており、送信側がセッション , 則を適用できなかったことが原因で生成されることがあります。

LUA_CD_NOT_SUPPORTED

方向転換標 1 が設定されていますが、これはサポートされません。 RH 内のパラメーターまたはパラメーターの組み合わせの値が、すでに選択された LOGON オプションのアーキテクチャー , 則に違 ? していました。これらのエラーにより、意図されたハーフセッション構成要素に要 a が送達できませんでした。これらのエラーはセッションの現在の状態とは独立しており、送信側がセッション , 則を適用できなかったことが原因で生成されることがあります。

LUA_INCORRECT_USE_OF_FI

A 0 標 1 (FI) が誤って指定されていました (たとえば、 FI が BCI=BC 以外で設定されていたか、あるいは FI が DFC 要 a で設定されていませんでした)。 RH 内のパラメーターまたはパラメーターの組み合わせの値が、すでに選択された LOGON オプションのアーキテクチャー , 則に違 ? していました。これらのエラーにより、意図されたハーフセッション構成要素に要 a が送達できませんでした。これらのエラーはセッションの現在の状態とは独立しており、送信側がセッション , 則を適用できなかったことが原因で生成されることがあります。

LUA_ALTERNATE_CODE_NOT_SUPPORTED

セッションでサポートされない場合にコード選択標 1 (CSI) が設定されていました。 RH 内のパラメーターまたはパラメーターの組み合わせの値が、すでに選択された LOGON オプションのアーキテクチャー , 則に違 ? していました。これらのエラーにより、意図されたハーフセッション構成要素に要 a が送達できませんでした。これらのエラーはセッションの現在の状態とは独立しており、送信側がセッション , 則を適用できなかったことが原因で生成されることがあります。

LUA_INCORRECT_RU_CATEGORY

RU カテゴリー標 1 が誤って指定されていました (たとえば、 RU カテゴリー標 1 が FMD になっているときに ^ 送フロー要 a または ~ 答が指定されていました)。 RH 内のパラメーターまたはパラメーターの組み合わせの値が、すでに選択された LOGON オプションのアーキテクチャー , 則に違 ? していました。これらのエラーにより、意図されたハーフセッション構成要素に要 a が送達できませんでした。これらのエラーはセッションの現在の状態とは独立しており、送信側がセッション , 則を適用できなかったことが原因で生成されることがあります。

LUA_INCORRECT_REQUEST_CODE

~ 答の要 a コードが、対 ~ する要 a の要 a コードと一致していません。 RH 内のパラメーターまたはパラメーターの組み合わせの値が、すでに選択された LOGON オプションのアーキテクチャー , 則に違 ? していました。これらのエラーにより、意図されたハーフセッション構成要素に要 a が送達できませんでした。これらのエラーはセッションの現在の状態とは独立しており、送信側がセッション , 則を適用できなかったことが原因で生成されることがあります。

LUA_INCORRECT_SPEC_OF_SDI_RTI

sense-data-included 標 1 (SDI) および ~ 答タイプ標 1 (RTI) が ~ 答で正しく指定されていませんでした。正しい値の組は、 (SDI=SD、RTI=] 定) および (SDI=SD 以外、RTI=肯定) です。 RH 内のパラメーターまたはパラメーターの組み合わせの値が、すでに選択された LOGON オプションのアーキテクチャー , 則に違 ? していました。これらのエラーにより、意図されたハーフセッション構成要素に要 a が送達できませんでした。これらのエラーはセッションの現在の状態とは独立しており、送信側がセッション , 則を適用できなかったことが原因で生成されることがあります。

LUA_INCORRECT_DR1I_DR2I_ERI

確定 ~ 答 1 標 1 (DR1I)、確定 ~ 答 2 標 1 (DR2I)、および例外 ~ 答標 1 (ERI) が誤って指定されていました。たとえば、CANCEL 要 a が DR1I=DR1、DR2I=DR2 以外、および ERI=ER 以外を用いて指定されていませんでした。 RH 内のパラメーターまたはパラメーターの組み合わせの値が、すでに選択された LOGON オプションのアーキテクチャー , 則に違 ? していました。これらのエラーにより、意図されたハーフセッション構成要素に要 a が送達できませんでした。これらのエラーはセッションの現在の状態とは独立しており、送信側がセッション , 則を適用できなかったことが原因で生成されることがあります。

LUA_INCORRECT_USE_OF_QRI

待ち行列 ~ 答標 1 (QRI) が誤ってされていました (たとえば、 ^ 送フロー要 a で QRI=QR が指定されていました)。 RH 内のパラメーターまたはパラメーターの組み合わせの値が、すでに選択された LOGON オプションのアーキテクチャー , 則に違 ? していました。これらのエラーにより、意図されたハーフセッション構成要素に要 a が送達できませんでした。これらのエラーはセッションの現在の状態とは独立しており、送信側がセッション , 則を適用できなかったことが原因で生成されることがあります。

LUA_INCORRECT_USE_OF_EDI

暗号化データ標 1 (EDI) が誤ってされていました (たとえば、 DFC 要 a で EDI=ED が指定されていました)。 RH 内のパラメーターまたはパラメーターの組み合わせの値が、すでに選択された LOGON オプションのアーキテクチャー , 則に違 ? していました。これらのエラーにより、意図されたハーフセッション構成要素に要 a が送達できませんでした。これらのエラーはセッションの現在の状態とは独立しており、送信側がセッション , 則を適用できなかったことが原因で生成されることがあります。

LUA_INCORRECT_USE_OF_PDI

埋め込みデータ標 1 (PDI) が誤ってされていました (たとえば、 DFC 要 a で PDI=PD が指定されていました)。 RH 内のパラメーターまたはパラメーターの組み合わせの値が、すでに選択された LOGON オプションのアーキテクチャー , 則に違 ? していました。これらのエラーにより、意図されたハーフセッション構成要素に要 a が送達できませんでした。これらのエラーはセッションの現在の状態とは独立しており、送信側がセッション , 則を適用できなかったことが原因で生成されることがあります。

LUA_NAU_INOPERATIVE

NAU が要 a または ~ 答を処理できません。たとえば、NAU が異常終了によって破壊されました。パスの障害、活動化要 a 順序の誤り、またはリストさ

れたパス情報単位 (PIU) のうちの 1 つの誤りが原因で、意図した u 信側に要 a を送達することができませんでした。セッションがアクティブになっているときにパス・エラーを u 信した場合、一 L には、セッション相 j 側へのパスが : われています。

LUA_NO_SESSION

指 (された / 点宛先のペアについて u 信側エンド・ノードでアクティブになっているハーフセッションがないか、境界 ! 能を提供するノードで / 点宛先のペアについてアクティブになっている境界 ! 能ハーフセッション構成要素がありません。セッション活動化要 a が必要です。パスの障害または活動化要 a 順序の誤りが原因で、意図した u 信側に要 a を送達することができませんでした。セッションがアクティブになっているときにパス・エラーを u 信した場合、一 L には、セッション相 j 側へのパスが : われています。

LUA_BRACKET_RACE_ERROR

ブラケット・プロトコルでコンテンションが : われました。両方の NAU によるブラケット開始またはブラケット終了が行われると、コンテンションは : われます。予定されていたハーフセッション構成要素はサポートされる要 a を認 1 しましたが、その要 a を処理しませんでした。

