パーソナル・コミュニケーションズ Windows 版
バージョン 5.9

エミュレーター・プログラミング

SC88-5629-09
(英文原典：SC31-8478-09)

パーソナル・コミュニケーションズ Windows 版
バージョン 5.9

エミュレーター・プログラミング

SC88-5629-09
(英文原典：SC31-8478-09)

ご注意

本書の情報およびそれによってサポートされる製品を使用する前に、 535 ページの『付録 G. 特記事項』に記載する一般情報をお読みください。

本書は IBM パーソナル・コミュニケーションズ Windows 版 バージョン 5.9 (プログラム番号: 5639-I70) に適用されます。特に断りのない限り、後続のすべてのリリースおよびモディフィケーションにも適用されます。

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

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

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

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

(URL は、変更になる場合があります)

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

原　典： SC31–8478–09
Personal Communications for Windows, Version 5.9
Emulator Programming

発　行： 日本アイ・ビー・エム株式会社

担　当： ナショナル・ランゲージ・サポート

第1刷 2006.6

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

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

目次

図

表

本書について

本書は、 IBM® パーソナル・コミュニケーションズ Windows® 版エミュレーター高水準言語アプリケーション・プログラム・インターフェース (EHLLAPI)、動的データ交換 (DDE)、パーソナル・コミュニケーションズ・セッション API (PCSAPI)、およびサーバー/リクエスター・プログラム・インターフェース (SRPI) を使用するために必要なプログラミング情報を提供しています。ホスト・アクセス・クラス・ライブラリーについては、「ホスト・アクセス・クラス・ライブラリー」で説明しています。

EHLLAPI/DDE/PCSAPI を、パーソナル・コミュニケーションズで使用すると、ユーザーおよびプログラマーは、ワークステーション・セッションで実行中のアプリケーション・プログラムから呼び出すことのできる一連の関数によって、ホスト表示スペースにアクセスすることができます。

本書では、*Windows* は、Windows 2000、Windows 2003、および Windows XP を意味します。特定のオペレーティング・システムにのみ適用される情報は、本文中にそのことが明記されます。

本書の対象読者

本書は、本書に記載されている API を使用するアプリケーション・プログラムを作成するプログラマーを対象としています。

読者に Windows の知識と経験があるものと想定して説明しています。Windows については、『詳細情報』にある資料のリストを参照してください。

プログラマーの方は、端末から、または端末エミュレーション・ソフトウェアがインストールされているワークステーションからのホスト・システムへの接続についても熟知している必要があります。

使用する言語およびコンパイラーについても熟知していることを前提としています。どのようにプログラムを作成し、コンパイルし、リンクするかの情報については、『詳細情報』を参照して、使用する特定の言語の適切な解説書を調べてください。

詳細情報

パーソナル・コミュニケーションズ・ライブラリーには、以下の資料が含まれています。

- *CD-ROM インストール・ガイド*
- *はじめに*
- *エミュレーター・ユーザー用解説書*
- *管理者ガイドおよび解説書*
- *エミュレーター・プログラミング*
- *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*
- *システム管理プログラミング*

- *CM Mouse Support User's Guide and Reference*
- *ホスト・アクセス・クラス・ライブラリー*
- *構成ファイル解説書*

印刷された資料に加えて、パーソナル・コミュニケーションズとともに提供されるハイパーテキスト・マークアップ言語 (HTML) 文書があります。

ホスト・アクセス・クラス・ライブラリー
この HTML 文書は、パーソナル・コミュニケーションズを組み込みオブジェクトとして使用するための、ActiveX/OLE 2.0 に準拠するアプリケーションの作成方法について説明しています。

Java 用 ホスト・アクセス Bean™
この HTML 文書は、 JavaBeans™ のセットとして納入される、パーソナル・コミュニケーションズのエミュレーター機能について説明しています。

Java 用 オープン・ホスト・インターフェース・オブジェクト (OHIO)
この HTML 文書は、パーソナル・コミュニケーションズを組み込みオブジェクトとして使用するための、OHIO に準拠する アプリケーションの作成方法について説明しています。

関連資料としては、以下のものがあります。

- *IBM 3270 情報表示システム データストリーム プログラマー用解説書* (N:GA23-0059)
- *IBM 5250 Information Display System Functions Reference Manual* (SA21-9247)

本書で使用されている技術用語の詳細については、 IBM Glossary of Computing Terms (http://www.networking.ibm.com/nsg/nsgmain.htm) を参照してください。

表記規則

33 ページの『第 3 章 EHLLAPI 関数』、 213 ページの『第 5 章 PCSAPI 関数』、 241 ページの『第 6 章 32 ビット環境での DDE 関数』、 395 ページの『付録 E. 16 ビット環境での DDE 関数』の各セクションのはじめに、API 関数または DDE 関数を説明する表があります。この表は、そのセクションで説明している関数を備えている製品について、その関数がサポートされているかどうかを示します。 Yes は、ホスト・タイプについてその関数がサポートされていることを意味し、No は、その関数がサポートされていないことを意味します。例えば、以下の表は、3270 セッションおよび VT セッションではその関数を使用できるが、5250 セッションでは使用できないことを示しています。

3270	*5250*	*VT*
YES	NO	YES

第 1 章 エミュレーター API の概要

IBM パーソナル・コミュニケーションズ製品は、いくつかのアプリケーション・プログラミング・インターフェース (API) を備えています。それぞれのインターフェースは、特定の関数群を備えており、目的に応じて使用することができます。ご使用のアプリケーションの機能要件に最も適したプログラミング・インターフェースを選択してください。アプリケーションによっては、希望する結果を得るために複数のインターフェースを使用する場合もあります。プログラミング・インターフェースには、以下のものがあります。