LUA_BB_REJECT_NO_RTR

ファースト・スピーカーがブラケット内状態になっているとき、またはファースト・スピーカーがブラケット間状態になっているときに、 BID またはブラケット開始標 1 を u 信しました。ファースト・スピーカーが v 可を q] しました。 RTR コマンドは送信されません。予定されていたハーフセッション構成要素はサポートされる要 a を認 1 しましたが、その要 a を処理しませんでした。

LUA_CRYPTOGRAPHY_INOPERATIVE

暗号化ファシリティーで誤動作が / 生したために、要 a の u 信側が要 a を復号できませんでした。予定されていたハーフセッション構成要素はサポートされる要 a を認 1 しましたが、その要 a を処理しませんでした。

LUA_SYNC_EVENT_RESPONSE

同期化要 a に対する] 定 ~ 答を u 信しました。予定されていたハーフセッション構成要素はサポートされる要 a を認 1 しましたが、その要 a を処理しませんでした。

LUA_RU_DATA_ERROR

要 a RU 内のデータが、 u 信側 FMDS 構成要素で u け入れ不能でした。たとえば、サポートされるセットに文字コードが含まれていなかったか、定様 0 データ・パラメーターが表 (サービスで u け入れ不能であったか、あるいは要 a 内の必須名が省略されていました。予定されたハーフセッション構成要素に RU が送達されましたが、それを解 a または処理することができませんでした。この条 o は、ハーフセッション能力のミスマッチを表しています。

LUA_INCORRECT_SEQUENCE_NUMBER

通常流れ要 a で u 信されたシーケンス V 号が、最後のシーケンス V 号よりも大きくなっていませんでした。シーケンス V 号エラー、あるいはこの u 信側の現行セッション制御状態またはデータ・フロー制御状態では v 可されない RH または RU が検出されました。このエラーにより、意図されたハーフセッション構成要素に要 a が送達できませんでした。

付録C. APPC 会話状態の変化

以下のテーブルは、各 APPC verb を / 行できる会話の状態と、その verb の完了時に / こる状態変化を (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 (してあります。

戻りコードが (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)。

/ こりうる会話状態は、表の一V 上の欄に (されています。各 verb について、その verb を各状態で / 行したときの k 果が、各欄の見出しの下に！のように表 (されます。

- **X** は、この状態では該当の verb を / 行できないことを (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 (します。**R**eset（リセット）、**S**end（送信）、 **S**end **P**ending（送信保留）、**R**eceive（u 信）、**C**onfirm（確認）、**C**onfirm **S**end（送信確認）、 **C**onfirm **D**eallocate（割り振り解除確認）、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 (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 / 行した会話にはF 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 / 生しないことを (します。

全二重会話の状態変移については、 392ページの表 23を参照してください。

表 22. APPC > 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)**	**受信 (R)**	**確認 (C)**	**送信確認 (CS)**	**割り振り解除確認 (CD)**	**通知保留 (PS)**
[MC_]ALLOCATE AP_OK (その他)	 S T	/	/	/	/	/	/	/
CANCEL_CONVERSATION	X	T	T	T	T	T	T	T
[MC_]CONFIRM AP_OK AP_ERROR	X	 S R	 S R	X	X	X	X	X
[MC_]CONFIRMED	X	X	X	X	R	S	T	X
[MC_]DEALLOCATE（異常終了） [MC_]DEALLOCATE（その他） AP_ERROR (その他)	X X	T R T	T R T	T X	T X	T X	T X	T X
[MC_]FLUSH	X	S	S	X	X	X	X	X

表 22. APPC > 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)	受信 (R)	確認 (C)	送信確認 (CS)	割り振り解除確認 (CD)	通知保留 (PS)
[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 AP_OK (その他)	R T	/	/	/	/	/	/	/
[MC_]RECEIVE_AND_POST (注 4)	X	P	P	P	X	X	X	X
[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 AP_OK AP_ERROR	X	S R	S	X	X	X	X	X
[MC_]SEND_ERROR AP_OK AP_ERROR	X	S R	S	S	S	S	S	S
[MC_]TEST_RTS	X	S	S	R	C	C	C	P

注:

1. 表の「戻りコード」の欄で、AP_ERROR という省略語は以下の戻りコードに使用されます。

   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. ！のいずれかの戻りコードを u 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. ！のような異常戻りコードを u 信した場合は、状態変化は生じません。会話は、常に、verb を / 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 / 行され、初期 **primary_rc** として AP_OK を u 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 (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 (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

   割り振り解除確認状態
   AP_CONFIRM_DEALLOCATE、AP_DATA_CONFIRM_DEALLOCATE、
   AP_DATA_COMPLETE_CONFIRM_DEALL

> 二重会話の状態変移については、 389ページの表 22 を参照してください。

表 23. 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 AP_OK (その他)	 SR T	 /	 /	 /
CANCEL_CONVERSATION	X	T	T	T
[MC_]DEALLOCATE (異常終了) [MC_]DEALLOCATE (フラッシュ)	X X	T R	T T	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 (その他)	 SR T	 /	 /	 /
[MC_]RECEIVE_AND WAIT AP_OK AP_ERROR AP_DEALLOC_NORMAL	 X X X	 SR SR S	 X X X	 R R T
RECEIVE_EXPEDITED_DATA	X	SR	S	R
[MC_]RECEIVE_ IMMEDIATE AP_OK AP_ERROR AP_DEALLOC_NORMAL	 X X X	 SR SR S	 X X X	 R R T
[MC_]SEND_DATA AP_OK AP_ERROR_INDICATION	 X X	 SR SR	 S T	 X X
[MC_]SEND_ERROR AP_OK AP_ERROR_INDICATION	 X X	 SR SR	 S T	 X X

注:

1. 表の「戻りコード」の欄で、AP_ERROR という省略語は以下の戻りコードに使用されます。

   AP_PROG_ERROR_TRUNC

   AP_PROG_ERROR_NO_TRUNC

   AP_SVC_ERROR_TRUNC

   AP_SVC_ERROR_NO_TRUNC

2. ！のいずれかの戻りコードを u 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. ！のような異常戻りコードを u 信した場合は、状態変化は生じません。会話は、常に、verb を / 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

付録D. Communications Server サービス検索プロトコル

ディスカバリーおよびロード・バランシング API

Communications Server (IBM Communications Server for Windows NT と IntranetWare for SAA の両方を指します) のアプリケーション・プログラム開 / T は、 TCP/IP プロトコルを使用してサービスを検索し、それらのサービス間のロード・バランシングを行えるようになりました。アプリケーション・プログラムでこの新 , ファンクションを利用するための基本的な方 0 として、 ! の 3 つがあります。

- Communications Server SNA API (LUx (RUI/SLI)、APPC、CPIC)。既存のアプリケーションがすでに SNA API に合わせて書かれている場合には、これらの API を使用すると、基本的に ″無料で″ サポートを得ることができます。この方 0 を使用する場合、新しくコードを書かなくても検索/ロード・バランシング・ファンクションを利用することができます。この方 0 を使用する場合の唯一の制約は、 API コードが、クライアントの構成データが Novell Directory Services (NDS) for IntranetWare または INI ファイル、あるいは LDAP Communications Server for Windows NT に入っていることを予想していることです。
- サービス検索プロトコル (SLP) ユーザー・エージェント (UA) API。製品のパッケージに含まれている SLP UA DLL は、 TCP/IP 接続を介して Communications Server サービス検索およびロード・バランシングをサポートします。これは、サービス検索/ロード・バランシングを B 施する方法、クライアント構成を獲得する場所、およびこれらのファンクションをエンド・ユーザーに提供する方法の点で、アプリケーション開 / T にとって最も柔軟な方 0 です。
- UA (検索用) と QEL/MU CM_CSLIST_GETII プリミティブ (ロード・バランシング用) の組み合わせ (3270 および LU6.2 アプリケーションの場合のみ)。この方 0 は、最初の 2 つの方 0 を混成したもので、作成する必要のあるコードの量を検索ファンクションだけに減らしつつ、クライアント構成については最大の柔軟性を得ることができます。

IBM および Novell は、検索とロード・バランシングに API クライアントを使用することをお勧めします。アプリケーション開 / T がそのようにできない場合、または Telnet のサポートを希望する場合のために、 2 V 目の方 0 が用意されています。すでに QEL/MU のサポートが提供されている場合には、3 V 目の方 0 を使用することができます。最初の方 0 はアプリケーション開 / T の観点からは特に目新しいことがないため、以下の説明は後の 2 つの方 0 に適用されます。

構造

UA API は、 ″サービス検索用 API″ インターネット草案 (97 年 3 n 25 日付) で提案されたモデルに基づく F 用の C 言語 APIです。サービス登録には以下の特性が適用されます。

- すべての登録は米国 Q 語で行われます。
- 文字セットは US-ASCII です。

この API は、 Windows 95 および Windows NT プラットフォームでは IBMSLP.DLL としてパッケージ化されています。この SDK には、関 8 する構造、定数、およびファンクション・プロトタイプを定 A するヘッダー・ファイルが用意されています。DLL は API クライアントのインストール時にインストールされ、製品 CD-ROM の ¥NWSAA¥SAASDK¥SLP¥BINARY¥IBMSLP.DLL または ¥CSNT¥SDK¥SLP¥BINARY¥IBMSLP.DLL にも、他の SLP SDK ファイルとともに } められています。

シナリオ

各シナリオでは、ユーザー・エージェント API を使用するアプリケーション・プログラムは ″app″ と呼ばれています。エンド・ユーザー (app の使用 T) は、短縮して ″ユーザー″ と呼ばれています。

方 0 2: UA API で最小ロード (または ″低ロード″) のサービスを検索する。

1. アプリケーションが SL_Open を / 行して SLP とのセッションをオープンします。
2. 有効 O 囲が構成されていない場合、または他の方法では app によって使用できない場合、アプリケーションは、希望するサービス・タイプに関して、属性タグ・フィルター 'SCOPE' を指定して SL_GetAttrs API を呼び出し、有効で到達可能な有効 O 囲を獲得します。この API 呼び出しで、管理される IntranetWare for SAA 3.0 または CSNT 6.0 サービスのうちの 1 つのサービス名を提供すると、提供したサービス・タイプに該当する有効 O 囲だけが戻されるようになります。
3. アプリケーションは ! に、希望するサービス、獲得された有効 O 囲名のうちの 1 つ、および必要なサービス属性を (す照会文字列を指定して、SL_GetService を / 行します。この例の照会では、説明のためにサービス属性として LUPOOL および LOAD を指定します。サービス ~ 答には、一致するサービスが見つからなかったことを (す標 1 、またはサービスを提供することのできる URL のリストが含まれるとともに、照会文字列の要 o も満たされます。
4. アプリケーションが戻されたリストを分析します。
5. URL が戻されない場合、アプリケーションは、ステップ 3 で述べたオリジナルの SL_GetService 要 a を変更し、新しい LOAD 基準を指定して再 / 行するか、あるいはサービスが現在使用不能であることをエンド・ユーザーに通知します。
6. 単一の URL が戻された場合には、分析が行われます。
7. URL のリストが戻された場合には、 ! のようになります。
   - **オプション 1 - ″最小ロード″ 検索**
     a. アプリケーションは、サービス ~ 答で戻された各 URL について SL_GetAttrs を / 行します。それぞれの呼び出しで、選択文節に LOAD 属性が指定されます。この LOAD 値は属性 h 得 ~ 答に入れて戻されます。
     b. アプリケーションは LOAD 値が最も低い URL を選択します。
     c. アプリケーションは選択した URL で表されているサーバーに接続し、その SNA セッションを開始します。
     d. アプリケーションは SL_Close を / 行して SLP セッションをクローズします。
   - **オプション 2 - ″低ロード″ 選択**

a. 戻されたリストからランダムに URL を選択します。
b. アプリケーションは選択した URL で表されているサーバーに接続し、その SNA セッションを開始します。
c. アプリケーションは SL_Close を / 行して SLP セッションをクローズします。

多数のサーバー間でロード・バランシングを行うために、 2 つのオプションが提供されていることに注意してください。 2 つのオプション間の g 要な相違は、！のとおりです。オプション 1 では、最小ロードのサーバーが選択されることが保証されますが、オプション 2 よりも多くの LAN トラフィックが生成されます。オプション 2 では、″低ロード″ サーバーが選択されることだけが保証されますが、オプション 1 に f べて選択プロセスでの潜在的な LAN 回線トラフィックが少なくなります。

再試行: 多くの場合、ユーザーがリソースを最大限に使用できるようにするために、アプリケーションによる接続再試行が必要です。アプリケーションによる接続再試行が必要になる場合の 1 つとして、アプリケーションが SL_GetService で戻された URL への接続を試み、その後で SNA セッションを確立したにもかかわらず、使用可能な LU がない場合があります。この条 o は、 SLP を介して登録されたサービスと登録サーバーで現在選択可能なサービスとの間の k 合が：われているために / こることがあります。アプリケーションは、選択したサービスへの接続に：敗した場合、戻された別のサービス (たとえば、！にロードが少ないサーバー) への接続を再試行する必要があります。選択可能なサービスがほかにない場合、アプリケーションは、最初の SL_GetService からやり直すか、あるいはその条 o をエンド・ユーザーに報告することができます。

URL 形 0: Communications Servers によって公 (するされる URL は、小数点付き 10 進数の IP アドレスとポート V 号の 2 つの部分からなります。

URL は、！の A 0 の ASCII 文字列です。

```
<IP address>:<port number>
```

この IP アドレスは、サーバーのデフォルト IP アドレスです。ポート V 号は、公 (されるサービスのタイプによって異なります。

表 24. サービス・タイプ/ポート情報

サービス・タイプ	ポート
commserver	ウェルノウン CommExec listen ポート 1366
cs3270	ウェルノウン CommExec listen ポート 1366
csappc	ウェルノウン CommExec listen ポート 1366
tn3270	サーバーの ETC/SERVICES ファイルから h 得した Telnet ポート、または Telnet サーバーに構成された Telnet ポート