- **エミュレーター高水準言語 API (EHLLAPI)**: このインターフェースには、ホスト画面上の文字などのエミュレーターの「表示スペース」データにアクセスする機能があります。また、このインターフェースには、ホストへのキー・ストロークの送信、ユーザーが入力したキー・ストロークの代行受信、ホスト・セッション状況の照会、ファイルのアップロードとダウンロードの機能、およびその他の機能もあります。このインターフェースは、ユーザーが直接介入することなく、ホスト画面を読み取ってキー・ストロークを入力する「*自動化操作プログラム*」アプリケーションでしばしば使用します。 33 ページの『第 3 章 EHLLAPI 関数』を参照してください。
  - **IBM 標準 HLLAPI サポート**: これは、ホスト・エミュレーター・セッションへのプログラムによるアクセスを可能にする標準プログラミング・インターフェースです。 7 ページの『第 2 章 IBM 標準 EHLLAPI、IBM 拡張 EHLLAPI、および WinHLLAPI プログラミングの概要』を参照してください。
  - **IBM 拡張 HLLAPI サポート**: このインターフェースは、IBM 標準 HLLAPI インターフェースに基づいたものです。既存の関数のすべてを提供しますが、データ構造は変更後のものを使用します。 7 ページの『第 2 章 IBM 標準 EHLLAPI、IBM 拡張 EHLLAPI、および WinHLLAPI プログラミングの概要』を参照してください。
  - **Windows 高水準言語 API (WinHLLAPI)**: このインターフェースは、IBM 標準 EHLLAPI とほとんど同じ関数を提供し、さらに、Windows 環境を利用した拡張関数をいくつか追加しています。 7 ページの『第 2 章 IBM 標準 EHLLAPI、IBM 拡張 EHLLAPI、および WinHLLAPI プログラミングの概要』を参照してください。
  - **REXX EHLLAPI**: これにより、EHLLAPI を使用するプログラマーは REXX 言語のアプリケーション・プログラムを作成することが可能となります。
- **動的データ交換 (DDE)**: このインターフェースは、ホスト画面を読み取り、キー・ストロークを送信し、関連関数を実行するためのプログラム式手段を提供するという点で EHLLAPI インターフェースに似ています。このインターフェースには、長方形のクリッピング、マウス・イベントの代行受信、エミュレーターのメニュー・バー上のコマンドの追加または除去を行うなどのエミュレーターにアクセスする追加関数があります。 241 ページの『第 6 章 32 ビット環境での DDE 関数』を参照してください。

- **パーソナル・コミュニケーションズ・セッション API (PCSAPI)**: このインターフェースは、エミュレーター・セッションおよび設定の開始、停止、制御を行うために使用します。 213 ページの『第 5 章 PCSAPI 関数』を参照してください。

  パーソナル・コミュニケーションズ バージョン 5.9 では、ページ設定およびプリンター設定の制御と検索を行うための関数が追加されています。 222 ページの『ページ設定関数』および 229 ページの『プリンター設定関数』を参照してください。
- **サーバー/リクエスター・プログラミング・インターフェース (SRPI)**: このインターフェースは、ホスト・システムで実行される IBM 拡張接続機能 (ECF) アプリケーションとともに使用します。この API には、リモート・サーバー・プログラムに対する同期呼び出し/戻りインターフェースを作成するための関数があります。 357 ページの『第 8 章 サーバー/リクエスター・プログラム・インターフェース (SRPI) のサポート』を参照してください。
- **IBM パーソナル・コミュニケーションズのホスト・アクセス・クラス・ライブラリー (ECL)**: ECL は、アプリケーション・プログラマーおよびスクリプト言語ライターが簡単かつ迅速にホスト・アプリケーションにアクセスできるようにするためのオブジェクトのセットです。パーソナル・コミュニケーションズでは、3 つの別個の ECL 層 (C++ オブジェクト、 ActiveAutomation (OLE)、および LotusScript Extension (LSX)) をサポートしています。詳細については、「ホスト・アクセス・クラス・ライブラリー *(HACL)*」を参照してください。

API ヘッダー・ファイルの使用

アプリケーション・プログラムに API ヘッダー・ファイルを組み込む前に、オペレーティング・システムのヘッダー・ファイルを組み込む必要があります。次に例を示します。

```
#include <windows.h>      // Windows main header
#include "pcsapi.h"       // PComm PCSAPI header
...
```

クリティカル・セクション

ご使用のプログラムでエミュレーター API を呼び出す場合、クリティカル・セクション (**EnterCriticalSection** 関数) は慎重に使用してください。クリティカル・セクション内でのエミュレーター API 呼び出しは行わないでください。アプリケーションの 1 つのスレッドがクリティカル・セクションを設定する際に、別のスレッドがエミュレーター API 呼び出し内にある場合、ユーザーがクリティカル・セクションを終了するまで、その呼び出しは延期されます。

API 呼び出しの処理中は、その呼び出しが完了する、または着信データを待つことが必要となるまで、すべてのシグナル (数値演算コプロセッサー・シグナルを除く) は遅らされます。また、アプリケーションが処理中の API 呼び出しを完了するまで、別のプロセスから発行された **TerminateProcess** は保留されます。

スタック・サイズ

エミュレーター API は、実行時に呼び出し側プログラムのスタックを使用します。オペレーティング・システム、アプリケーション、および API のすべてが、ダイナミック変数と関数仮引き数のためのスタック・スペースを必要とします。 API 呼び出しの時点で、少なくとも 8196 バイト (8K) のスタック・スペースが使用可能である必要があります。アプリケーション・プログラム側で、API 用のスタック・スペースが十分に使用可能となるように保証する必要があります。

16 ビット Windows EHLLAPI プログラムの実行

16 ビット EHLLAPI DLL を使用する複数の 16 ビット Windows EHLLAPI タスクを実行する場合、各 16 ビット EHLLAPI タスクは別々の NTVDM のもとで実行する必要があります。これは、次のいずれの方法でも実行できます。

- 16-ビット EHLLAPI アプリケーションを開始するショートカットは、プログラムが分離したメモリー・スペース (NTVDM) で稼働するように指定する必要がある。
- コマンド・プロンプトまたはバッチ・プログラムで hllapi16.exe のような 16 ビット EHLLAPI アプリケーションを開始する場合、次のコマンドを入力します。

  ```
  start /separate hllapi16.exe
  ```