ポート

Communications Server for NWSAA の！期リリースでは、 AS400 システムへの複数の接続を行うために複数のポートを用意できるようになる予定です。 CSNT 6.0 は現在複数のポートをサポートしています。暗号化されたセキュア Telnet セッションもサポートされるようになる予定です。そのためには、セキュア・セッション用にデフォルト・ポートV号とは異なるポートV号が必要になります。エミュレーターは、SLP サービス・ディスカバリーから戻されたポートV号を使用できる必要があります。サービス・タイプに関する詳細については、TEMPLATE.HTM ファイルに - 載されています。

例 1: アプリケーションが Telnet を介して 3270 エミュレーションを行います。構成された ACCOUNTS の LU プールで選択可能ないずれかの LU に接続する必要があり、また最もロードがZいサーバーを介して接続する必要があります。ネットワークでは有効O囲は構成されません。メインフレーム・ホストがダイナミック装置タイプをサポートするため、アプリケーションで装置タイプを指定する必要はありません。

アプリケーションは最初に SL_GetService 要 a に関する以下の述部を / 行して、サーバーを検索します (すべての例で、'¥t' はタブ文字です)。

```
tn3270//LUPOOL==ACCOUNTS*/
```

この時点で、(！に (すような) 3 つの URL のリストが戻されます (ポートV号 23 は、 Telnet 接続要 a の場合の標準ポートです)。

```
service:tn3270://9.37.51.254:23
service:tn3270://9.37.51.260:23
service:tn3270://9.37.51.256:23
```

このアプリケーションは最小ロード検索をB施するように設Wされているため、各 URL に宛てて一連の SL_GetAttrs 呼び出しを / 行して各サーバーのロード測定を行います。ロード情報だけをu けh るために、下 - のような選択文節を指定します。

```
URL = service:tn3270://9.37.51.254:23
Attribute filter = LOAD
```

- 属性 LOAD は値 ″5″ とともに戻されます。
- アプリケーションが 2 つ目の URL に関する 2 V目の SL_GetAttrs を / 行し、ロード ″2″ が戻されます。
- 最後に 3 V目のサーバーのロードが測定され、ロード ″10″ が戻されます。

2 V目のサーバーのロードが最も低いため、アプリケーションは接続ターゲットとして 9.37.52.260:23 を選択します。アプリケーションは 9.37.51.260 を介して接続を試みますが、選択可能な LU がないため、接続は : 敗します。そこで、(その！にロードが低いサーバーである) 9.37.51.254 を介して接続を試み、今度は成功します。

例 2: 別のアプリケーションで tn3270 エミュレーションが提供されています。このアプリケーションは、このサービスを提供する、ロードのZいサーバーを見付ける必要があります。クライアントの構成は INI ファイル または NDS から入 j され、その有効O囲は ENGINEERING になっています。また、LU プール SMITH_1 から LU タイプ 2 モデル 2 を検索する必要があります。

このアプリケーションは最初に、サービス・タイプに TN3270: を指定し、属性タグ・フィルターに 'SCOPE' を指定して、 SL_GetAttrs 呼び出しを / 行します。これにより、TN3270 をサポートするサーバーについて管理されていた有効O囲値のリストが戻されます。説明のために、SL_GetAttrs 呼び出しで有効O囲値 'ENGINEERING' が戻されたものと仮定します。アプリケーションは！に、SL_GetService 要 a に関する以下の述部を組み立てて、この有効O囲内のサーバーのうちで、最初の LU 装置タイプ、およびロード要 o を満たすものを検索します (すべての例で、'¥t' はタブ文字です)。

```
tn3270/ENGINEERING/LUPOOL==SMITH_1&yen;t3270002,LOAD <= 10/
```

このアプリケーションは、ロード増分 10 で検索を行うように設Wされているため、最初の SL_GetService 要 a で空のリストが戻された場合には、そのサービスを再び指定し、さらに新しいロード属性を指定して、SL_GetService を最 / 行します。

```
tn3270/ENGINEERING/LUPOOL==SMITH_1&yen;t3270002,LOAD <= 20/
```

この時点で、(！に (すような) 2 つの URL のリストが戻されます (ポートV号 23 は、 Telnet 接続要 a の場合の標準ポートです)。

```
service:tn3270://9.37.51.254:23
service:tn3270://9.37.51.260:23
```

アプリケーションは、ロードが 20% 以下であれば、絶対最小ロード・サーバーであるかどうかを考慮せずにそのサーバーを選択します。したがって、戻された 2 つの URL のうちの 1 つをランダムに選択します。

```
URL = service:tn3270://9.37.51.260:23
```

アプリケーションは接続ターゲットとして 9.37.52.260:23 を選択し、接続は成功します。

方式 3: サービス検索に UA を使用し、ロード・バランシングに CM_CSLIST_GETII を使用する。: CM_CSLIST_GETII プリミティブが QEL/MU エミュレーター用に提供されています。このプリミティブは拡張され、アプリケーションで複数のフィルターを提供できるようになっています。この方 0 で使用する構造および定A は、この SDK 内のヘッダー・ファイル cmi.h に入っています。この方 0 を使用するためには、以下の j 順が適用されます。

1. アプリケーションが SL_Open を / 行して SLP とのセッションをオープンします。
2. 有効O囲が構成されていない場合、または他の方法では app によって使用できない場合、アプリケーションは、'cs3270' サービス・タイプに関して、属性タグ・フィルター 'SCOPE' を指定して SL_GetAttrs API を呼び出し、有効で到達可能な有効O囲を獲得します。この API は、IP バージョンの CM_CSLIST_GETII プリミティブに~答することのできる、 Communications Server のサービス URL に対~する有効O囲のリストを戻します。
3. アプリケーションは、 'cs3270' サービスと有効な有効O囲だけを指定して SL_GetService を / 行します。サービス~答には、アプリケーションが接続できるサーバーのうちで CM_CSLIST_GETII プリミティブを処理できるサーバーの、URL のリストが含まれます。
4. アプリケーションはリスト内の選択した URL で表されているサーバーに接続します。

5. アプリケーションは SL_Close を / 行して SLP セッションをクローズします。
6. アプリケーションは、ロード・バランスを考慮したサーバーのリストを検索するために、 CM_CSLIST_GETII プリミティブを組み立てます。このプリミティブ内の AgentType フィールドは希望するサービスに設定され、フィルター仕様には有効O囲と LU プール名 (該当する場合) が含まれます。
7. サーバーの TCP/IP アドレスをロード・バランスの順 (最低ロードから最高ロードの順) に並べたリストが入った、 CM_CSLIST_GETII_ACK が戻されます。
8. アプリケーションは、リスト内の最初のサーバーを選択し、それに接続します。
9. アプリケーションは、そのサーバーとの SNA セッションの確立を試みます。セッションが確立できなかった場合には、確立が成功するかあるいは戻されたリストが終わるまで、リスト内の！のサーバーについて前のステップを + り返します。

表 25. CM_CSLIST_GETII プリミティブ

フィールド名	フィールド・オフセット (16 進数)	フィールド長 (10 進数)	タイプ	内容および用途
PrimType	x00	4	long int	cmi.h におけるような CM_CSLIST_GETII
UserParm	x04	4	long int	ユーザーが ~ 答で戻したい値
Reserved	x08	4	long int	ゼロ
ServiceType	x0c	4	long int	0x12B (ロード・バランシング・サポート用)
ProdVersion	x10	4	long int	-1 (考慮しないことを表す)
NWVersion	x14	4	long int	-1 (考慮しないことを表す)
Flags	x18	4	long int	「Flags の値」テーブルを参照
AgentType	x0c	4	long int	「AgentType の値」テーブルを参照
FilterList	x1c	*	FilterList_t	「FilterList_t」テーブルを参照 (値は ″Flags″ の設定によって異なる)

表 26. CM_CSLIST_GETII プリミティブ

定数	値	意味
ゼロ	0	V号なしリストが必要。フィルターが指定されていない。(従来との互換性のために提供)

表 26. CM_CSLIST_GETII プリミティブ (続き)

定数	値	意味
CMCsListFlags_LBPool	1	ロード・バランスされたプール名を指定した、ロード・バランスされたリストが必要 (従来との互換性のために提供された値)
CMCsListFlags_LBAgent	2	ロード・バランスされたリストが必要。ロード・バランスのために AgentType を使用。
CMCsListFlags_LBFilter	3	ロード・バランスされたリストが必要。フィルターの可変長リストが後に続く。

表 27. Flags の値 (cmi.h より)

定数	値	意味
CSA_3270	0x126	LU タイプ 1/2/3 用の SNA ゲートウェイ・エージェントが必要
CSA_SAA	0x12B	LU タイプ 6.2 用の SNA ゲートウェイ・エージェントが必要

表 28. AgentType の値 (csobjtyp.h より)

フィールド名	フィールド・オフセット (16 進数)	フィールド長 (10 進数)	タイプ	内容および用途
FilterNameLen	x00	4	long int	以下のロード・バランシング・グループ (プール) 名の長さ
FilterName	x04	*	ASCII	ロード・バランシング・グループ (プール) 名

表 29. FilterList_t (Flags = CMCsListFlag_LBPool の場合)

フィールド名	フィールド・オフセット (16 進数)	フィールド長 (10 進数)	タイプ	内容および用途
FilterCount	x00	4	long int	この後のフィルター・リスト名構造の数 (Flags = zero の場合には 0)
FilterList	x04	*	Filter_t	フィルター・リスト名構造のリスト。各構造は可変長。

表 30. FilterList_t (Flags = zero | Flags = CMCsListFlag_LBFilters の場合)

フィールド名	フィールド・オフセット (16 進数)	フィールド長 (10 進数)	タイプ	内容および用途
FilterLength	x00	4	long int	構造の長さ (この長さフィールドも含む)
FilterType	x04	4	long int	「FilterType の値」テーブルを参照
FilterName	x08	*	ASCII	フィルター名の値

表 31. Filter_t

定数	意味
CMCsListFilter_LBPool	ロード・バランシング・プール名。リスト当たり 1 つのプールだけを指定できる。このフィルターは、AgentType CSA_3270 の場合にだけ有効。
CMCsListFilter_Scope	SLP 有効O囲名。 1 つの有効O囲だけを指定できる。有効O囲が指定されていない場合には、有効O囲のないすべてのサービスが想定される。

表 32. FilterType の値 (cmi.h より)

フィールド名	フィールド・オフセット (16 進数)	フィールド長 (10 進数)	タイプ	内容および用途
PrimType	x00	4	long int	cmi.h におけるような CM_CSLIST_GETII_ACK
UserParm	x04	4	long int	CM_CSLIST_GETII で渡される
Reserved	x08	4	long int	ゼロ
ServiceType	x0c	4	long int	CM_CSLIST_GETII で渡される
Flags	x10	4	long int	CM_CSLIST_GETII で渡される
ServiceCount	x14	4	long int	後に続くサーバー項目の数

表 33. CM_CSLIST_GETII_ACK プリミティブ

フィールド名	フィールド・オフセット (16 進数)	フィールド長 (10 進数)	タイプ	内容および用途
ProdVersion	x00	4	long int	製品のバージョン
Platform	x04	4	long int	CMCsListPlatform_IWSAA

表 33. CM_CSLIST_GETII_ACK プリミティブ (続き)

フィールド名	フィールド・オフセット (16 進数)	フィールド長 (10 進数)	タイプ	内容および用途
CSNameLen	x08	4	long int	！のサーバー名の長さ
CSName	*	*	long int	サーバーの名前 (NULL 終了)
CSAddrLen	*	4	long int	！の IP アドレスの長さ
CSAddress	*	*	ASCII	！のA 0 によるサーバーの IP アドレス: 小数点付き 10 進数の IP アドレス:ポート
NameLen	*	4	long int	！のエージェント名の長さ
AgentName	*	*	*	サーバーのエージェントの名前 (NULL 終了)

表 34. CM_CSLIST_GETII_ACK プリミティブのサーバー情報構造

フィールド名	フィールド・オフセット (16 進数)	フィールド長 (10 進数)	タイプ	内容および用途
PrimType	x00	4	long int	cmi.h におけるような CM_CSLIST_GETII_ERR
UserParm	x04	4	long int	CM_CSLIST_GETII で渡される
Reserved	x08	4	long int	ゼロ
Errno	x0c	4	long int	エラーV号

構成の考慮事項

有効範囲: クライアントからのサービス要 a の有効O囲値の獲得方法としては、！の 2 つがあります。

ディスカバリー

有効O囲値は、 SL_GetAttrs API を使用して (属性フィルターが ″SCOPE″ のサービス・タイプに関して有効O囲のない属性要 a を / 行して) 検出することができます。この API は、ネットワークで現在アクティブなサービスの有効O囲のリストを戻します。このリストは、ユーザー選択のために表 (することができます。

構成

有効O囲値は、クライアントの構成によって獲得することができます。 Novell クライアント・リクエスターを介して NDS にアクセスできるアプリケーションの場合、有効O囲は NDS に構成してそこから獲得することができます。 IntranetWare for SAA 3.0 は、NDS スナップインを提供し、 NDS スキーマを拡張して SLP 有効O囲を含む多くのタイプのクライアント構成値に対~できるようにします。有効O囲とその意味の詳細については、下 - を参照してください。

DA-ディスカバリー・タイムアウト

SLP_Open API のパラメーターである DA-ディスカバリー・タイムアウト値は、ネットワーク内のディレクトリー・エージェント (DA) を検出するために SLP API が待つ必要のある時間の長さを制御するために使用されます。ディスカバリー要 a はマルチキャスト要 a であり、すべての DA ~答を集めるために必要な時間の長さは、多くの 8 数に~じて異なります。ネットワークに DA がない場合、このタイムアウト値をゼロに設定すると、 DA ディスカバリーが行われないように指定することができます。タイムアウトはミリ秒単位で表されます。

SA マルチキャスト・タイムアウト

SL_Open API のパラメーターである SA マルチキャスト・タイムアウト値は、要 a の有効O囲をサポートする DA が 1 つもない場合に、ネットワーク内でサービス、属性、またはサービス・タイプを検出するために SLI API が待つ必要のある時間の長さを制御するために使用されます。この状態では、これらの要 a はマルチキャスト要 a であり、 SLP API は、戻される複数の~答を集めるためにタイムアウト値まで待ちます。タイムアウトはミリ秒単位で表されます。