- Win32 アプリケーションが Windows API CreateProcess を使用して 16 ビット EHLLAPI アプリケーションを作成する場合、そのアプリケーションは CREATE_SEPARATE_WOW_VDM という名前のプロセス作成フラグを使用する必要がある。

Windows x64 プラットフォームのサポート

Microsoft® Windows Server 2003 の x64 ベース・バージョンおよび Microsoft Windows XP Professional x64 Edition は、64 ビット・プログラムに最適化されていますが、32 ビット・ドライバーまたは 16 ビット・アプリケーションはサポートしません。

これらのプラットフォームの場合、パーソナル・コミュニケーションズ は以下のライブラリーをインストールしません。

- DOS EHLLAPI
- 16 ビット API サポート:
  - 標準 EHLLAPI 16 ビット・インターフェース
  - WinHLLAPI 16 ビット・インターフェース
  - PCSAPI 16 ビット・インターフェース
  - SRPI 16 ビット・インターフェース

サンプル・プログラム

パーソナル・コミュニケーションズ API の使用方法を例示するために、いくつかのサンプル・プログラムが提供されています。サンプル・プログラムをインストールする場合は、サンプル・プログラムはデフォルトのディレクトリー ¥SAMPLES にインストールされます。

注: IBM は、これらのファイルをあくまで「参考用」として提供しており、販売または特定の用途に対する適合性の保証を含め、またこれに限らず、いかなる種類の保証も行いません。

サンプル・プログラムには、以下のパーソナル・コミュニケーションズ API 用のソースおよびサポート・ファイルが含まれています。

- エミュレーター高水準言語プログラミング・インターフェース (EHLLAPI)
- 動的データ交換 (DDE)
- サーバー/リクエスター・プログラム・インターフェース (SRPI)
- PCSAPI 関数

以下のファイルが、¥SAMPLES ディレクトリーにインストールされます。

表 1. サンプル・プログラム・ファイル

ファイル名	説明
DDE_C.H	DDE インクルード・ファイル
EHLAPI32.H	IBM 標準 32 ビットの EHLLAPI インクルード・ファイル
WHLLAPI.H	WinHLLAPI 16 ビットのインクルード・ファイル
HAPI_C.H	EHLLAPI インクルード・ファイル
PCSAPI.H	PCSAPI インクルード・ファイル
PCSCALLS.LIB	インポート・ライブラリー (標準インターフェース用)
PCSCAL32.LIB	インポート・ライブラリー (拡張インターフェース用)
EHLAPI32.LIB	インポート・ライブラリー (IBM 標準 32 ビットの EHLLAPI インターフェース用)
WHLLAPI.LIB	インポート・ライブラリー (WinHLLAPI 16 ビット・インターフェース用)
WHLAPI32.LIB	インポート・ライブラリー (WinHLLAPI 32 ビット・インターフェース用)
UUCCPRB.H	SRPI インクルード・ファイル

以下のサブディレクトリーが、¥SAMPLES ディレクトリーに作成されます。

表 2. サンプル・プログラム・サブディレクトリー

ファイル名	説明
DDXFER	ファイル転送用に「ドラッグ・アンド・ドロップ」アプリケーションを作成するための EHLLAPI の使用法。
ECL	HACL サンプル・ファイル
HLLSMP	EHLLAPI を使用して、キー・ストロークを要求し VM システムにログオンする方法。
LISTFILE	DDE で LOAD ボタンを使用して、ホストからファイルを転送する方法の例。
PCSMAIN	PCSAPI を使用して、セッションの開始と停止、セッション状況の照会、およびセッションのプロファイルの照会を行う方法の例。
SPL2FILE	DDE を使用して、iSeries™, eServer™ i5、または System i5™ スプール・ファイルを、ASCII ファイルとして PC に保管するプログラム。

表 2. サンプル・プログラム・サブディレクトリー (続き)

ファイル名	説明
SRPSMP	サーバー/リクエスター・プログラミング・インターフェース (SRPI) の使用例。
VBDDE	VBDDE サンプル・ファイル
VBHLLAPI	VBHLLAPI サンプル・ファイル
VBPCSAPI	VBPCSAPI サンプル・ファイル

第 2 章 IBM 標準 EHLLAPI、IBM 拡張 EHLLAPI、および WinHLLAPI プログラミングの概要

本章では、高水準言語で書かれたアプリケーションに IBM 標準 EHLLAPI (16 ビットおよび 32 ビット) 関数、WinHLLAPI (16 ビットおよび 32 ビット) 関数、および IBM 拡張 32 ビット EHLLAPI (EHLAPI32) 関数を組み込むのに必要な情報を提供します。呼び出し形式、メモリー割り振りに関する考慮事項、インターフェースの初期化の方法、およびアプリケーションのコンパイルとリンクの方法について詳しく説明します。また、簡単なサンプル EHLLAPI プログラム、およびこのプログラムを作成するのに使用するコンパイル/リンクに関する命令も含まれています。この章の最後では、EHLLAPI インターフェースについて考えられる、一連の用途 (シナリオ) を説明します。

EHLLAPI アプリケーションとは、EHLLAPI インターフェースを使用してホスト 3270/5250/VT 表示スペースにアクセスするアプリケーション・プログラムのことです。表示スペースには、可視のエミュレーター文字データ、フィールドおよび属性データ、キー・ストローク・データ、その他の情報が含まれます。

EHLLAPI の概要

以下は、HLLAPI プログラミング・インターフェースの概要です。

IBM 標準 EHLLAPI

EHLLAPI は、標準プログラミング・インターフェースです。これを使用することで、ホスト・エミュレーター・セッションにプログラムでアクセスすることができます。ホスト画面データ (文字および属性など) の読み取り、キー・ストロークの送信、その他のエミュレーター関連の関数を実行するための関数が用意されています。