管理T用のヘルプ情報

有効範囲

有効O囲は、クライアントによるネットワーク内のサーバーへのアクセスを制御および管理するために使用されるパラメーターです。これは、サービス検索プロトコル有効O囲と同じです。有効O囲が提供する制御は、！の 2 つの理由から必要になります。

- ネットワーク、クライアントの数、およびサーバーの数が多くなるにつれて、多くのクライアントによるそれらのサーバーへのアクセスを分割して、ネットワークへの全体的なトラフィックを減少させることが必要になります。
- 管理Tが管理グループ内のユーザーとサーバーを編成できるようになります。

有効O囲の値の意味は、ネットワークの管理Tによって定Aされます。これらの値は、任意のエンティティーを表すことができます。一L的には、部門、地域、または組織のいずれかを表します。

有効範囲の使用方法

IWSAA 3.0 または CSNT 6.0 の各サーバーは、それぞれの構成ツールを介して 1 つまたは複数の有効O囲に割り当てられます。それらのサーバーを使用するクライアントは、単一の特定有効O囲内のサーバーまたは有効O囲のないサーバーに接続するように構成しなければなりません。構成可能サービス 3270 および APPC には、別の有効O囲を割り当てることができます。

有効範囲と SLP の関連

IntranetWare for SAA 3.0 または CSNT 6.0 の有効O囲は、 SLP 有効O囲に直接関連付けられます。したがって、SLP サービス・エージェントおよびディレクトリー・エージェントは、それらの構成済み有効O囲をサポートするネットワーク内に置く必要があります。クライアントが有効O囲に基づいてサービスを検索できるようにしたい場合には、有効O囲がネットワーク全体とどのように関連しているのか、常に考慮してください。有効O囲が使用されているネットワーク内に有効O囲のないサービスがある場合、それらのサービスは、どのような有効O囲の要 a も満たす適格があり、そのことが有効O囲のないサービスをサポートするサービス・エージェントおよびディレクトリー・エージェントにとって負担になる可能性があります。したがって、到達可能なすべてのサーバーで有効O囲を構成するか、あるいはどのサーバーにも有効O囲を構成しないようにすることをお勧めします。サイト・ネットワークで (上方向スケール拡張のために) ディレクトリー・エージェントを使用する場合には、サーバー用に構成された有効O囲を扱えるように、それらのエージェントを構成する必要があります。また、ディレクトリー・エージェントを含むネットワークで有効O囲のないサービスが使用されている場合には、有効O囲のないディレクトリー・エージェントを少なくとも 1 つは設定する必要があります。

注: SNA API クライアントが有効O囲のないサーバーに接続するように構成されている場合、有効O囲のないサーバーだけが ~ 答します。

ロード・バランシングの重み係数

ロード・バランシングの重み 8 数を使用すると、管理 T は、通信サーバーごとにロード・バランシング測定の変更または操作を行えるようになります。 8 数は、サーバーごとに異なる可能性があります。ロード測定値は 0 から 100 までの整数であり、サーバーにおけるロードのおよそのパーセンテージを表します (最大値は 100)。重み 8 数は、このW算の要素を管理 T に提供します。この 8 数は、 ! のように役立ちます。

- Communications Server アルゴリズムでは考慮されていない 8 数がほかにあって、それらがサーバー・ロードに F 響を与えることがあります。たとえば、通信サーバーが SNA ゲートウェイ・トラフィック専用になっていない場合がこれに該当します。
- TN3270/TN5250 サービスを提供する IntranetWare for SAA 3.0 ゲートウェイが、ロード・バランシングに SLP を使用する他の TN サーバー・インプリメンテーション (CSNT 6.0) とネットワーク上で共存している場合、 IntranetWare for SAA 3.0 8 数を調整して誤差を補正することができます。

重み 8 数を使用すると、管理 T は、サーバーのロード測定を偏向させて、そのサーバーが選択されやすいようにしたり、選択されにくいようにしたりすることができます。

サービス・テンプレート

通信サーバーのサービス・テンプレート

commserver サービス・タイプは、Communications Server for Windows NT または IntranetWare for SAA 3.0 のいずれかです。

IntranetWare for SAA 3.0 の場合、commserver サービス・タイプは CommExec のロード時に登録されます。これは、IntranetWare for SAA サーバーの総称属性を - 述するものです。これらの属性は、IntranetWare for SAA で提供されるその他のサービス・タイプでも + り返されます。

```
Release = <version/release>
```

これは、commserver 公 (サービスのバージョンおよびリリースのレベルです。A 0 は、vv.rr.mm で、″vv″ は大バージョン V 号、″rr″ は小バージョン V 号、 ″mm″ は修正レベルです。すべての V 号は、2 文字になるように左側にゼロが埋め込まれます。例: バージョン 6、リリース 0、修正レベル 0 は ″06.00.00″ で表されます。

```
Platform = <platform>
```

これは、公 (サービスの基礎になるネットワーク・オペレーティング・システム・プラットフォームです。定 A されている値は ! のとおりです。

IW サーバーは IntranetWare ネットワーク・オペレーティング・システムを使用します。

NT サーバーは Microsoft の Windows NT オペレーティング・システムを使用します。

OS2 サーバーは OS2 オペレーティング・システムを使用します。

AIX サーバーは AIX オペレーティング・システムを使用します。

```
Protocol = <protocol>
```

これは、このサービスを提供するサーバーによってサポートされる 1 つまたは複数のプロトコルです。定 A されている値は ! のとおりです。

IP サーバーは IP (TCP/IP または UDP/IP) を介してクライアント接続をサポートします。

IPX サーバーは IPX (SPX/IPX) を介してクライアント接続をサポートします。

```
Server name = <server name>
```

これは、インストール中に構成されたサーバーの名前です。この値は、IW プラットフォームの場合にだけ意味を持ちます。

通信サーバーのサービス登録メッセージ

```
URL:service:commserver://<addr-spec>:<port-number>
```

属性:

```
[(SCOPE=<string>),]
(RELEASE=06.00.00),
(PLATFORM=NT),
(PROTOCOL=IP),
(SERVERNAME=<string>)
```

従属 LU のサービス・テンプレート

通信サーバーの従属 LU サービスは、サーバー特定 API およびプロトコルを介して、SNA ネットワークへの 3270 ゲートウェイ・アクセスを行います。属性は、サーバーで使用可能な 3270 装置のタイプ、LU プール、およびロード情報を ? G しています。

```
Load = <server_load>:
```

これは、サービスを利用するために接続する最小ロードの通信サーバーを = 別するための、ロード・バランシング数量です。有効な値の O 囲は、0 から 100 までの整数です。0 は可能な最小ロードを表し、 100 は最大ロードを表します。

```
LU Pool = <pool_name>,
<pool_name>/t<dev-type>,
<pool_name>/t<dev_type>, ...
<pool_name>/t<dev-type>
```

このサービスで使用可能な LU プールの LU プール名と、各プールでサポートされる関連装置タイプを 1 別します。それぞれの値はレコードで、その最初のトークンはプールのプール名、 2 V 目のトークンはそのプールでサポートされる装置タイプです。装置タイプが指定されていないプール名は、そのプールに未知のタイプの LU が含まれていることを (します。特定のプール名に関連したレコードは、サポートされる装置タイプごとに + り返されます。少なくとも 1 つの LU をプールに提供する PU プロファイルがサーバーでアクティブになっている場合、登録要 a に特定のプールが組み込まれます。有効な dev_type の O 囲は ! のとおりです。

表 35. CM_CSLIST_GETII_ERR プリミティブ

dev_type	意味
3270002	LU タイプ 2 モデル 2
3270003	LU タイプ 2 モデル 3
3270004	LU タイプ 2 モデル 4
3270005	LU タイプ 2 モデル 5
3270DSC	プリンター LU

特定の装置タイプとして構成された LU がサーバーのアクティブ PU プロファイルに含まれている場合には、その装置タイプが登録要 a に組み込まれます。

従属 LU のサービス登録メッセージ

```
URL: service:cs3270://<addr-spec>:<port-number>
```

属性:

```
[(SCOPE=<string>),]
(RELEASE=06.00.00),
(PLATFORM=NT),
(PROTOCOL=IP),
(SERVERNAME=<string>),
(LOAD=<integer 0 to 100>),
[(LUPOOL=pool-name0/tANY,
pool-name1/tdevice_type1,
pool-name2/tdevice-type2, ...
pool-namen/tdevice-typen)]
```

TN3270 サービス・テンプレート

tn3270 サービスは、 TN3270 プロトコルを介して SNA ネットワークへの 3270 ゲートウェイ・アクセスを行います。属性は、サーバーで使用可能な 3270 装置のタイプ、LU プール、およびロード情報を ? G しています。 LU のプール属性とロード属性は、サービス・タイプ cs3270 の場合と同じです。

BIND, DATA, RESPONSES, SCS, SYSREQ

これらのキーワード属性は、このサービスでサポートされる TN3270e ファンクションを - 述するものです。

BIND サーバーは SNA バインド・イメージ・ファンクションをサポートします。

DATA s SNA 3270 データ・ストリームがサーバーによってサポートされます。

RESPONSES
サーバーは SNA ~答モードをサポートします。

SCS サーバーは SNA 3270 SCS データ・ストリームをサポートします。

SYSREQ
SYSREQ キーボード・キーがサーバーでサポートされます。

これらの属性によって - 述されるファンクションが選択可能な場合には、これらの属性はサービス公 (で表 (されます。

```
Security = <security>
```

このフィールドには、サーバーによってサポートされるセキュリティー ; 法が - 入されます。定 A されている値は ! のとおりです。

NONE このサーバーには明 (的なセキュリティー ; 法がありません。
SSLV3 このサーバーはセキュア・ソケット・レイヤー バージョン 3 標準をサポートします。
Ciphersuites = <CipherSpec>,
<CipherSpec>, ...
<CipherSpec>
このサーバーでサポートされる暗号仕様を 1 別します。定 A されている値は ! のと

おりです。
NULL_NULL
NULL_MD5
NULL_SHA
RC4_MD5_EXPORT
RC4_MD5_US
RC4_SHA_US
RC2_MD5_EXPORT
DES_SHA_EXPORT
TRIPLE_DES_SHA_US

```
RFC1576, RFC1646, RFC1647
```

サービスによってサポートされる！能が - 載されている RFC V号。 TN3270 に関する現行の RFC には、1576、1646、および 1647 があります。

TN3270 のサービス登録メッセージ

```
URL: service:tn3270://<addr-spec>:<port-number>
```

属性:

```
[(SCOPE=<string>),]
(RELEASE=06.00.00),
(PLATFORM=NT),
(PROTOCOL=IP),
(SERVERNAME=<string>),
(LOAD=<integer 0 to 100>),
[(LUPOOL=pool-name(0)/tANY,
pool-name1/tdevice_type1,
pool-name2/tdevice-type2, ...
pool-namen/tdevice-typen)]
BIND,
DATA,
RESPONSES,
SCS,
SYSREQ,
(SECURITY=NONE),
(SECURITY=<security>),
(CIPHERSUITES=<Spec1,Spec2,...Specn>),
RFC1576,
RFC1646,
RFC1647
```

TN5250 サービス・テンプレート

tn5250 サービスは、 TN5250 プロトコルを介して SNA ネットワークへの 5250 ゲートウェイ・アクセスを行います。属性は、アクセス可能な AS400、サーバーで使用可能なサービスおよびロード情報を？ G しています。

```
Release = <release>
```

これは、公 (を行う commserver のバージョンおよびリリースです。

```
Protocol = <protocol>
```

これは、このサービスを提供するサーバーによってサポートされる 1 つまたは複数のプロトコルです。定Aされている値は！のとおりです。

```
IP - サーバーは IP (TCP/IP または UDP/IP) を介して接続をサポートします。
Platform = <platform>
```

これは、公 (されるサービスの基礎になるネットワーク・オペレーティング・システム・プラットフォームです。定Aされている値は！のとおりです。

```
IW - サーバーは IntranetWare ネットワーク・オペレーティング・システムを使用します。
NT - サーバーは Microsoft の Windows NT オペレーティング・システムを使用します。
Server Name = <server name>
```

これは、インストール中に構成されたサーバーの名前です。

```
AS400 Name = <host name>
```

これは、サービス保 i が適用される AS400 ホストの名前です。

```
Load = <INTEGER>
```

これは、最小ロードの通信サーバーを = 別するための、ロード・バランシング数量です。有効な値のO囲は 0 から 100 までの整数です。

```
Security = <security>
```

このフィールドには、サーバーによってサポートされるセキュリティー ; 法が - 入されます。 B際の値は！のとおりです。

NONE　このサーバーには明 (的なセキュリティー ; 法がありません。
SSLV3　このサーバーはセキュア・ソケット・レイヤー バージョン 3 標準をサポートします。
Ciphersuites = <CipherSpec>,
<CipherSpec>, ...
<CipherSpec>
このサーバーでサポートされる暗号仕様を 1 別します。定Aされている値は！のとおりです。
NULL_NULL
NULL_MD5
NULL_SHA
RC4_MD5_EXPORT
RC4_MD5_US
RC4_SHA_US
RC2_MD5_EXPORT
DES_SHA_EXPORT
TRIPLE_DES_SHA_US

```
Function = <function>
```

このフィールドには、サーバーによってサポートされる TN5250 ファンクションが - 入されます。現在定Aされているファンクションはありません。

```
RFC1205
```

サービスによってサポートされる！能が - 載されている RFC V号。 TN5250 に関する現行の RFC は 1205 です。

TN5250 のサービス登録メッセージ

```
URL: service:tn5250://<addr-spec>:<port-number>
```

属性:

```
(SCOPE=<string>),

(PROTOCOL=<string>),

(RELEASE=<string>),

(PLATFORM=<string>),

(LOAD=<integer 0 to 100>),

(SECURITY=NONE),

(SECURITY=<security>),
(CIPHERSUITES=<Spec1,Spec2,...Specn>),
(FUNCTIONS=NONE),

(RFC1205),

(SERVERNAME=<string>),

(AS400NAME=<string>),
```

LU6.2 サービス・テンプレート

csappc サービス・タイプは SNA APPC アクセスを行います。構成されたローカル LU 定 A は、このサービスで登録されます。

```
LLU = <llu1>,<llu2>,...,<llun>
```