EHLLAPI インターフェースは、呼び出し点が 1 つのインターフェースです。 1 つの呼び出し可能 API があり、これを介してすべての EHLLAPI 関数を要求します。このインターフェースを呼び出すたびに、アプリケーションは、要求された関数を識別する関数番号、データ・バッファーを指すポインター、データ・バッファー長を指すポインター、および戻りコードを指すポインターを提供します (8 ページの『EHLLAPI 呼び出し形式』を参照してください)。

WinHLLAPI

WinHLLAPI は、よく知られている EHLLAPI.API を基にしたものです。これは、既存の関数をすべて含み、さらに、Windows のメッセージ・ドリブン環境を利用する拡張関数が追加されています。 IBM パーソナル・コミュニケーションズ EHLLAPI インターフェースのユーザーは、WinHLLAPI 拡張関数を組み込まない場合には機能上の差に気がつきません。

WinHLLAPI 拡張関数および EHLLAPI 形式から逸脱する関数については、 201 ページの『第 4 章 WinHLLAPI 拡張関数』で説明します。共通関数の詳細については、 33 ページの『第 3 章 EHLLAPI 関数』を参照してください。

WinHLLAPI 対 IBM 標準 EHLLAPI

WinHLLAPI の入り口記号は、専有的に **WinHLLAPI** です。WinHLLAPI のインプリメンテーションに切り替えたいと考えている EHLLAPI ユーザーは、**hllapi** 標準入り口からの変更が必要です。新規ユーザーは、 33 ページの『第 3 章 EHLLAPI 関数』に記載されているすべての指示に従い、さらに、標準の **hllapi** 入り口の代わりに **WinHLLAPI** 入り口を使用してください。

IBM 拡張 EHLLAPI 対 IBM 標準 EHLLAPI

IBM 拡張 EHLLAPI は、よく知られている EHLLAPI API に基づいています。これは、既存の関数をすべて含んでいますが、32 ビット環境を利用し、変更されたデータ構造を使用します。標準インターフェースのユーザーが IBM 拡張 32 ビット EHLLAPI への切り替えを望む場合、必要なのは 1 番目、3 番目、および 4 番目のパラメーター内の LPWORD を LPINT に変更することだけです。新規ユーザーは、以下のセクションに記載されているプロシージャーを使用してください。

言語

「Pascal」の呼び出し規則で DLL 内のエントリー・ポイントを呼び出すことができるプログラミング言語であれば、EHLLAPI 関数を実行するために使用できます。ただし、パーソナル・コミュニケーションズ EHLLAPI ツールキットでは、 C++ 言語用のヘッダー・ファイルと関数プロトタイプだけが提供されています。他の言語を使用する場合には、データ構造レイアウトと呼び出し規則を十分に理解することが必要となります。 EHLLAPI ツールキットは、以下の C/C++ コンパイラーをサポートしています。

- IBM VisualAge® for C/C++
- Microsoft Visual C/C++ バージョン 4.0 およびそれ以降

その他のほとんどの C/C++ コンパイラーもツールキットとともに使用できます。

EHLLAPI C/C++ アプリケーションには、パーソナル・コミュニケーションズ EHLLAPI ヘッダー・ファイル (HAPI_C.H) を組み込む必要があります。このファイルはデータ構造のレイアウトを定義し、EHLLAPI エントリー・ポイントのプロトタイプを提供します。

注: 16 ビットと 32 ビットのアプリケーションのデータ構造レイアウトは、同じではありません (25 ページの『標準および拡張インターフェースの考慮事項』を参照してください)。

EHLLAPI 呼び出し形式

EHLLAPI エントリー・ポイント (**hllapi**) は、常に以下の 4 つのパラメーターを用いて呼び出されます。

1. EHLLAPI 関数番号 (入力)
2. データ・バッファー (入出力)

3. バッファー長 (入出力)
4. 位置 (入力) ; 戻りコード (出力)

IBM 標準 EHLLAPI のプロトタイプは次のとおりです。

```
[long hllapi (LPWORD, LPSTR, LPWORD, LPWORD);
```

IBM 拡張 EHLLAPI のプロトタイプは次のとおりです。

```
[long hllapi (LPINT, LPSTR, LPINT, LPINT);
```

各パラメーターは、値ではなく、*参照* によって受け渡されます。したがって、関数呼び出しの各パラメーターは、値そのものではなく、値を指すポインター となっている必要があります。例えば、EHLLAPI Query Session Status 関数の呼び出しの正しい例は、以下のとおりです。

```
#include "hapi_c.h"
struct HLDQuerySessionStatus QueryData;
int    Func, Len, Rc;
long   Rc;

memset(QueryData, 0, sizeof(QueryData)); // Init buffer
QueryData.qsst_shortname = 'A';          // Session to query
Func = HA_QUERY_SESSION_STATUS;          // Function number
Len  = sizeof(QueryData);                // Len of buffer
Rc   = 0;                                // Unused on input

hllapi(&Func, (char *)&QueryData, &Len, &Rc);  // Call EHLLAPI
if (Rc != 0) {                            // Check return code
  // ...Error handling
}
```

hllapi 呼び出しのすべてのパラメーターはポインターであり、EHLLAPI 関数の戻りコードは、この関数の値としてではなく、4 番目のパラメーターの値で戻されます。例えば、次の例は正しく**ありません**。

```
if (hllapi(&Func, (char *)&QueryData, &Len, &Rc) != 0) { // WRONG!
  // ...Error handling
}
```

hllapi 関数は、IBM 標準および拡張 EHLLAPI の場合には **long** データ・タイプを、WinHLLAPI の場合には **void** データ・タイプを戻すように定義されていますが、その値は未定義なので使用することはできません。

hllapi 呼び出しの 2 番目から 4 番目までのパラメーターは、アプリケーションに情報を戻すことができます。各 EHLLAPI 関数の説明では (情報がある場合)、どのような情報がこれらのパラメーターに戻されるかを示しています。