通信サーバーで構成された有効なローカル LU を指定します。

LU6.2 のサービス登録メッセージ

```
URL: service:csappc://<addr-spec>:<port-number>
```

属性:

```
[(SCOPE=<string>),]

(RELEASE=06.00.00),

(PLATFORM=NT),
```

```
(PROTOCOL=IP),

(SERVERNAME=<string>),

(LOAD=<integer 0 to 100>)

[,(LLU=<llu1>,<llu2>,...,<llun>)]
```

付録E. DLL のバージョン情報

32 ビット Windows DLL

以下の 32 ビット Windows DLL には、 DLL のバージョンを = 別するために使用できる情報が含まれています。

- E32APPC.DLL
- WAPPC32.DLL
- WCPIC32.DLL
- WINCSV32.DLL
- WINMS32.DLL
- WINNOF32.DLL
- WINRUI32.DLL
- WINSLI32.DLL

選択可能なキーは以下のとおりです。

- CompanyName
- LegalCopyright
- LegalTrademarks
- ProductName
- ProductVersion
- FileDescription
- InternalName
- FileVersion

注: すべてのキーは ″¥StringFileInfo¥040904E4¥″ バージョン・ブロックの一部であり、変換されません。

これらの情報は、！のように、プログラムを使用して検索することも、あるいは Windows Explorer または Windows NT Explorer を使用して検索することもできます。

1. ＆マウス・ボタンで DLL を選択します。
2. ポップアップ・メニューから「**プロパティー**」を選択します。
3. 「**バージョン**」タブを選択します。

この情報を使用して、DLL が IBM のものか他 R のものか (CompanyName) を = 別し、さらにその DLL が SNA API クライアント用かサーバー用か (ProductName) を = 別するコードを書くことができます。どのバージョンの DLL がインストールされているのか (FileVersion)、またどのバージョンの製品がインストールされているのか (ProductVersion) を = 別することができます。

以下のサンプル C ファンクションは、指定された DLL が IBM 製であるかどうかを = 別するものです。

```
//
//  Function returns TRUE if and only if given pathname is a versioned IBM DLL
//
#include <winver.h>
#define CMPNY_KEY "¥¥StringFileInfo¥¥040904E4¥¥CompanyName"
BOOL bDllFromIBM(char *pcDllPathname)
{
  DWORD  dwBufSize = 0, dwTemp = 0, dwReturnBytes = 0;
  LPVOID pReturnBuffer = NULL;
  VOID  *pVInfoBuffer = NULL;
  BOOL   bRC = FALSE;
 // verify parameters aren't null
 if (!pcDllPathname || !*pcDllPathname)
    return FALSE;
 // get size of Version Info
 dwBufSize = GetFileVersionInfoSize(pcDllPathname, &dwTemp);
 // no version info implies bad parameters or not versioned IBM DLL
 if (!dwBufSize)
    return FALSE;
 // allocate a buffer for the version information (+50 for safety)
 pVInfoBuffer = malloc(dwBufSize + 50);
 // malloc failure
 if (!pVInfoBuffer)
    return FALSE;
 // get version buffer filled
 bRC = GetFileVersionInfo(pcDllName,dwTemp,dwBufSize,pVInfoBuffer);
 // call failed
 if (!bRC)
    return FALSE;
 // get the company name
 bRC = VerQueryValue(pVInfoBuffer, TEXT(CMPNY_KEY), ReturnBuffer, ReturnBytes);
 // not found or empty
 if (!bRC || !dwReturnBytes)
    return FALSE;
 // value should begin with "IBM"
 if (strncmp(pReturnBuffer, "IBM", strlen("IBM")) == 0)
    return TRUE;
    return FALSE;
}
```

付録F. C記事項

本書において、日本では / 表されていない IBM 製品 (! 械およびプログラム)、プログラミングまたはサービスについて言 Z または説明する場合があります。 しかし、このことは、弊 R がこのような IBM 製品、プログラミングまたはサービスを、日本で / 表する意図があることを必ずしも (すものではありません。 本書で、IBM ライセンス・プログラムまたは他の IBM 製品に言 Z している部分があっても、このことは当該プログラムまたは製品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 ! 能的に同等な他 R 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 以外のプログラムまたは製品に関する動作の評価および検査はお R 様の責任で行っていただきます。

IBM および他 R は、本書で説明する g 題に関する特 v 権（特 v 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 v 権、商標権、および著作権について、本書で明 (されている場合を除き、 B 施権、使用権等を v 諾することを意味するものではありません。 B 施権、使用権等の v 諾については、下 - の宛先に、書面にてご照会ください。

〒106-0032 東 ~ 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 T で、 (i) 独 + 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 - に連絡してください。

IBM Corporation
Department TL3B/062
P.O. Box 12195
Research Triangle Park, NC 27709-2195
U.S.A.

本プログラムに関する上 - の情報は、適切な条 o 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。

他 R の製品に関する情報は、それらの製品の提供 T 、それらの製品の / 表資料、またはその他の一 L に入 j 可能な情報源から入 j しました。 IBM はそれらの製品をテストしておらず、パフォーマンスの精度、互換性、またはその他の他 R 製品に関するいかなる - 述をも保証するものではありません。他 R 製品の能力に関するご A 問は、それらの製品の提供 T に送るようお願い致します。

著作権表 (:

本書には、様々なオペレーティング・プラットフォームのプログラミング j 法を例 (するソース言語で書かれたサンプル・アプリケーション・プログラムが G 載され

ています。このサンプル・プログラムは、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準 r したアプリケーション・プログラムの開 / 、使用、N 売、または配布を目的として、いかなる A 0 においても IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。 これらの例は、すべての場合について完全にテストされたものではありません。 IBM はこれらのプログラムの信頼性、可用性、および ! 能について法律上の瑕疵担保責任を含むいかなる明 (または暗 (の保証責任も負いません。

いかなる場合であれ、サンプル・プログラム、またはサンプル・プログラムを改変した二 ! 的著作物をその一部に含む複製物については、これを第三 T に配布する場合、下 - の著作権表 (を行ってください。

(c) (. R 名) (年) このコードは一部分、IBM Corp. のサンプル・プログラムを使用しています。
(c) Copyright IBM Corp. (年) All rights reserved.

付録G. 商標

以下の用語は、IBM Corporation の商標です。

ACF/VTAM	IMS
Advanced Peer-to-Peer Networking	MVS/ESA
AFP	MVS/XA
AIX	NetView
AIXwindows	Operating System/2
Application System/400	OS/2
APPN	OS/400
AS/400	RACF
CallPath	
CallPath/2	SP
CallPath SwitchServer/2	System/370
CICS	S/370
Common User Access	Virtual Machine/Enterprise Systems Architecture
	VM/ESA
IBM	VTAM

二重アスタリスク (**) で (されている会R名、製品名およびサービス名は各Rの商標または登録商標です。

C-bus は Corollary, Inc. の商標です。

ActionMedia、LANDesk、MMX、Pentium、および ProShare は、米国ならびに他の国における Intel Corporation の商標または登録商標です。

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

Microsoft、Windows、Windows NT および Windows 95 のロゴは Microsoft Corporation の登録商標です。

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。

PC Direct は Ziff Communications Company の商標であり、 IBM Corporation がライセンスを得て使用しています。

索引

日本語，Q字，数字，特 l 文字の順に配列されています。なお，濁音と > 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

SC88-5630-01