データ構造

多くの EHLLAPI 関数は、アプリケーション・プログラムとの間の情報の受け渡しのために、定様式データ構造を使用します。各関数の説明では、データ構造のレイアウトを示します。EHLLAPI 関数との間で受け渡されるデータは、記述されているのとまったく同じ、すなわちバイト単位で同じようにストレージ内に存在する必要があります。構造レイアウトは、IBM 標準 および WinHLLAPI の 16 ビットおよび 32 ビットのすべてのアプリケーションで同じであることに注意してください。IBM 拡張 32 ビット・アプリケーションのデータ構造は、4 バイトの位置合わせでパックされます。

適切なデータの位置合わせとレイアウトを確保するために、提供されているヘッダー・ファイルとデータ構造の定義を使用することを*特にお勧めします*。 次の例は技術的には可能ですが、*お勧めしません*。

```
char QueryData[20];  // Not recommended
...
Func = HA_QUERY_SESSION_STATUS;
hllapi(&Func, QueryData, &Len, &Rc);
if (QueryData[13] == 'F') {
  // ...this is a 5250 session
}
```

この関数は次のように記述することをお勧めします。

```
#include "hapi_c.h"
struct HLDQuerySessionStatus QueryData;  // Recommended
...
Func = HA_QUERY_SESSION_STATUS;
hllapi(&Func, (char *)&QueryData, &Len, &Rc);
if (QueryData.qsst_sestype == 'F') {
  // ...this is a 5250 session
}
```

メモリー割り振り

EHLLAPI 関数はメモリーの割り振りや、解放は行いません。 **hllapi** エントリー・ポイントを呼び出す前に、アプリケーション・プログラムはバッファー・スペースを必要とする EHLLAPI 関数に対して、そのスペースを事前に割り振る必要があります。バッファー・スペースは、次のようにダイナミック変数として事前に割り振ることができます。

```
struct HLDQuerySessionStatus QueryBuff;
```

あるいは、次に示すように、C ライブラリーまたはオペレーティング・システム関数を呼び出して割り振ることもできます。

```
struct HLDQuerySessionStatus *QueryBuff;
...
QueryBuff = malloc(sizeof(struct HLDQuerySessionStatus));
```

いずれの場合でも、EHLLAPI 関数を呼び出す前に十分なバッファー・スペースを割り振ること、および不要になったバッファーを解放することは、アプリケーション側の責任で行う必要があります。

EHLLAPI 戻りコード

EHLLAPI 関数 (**Convert Position** や **RowCol** (99) 関数を除く) は、**hllapi** 関数呼び出しの 4 番目のパラメーターに完了コードまたは戻りコードを戻します。この戻りコードは、要求した関数が正常に実行されたかどうかを示します。

それぞれの関数の説明において特に断りのない限り、次の表は、各戻りコードの値の意味を示しています。一部の関数では、これらの戻りコードの解釈が若干異なる場合があります。詳細については、個々の関数に関する説明を参照してください。

表 3. EHLLAPI 戻りコード

戻りコード	説明
0	関数は正常に実行されました。または最後の呼び出しが発行された後、更新されていません。
1	無効なホスト表示スペース ID が指定されました。指定されたセッションが接続されていないか、存在しないか、または、論理プリンター・セッションです。
2	パラメーター・エラーが発生したか、または無効な関数番号が指定されました。 (詳細については、個々の関数を参照してください。)
4	宛先表示スペースが使用中の状態 (X CLOCK 状態 (X []) または X SYSTEM 状態) のため関数の実行が禁止されました。
5	戻りコード 4 で述べた理由以外の何らかの理由で関数の実行が禁止されました。
6	無効なパラメーターの指定 (例えば、切り捨ての原因となる長さエラー) によりデータ・エラーが発生しました。
7	指定された表示スペースが無効です。
8	関数手順エラー (例えば、矛盾した関数の使用または前提となる関数の欠如) が発生しました。
9	システム・エラーが発生しました。
10	この関数は EHLLAPI では使用できません。
11	このリソースは使用できません。
12	このセッションは停止しました。
24	ストリングが見つからないか、または表示スペースが定様式化されていません。
25	キー・ストロークが入力キューにありませんでした。
26	ホスト・イベントが発生しました。詳細については、**Query Host Update** (24) を参照してください。
27	ファイル転送が、Ctrl+Break コマンドにより終了しました。
28	フィールドの長さが 0 でした。
31	キー・ストローク・キューがオーバーフローしました。キー・ストロークは失われました。
32	アプリケーションは、通信のためにこのセッションにすでに接続しています。
33	予約済み。
34	ホストに送ったメッセージが取り消されました。
35	ホストから送られたメッセージが取り消されました。
36	ホストとの接続が失われました。
37	インバウンド通信が使用不可になっています。
38	要求された関数は実行を完了していません。
39	他の DDM セッションがすでに接続されています。
40	接続の切断の試行は正常に実行されましたが、切断時にまだ完了していない非同期要求がありました。
41	要求されたバッファーは、他のアプリケーションによって使用されています。

表 3. EHLLAPI 戻りコード (続き)

戻りコード	説明
42	一致する未処理の要求がありません。
43	API がすでに他の EHLLAPI アプリケーションによってロックされていたか (ロック要求の場合)、または API がロックされていませんでした (アンロック要求の場合)。

コンパイルとリンク

EHLLAPI 関数を使用するアプリケーションは、適切な関数プロトタイプ、定数、およびデータ構造定義を得るために、該当するヘッダー・ファイルを組み込む必要があります。このヘッダー・ファイルは、サポートされている C/C++ コンパイラー (8 ページの『言語』を参照) のいずれかで使用できます。異なったコンパイラーまたは言語を使用する場合は、ユーザー独自の同等の定義と構造を提供する必要があります。

エントリー・ポイントがどのように解決されるかによって、アプリケーション・プログラムをリンクする方法として、2 通りが考えられます。最も簡単な方法は、アプリケーションをパーソナル・コミュニケーションズ・ライブラリーに静的にリンクする方法です。この方法では、エントリー・ポイントはリンク時に解決されます。オペレーティング・システムは、アプリケーションの開始時に正しい DLL をアプリケーションと共にロードします。エントリー・ポイントにリンクするもう 1 つの方法は、ダイナミック・リンクを実行するという方法です。この場合、アプリケーションは、オペレーティング・システム呼び出しを使用して正しい DLL をロードし、実行時にエントリー・ポイント・アドレスを獲得します。

下記の表は、使用するヘッダー・ファイル、静的リンクの場合に使用する .LIB、および動的ロードの場合に使用する .DLL を示しています。

インターフェース	エントリー・ポイント	ヘッダー・ファイル	LIB	DLL
IBM 標準 (16 ビット)	hllapi	hapi_c.h	PCSCALLS.DLL	PCSHLL.DLL
IBM 標準 (32 ビット)	hllapi	ehlapi32.h	EHLAPI32.LIB	EHLAPI32.DLL
IBM 拡張 (32 ビット)	hllapi	hapi_c.h	PCSCAL32.LIB	PCSHLL32.DLL
WinHLLAPI (16 ビット)	winhllapi	whllapi.h	WHLLAPI.LIB	WHLLAPI.DLL
WinHLLAPI (32 ビット)	winhllapi	whllapi.h	WHLAPI32.LIB	WHLAPI32.DLL

静的リンク方式

次に示すように、アプリケーションは静的リンクを使用して、必要なときに **hllapi** エントリー・ポイントを簡単に呼び出すことができます。

```
#include "hapi_c.h"
int HFunc, HLen, HRc;                          // Function parameters
char HBuff[1];                                 // Function parameters
...
HFunc = HA_RESET_SYSTEM;                       // Run EHLLAPI function
HLen  = 0;
HRc   = 0;
```

```
hllapi(&Func, HBuff, &HLen, &HRc);
if (HRc != 0) {
  // ... EHLLAPI access error
}
```

アプリケーションがリンクされるとき、適切なパーソナル・コミュニケーションズ・ライブラリー・ファイルをアプリケーションの実行可能コードにリンクする必要があります。例えば、次のリンク・コマンドを使用することができます (IBM VisualAge C/C++)。

```
ilink /de /noe pcscal32.lib sample.obj
```

このように作成されたアプリケーションをオペレーティング・システムがロードするとき、パーソナル・コミュニケーションズの EHLLAPI モジュールは自動的にロードされます。

ダイナミック・リンク方式

アプリケーションは、ダイナミック・リンク方式を使用して、実行時にオペレーティング・システムに対する呼び出しを行い、パーソナル・コミュニケーションズ EHLLAPI モジュールをロードし、その中の **hllapi** エントリー・ポイントを見つけます。この方式では、アプリケーションにより多くのコードが必要となりますが、アプリケーションはエラー条件をより一層制御できるようになります。例えば、パーソナル・コミュニケーションズの EHLLAPI モジュールが見つからない場合、アプリケーションはユーザーに特定のエラー・メッセージを表示することができます。

ダイナミック・リンクを使用するには、アプリケーションは適切なパーソナル・コミュニケーションズ・モジュールをロードして、エントリー・ポイントを見つける必要があります。エントリー・ポイントは、名前ではなく、その序数で見つけられるようにすることをお勧めします。その序数は、ヘッダー・ファイルに定義されています。次の 32 ビット Windows コードは、IBM 標準 32 ビット EHLLAPI モジュールをロードし、**hllapi** エントリー・ポイントを見つけて、EHLLAPI 関数呼び出しを行います。

```
#include "hapi_c.h"

HMODULE Hmod;                                          // Handle of PCSHLL32.DLL
long (APIENTRY hllapi)(int *, char *, int *, int *);  // Function pointer
int HFunc, HLen, HRc;                                  // Function parameters
char HBuff[1];                                         // Function parameters

Hmod = LoadLibrary("PCSHLL32.DLL");                    // Load EHLLAPI module
if (Hmod == NULL) {
  // ... Error, cannot load EHLLAPI module
}

hllapi = GetProcAddress(Hmod, MAKEINTRESOURCE(ord_hllapi));
                                                       // Get EHLLAPI entry point
if (hllapi == NULL) {
  // ... Error, cannot find EHLLAPI entry point
}

HFunc = HA_RESET_SYSTEM;                               // Run EHLLAPI function
HLen  = 0;
HRc   = 0;
```

```
(*hllapi)(&Func, HBuff, &HLen, &HRc);
if (HRc != 0) {
  // ... EHLLAPI access error
}
```

マルチスレッド化

IBM 拡張 EHLLAPI (32 ビット) および IBM 標準 EHLLAPI 16 ビットは、プロセスごとに接続を行います。すべてのスレッドが、同じ接続済みホスト・セッションにアクセスします。接続を実行するスレッドは、切断も実行する必要があります。

IBM 標準 EHLLAPI (32 ビット) および WinHLLAPI はスレッドごとに接続を行います。各スレッドが各自の接続を維持する必要があります。これにより、マルチスレッド化プロセスが、一度に複数の接続済みホスト・セッションとの接続を維持することが可能となります。これによって、WinHLLAPI プログラムを使用して異なるホスト間でデータを調整する際に、マルチプロセス体系の必要がなくなります。また、必要に応じて、接続および切断の負荷を個々のスレッドにかけることも可能です。

表示スペース

多くの EHLLAPI 関数は、その関数に使用するホスト・エミュレーターを指示するために、*表示スペース ID (PSID)* を必要とします。 (これは、*短縮セッション ID* とも呼ばれます。) 表示スペース ID は、A から Z までの範囲の 1 つの文字です。

IBM 拡張 32 ビット・インターフェースの表示スペース ID

IBM 拡張 EHLLAPI アプリケーションの場合、セッション ID は 3 バイト追加することによって拡張されています。これらの拡張セッション・バイトは、将来の互換性のためにゼロに設定しておく必要があります。上記の設定を最も簡単に実行するには、EHLLAPI バッファーを必要な情報で埋める前に、そのバッファーの内容をすべて 2 進ゼロに設定します。例えば、次の例は、セッション B の状況を照会するのに使用できます。

```
#include "hapi_c.h"
int HFunc, HLen, HRc;                                  // Function parameters
struct HLDPMWindowStatus StatusData;                   // Function parameters

Func = HA_PM_WINDOW_STATUS;
HLen = sizeof(StatusData);
HRc  = 0;

// Set data buffer to zeros and fill in request
memset(&StatusData, 0x00, sizeof(StatusData));
StatusData.cwin_shortname = 'B';     // Short session ID
StatusData.cwin_option    = 0x02;    // Query command

hllapi(&Func, (char *)&StatusData, &HLen, &HRc);
```

表示スペースのタイプ

エミュレーター・セッションは、ディスプレイ・セッションまたはプリンター・セッションとして構成することができます。EHLLAPI アプリケーションは、PC400

のプリンター・セッションまたはルーター・セッションと接続することはできません。**Query Sessions (10)** 関数を使用して、特定のセッションのタイプを判別することができます。

表示スペースのサイズ

エミュレーター・ディスプレイ・セッションは、1920 バイト (24x80 の画面サイズ) から 9920 バイト (62x160 の画面サイズ) までの画面サイズの範囲で構成することができます。 **Copy PS to String (8)** のような一部の EHLLAPI 関数は、表示スペース全体を保持できるくらいのストレージを割り振るよう、アプリケーションに要求します。特定のセッションのための表示スペースのサイズは、 **Query Session Status (22)** 関数を使用して得ることができます。

表示スペース ID

EHLLAPI 関数は、一度に 1 つの表示スペースとのみ対話します。表示スペース ID (PSID) は、関数を操作する特定の表示スペースを識別するために使用します。

関数によっては、**Connect Presentation Space** (1) 関数への、先行する呼び出しに PSID が入っている場合があります。それ以外の関数では、PSID は呼び出し側のデータ・ストリング・パラメーターに入っています。

ホスト接続表示スペース

ホスト表示スペース (またはセッション) への接続は、 **Connect Presentation Space** (1) および **Disconnect Presentation Space** (2) 関数によって制御されます。いくつかの関数は、接続の状況によって実行できるかどうかが決まります。また、接続の状況は PSID の定義方法にも影響を与えます。次に、ホスト表示スペースへの接続状況を制御する方法について説明します。

- 常に、ホスト接続表示スペースが 1 つも存在しないか、ホスト接続表示スペースが 1 つだけ存在するかのいずれかになります。
- デフォルトのホスト接続表示スペースはありません。
- 接続すると、ホスト接続表示スペースが 1 つだけ存在するようになります。接続されたホスト表示スペースは、接続関数の呼び出しデータ・ストリング・パラメーター内で識別されます。
- 接続を行う呼び出し以降の呼び出しは、呼び出しを行うごとに切断することなく実行できます。この場合も、ホスト接続表示スペースが 1 つだけ存在します。接続されたホスト表示スペースは、この場合も接続関数の呼び出しデータ・ストリング・パラメーター内で識別されます。
- 切断すると、ホスト接続表示スペースはなくなります。この規則は、接続を行う呼び出しを連続して行った場合にも、接続を行う呼び出しを一度だけ行った場合にも適用されます。
- 論理プリンター・セッションに接続することはできません。

表示スペース ID の処理

PSID は、関数を操作するホスト表示スペース (またはセッション) を指定するために使用されます。 PSID の処理方法は、次の 2 つの要因の影響を受けます。

1. PSID を指定するために使用される方法

a. **Connect Presentation Space** (1) 関数への、先行する呼び出しの呼び出しデータ・ストリング・パラメーターとして。
b. 実行する関数の呼び出しデータ・ストリングの 1 文字として。処理は、その文字が次のどちらであるかによって異なります。
   - *A* から *Z* までの文字
   - ブランクまたは NULL

2. ホスト表示スペースへの接続状況

以下の項では、これら 2 つの要因の種々の組み合わせに対する PSID の処理方法を説明します。

接続を必要とする関数に対する PSID の処理

関数によっては、ホスト接続表示スペースとのみ対話するものがあります。これらの関数では前提呼び出しとして **Connect Presentation Space** (1) 関数が必要です。これらの関数の PSID は、 **Connect Presentation Space** (1) 関数および **Disconnect Presentation Space** (2) 関数によって次のように判別されます。

- ホスト接続表示スペースが存在しない場合、これらの関数はどの表示スペースとも対話を行いません。戻りコード 1 が生成されます。
- ホスト接続表示スペースが 1 つある場合、これらの関数は、最後の **Connect Presentation Space** (1) 関数呼び出しの呼び出しデータ・ストリング・パラメーターに指定された表示スペースと対話を行います。

接続を必要としない関数に対する PSID の処理

一部の関数は、接続されているかいないかにかかわらず、ホスト表示スペースと対話します。これらの関数では、PSID を呼び出しデータ・ストリング・パラメーターに指定することができます。このような機能を次に示します。

- **Connect Presentation Space** (1)
- **Convert Position RowCol** (99)
- **Get Key** (51)
- **Post Intercept Status** (52)
- **Query Close Intercept** (42)
- **Query Host Update** (24)
- **Query Session Status** (22)
- **Start Close Intercept** (41)
- **Start Host Notification** (23)
- **Start Keystroke Intercept** (50)
- **Stop Close Intercept** (43)
- **Stop Host Notification** (25)
- **Stop Keystroke Intercept** (53)

最初の 2 つを除く上記の関数では、次のいずれかを使用して PSID を指定できます。

- *A* から *Z* までの文字
- ブランクまたは NULL

最初の 2 つの関数では、文字を使用して PSID を指定する必要があります。

ホスト接続表示スペースが存在しない場合、次の規則が適用されます。

- ブランクまたは NULL ではなく、文字を使用して PSID を指定すると、その関数は任意のホスト表示スペースと対話を行うことができます。
- ブランクまたは NULL を使用して PSID を指定すると、戻りコード 1 が生成されます。関数は実行されません。
- 文字を使用して PSID を指定すると、PS 接続要求を除き、ホスト接続表示スペースは確立されません。

ホスト接続表示スペースが 1 つだけ存在する場合は、次の規則が適用されます。

- 文字を使用して PSID を指定すると、その関数は任意のホスト表示スペースと対話を行うことができます。
- ブランクまたは NULL を使用して PSID を指定すると、 **Connect Presentation Space** (1) 関数に対する最後の呼び出しで識別された表示スペース内で関数が実行されます。
- 文字を使用して PSID を指定すると、PS 接続要求を除き、ホスト接続表示スペースの確立済み PSID は変更されません。

プリンター・セッションでは、次の関数を使用できます。

- **Start Host Notification** (23)
- **Query Host Update** (24)
- **Stop Host Notification** (25)

EHLLAPI 表示スペースのプロセス間での共用

アプリケーションが共用をサポートしている場合 (つまり、アプリケーションが一緒に動作するよう開発されているか、またはお互いの動作が予測可能な場合[1])、複数の EHLLAPI アプリケーションで 1 つの表示スペースを共用することができます。共用をサポートするアプリケーションを判別するには、EHLLAPI アプリケーションに次のタイプの 1 つを指定します。

- 監視
- 読み取り特権のある排他的書き込み
- 読み取り特権のない排他的書き込み
- 共用書き込み
- 読み取り

共用アクセスのタイプは、 **Set Session Parameters** (9) 関数呼び出しで、各関数に対して次の読み取りおよび書き込み共用オプションを設定することにより定義できます。

1. これは、2 つの EHLLAPI プログラムが同時に同じ表示スペースを取得しようとすることはない、表示スペースが使用可能になるまでプログラムを待ち状態にすることができるようなロジックがプログラムに組み込まれている、あるいは、アプリケーションは他のアプリケーションをロックアウトするような方法ではセッションを使用しないということです。

SUPER_WRITE

アプリケーションは、共用が可能で書き込みアクセス許可を有する他のアプリケーションに同時に同じ表示スペースに接続することを許可します。この関数を呼び出したアプリケーションでは、監視タイプの関数が実行されますが、表示スペースを共用する他のアプリケーションに対してエラーは生成されません。

WRITE_SUPER

アプリケーションは、書き込みアクセスを要求し、監視アプリケーションだけにその表示スペースへ同時に接続することを許可します。これはデフォルト値です。

WRITE_WRITE

アプリケーションは、書き込みアクセスを要求し、その動作が予測可能なパートナーまたは他のアプリケーションに表示スペースを共用することを許可します。

WRITE_READ

アプリケーションは、書き込みアクセスを要求し、読み取り専用関数を実行する他のアプリケーションに表示スペースを共用することを許可します。また、表示スペースのコピーおよび他の読み取り専用操作を通常通り実行することもできます。

WRITE_NONE

アプリケーションは、表示スペースを排他使用します。監視アプリケーションも含めて、他のアプリケーションは、表示スペースを共用できません。また、表示スペースのコピーおよび他の読み取り専用操作を通常通り実行することもできます。

READ_WRITE

アプリケーションは、表示スペースをモニターするための読み取りアクセスのみを要求し、読み取りまたは書き込み (あるいはその両方の) 関数を実行する他のアプリケーションに表示スペースを共用することを許可します。また、表示スペースのコピーおよび他の読み取り専用操作を通常通り実行することもできます。

注: 表示スペースは、1 つのプロセスのスレッドの間では共用できません。

表 4. EHLLAPI 読み書き共用オプションの組み合わせ

呼び出すアプリケーション	Super_Write	Write_Super	Write_Write	Write_Read	Write_None	Read_Write
Super_Write	YES	YES	YES	NO	NO	YES
Write_Super (デフォルト値)	YES	NO	NO	NO	NO	NO
Write_Write	YES	NO	YES	NO	NO	YES
Write_Read	NO	NO	NO	NO	NO	YES
Write_None	NO	NO	NO	NO	NO	NO
Read_Write	YES	NO	YES	YES	NO	YES

互換性のある読み書きアクセス・オプションを指定することの他に、一緒に動作しても他のアプリケーションに同じ表示スペースを使用させないようにするアプリケーションでは、**Set Session Parameters** (9) 関数呼び出しの中で、キーワード

KEY$nnnnnnnn をオプションで定義することができます。このキーワードにより、同じキーワードを使用しているアプリケーションだけが、表示スペースを共用することができるようになります。

注:

1. **Start Keystroke Intercept** (50) 関数は共用できません。一度に 1 つのアプリケーションだけが、キー・ストロークをトラップすることができます。
2. **Connect To Presentation Space** (1) 関数および **Start Keystroke Intercept** (50) 関数は、共通サブシステム関数を共用します。あるアプリケーションによって、これら 2 つの関数のうちのいずれかを共用するという要求が出され、その要求が正常に実行されると、別のアプリケーションがこれら 2 つの関数についての要求を出した場合に影響を受けます。例えば、アプリケーション A が KEY$abcdefgh をキーワードとして Write_Read アクセスで **Connect To Presentation Space** (1) を要求し、その要求が正常に実行された場合、アプリケーション B による **Connect To Presentation Space** (1) または **Start Keystroke Intercept** (50) についての要求は、両方のアプリケーションに互換性のある読み書きオプションを設定している場合にのみ正常に実行されます。

表 5. 前提関数およびそれに依存する関数

前提呼び出し	関数	アクセス
Allocate Communications Buffer (120)	Free Communication Buffer (120)	N/A
Connect Window Service (101)	Change PS Window Name (106) Change Switch List Name (105) Disconnect Window Service (102) Query Window Service (103) Window Status (104)	書き込み 読み取り 照会 = 読み取り 設定 = 書き込み 書き込み
Connect Presentation Space (1)	Copy Field to String (34) Copy OIA (13) Copy Presentation Space (5) Copy Presentation Space to String (8) Copy String to Field (33) Copy String to Presentation Space (15) Disconnect Presentation Space (2) Find Field Length (32) Find Field Position (31) Query Cursor Location (7) Query Field Attribute (14) Release (12) Reserve (11) Search Field (30) Search Presentation Space (6) Send key (3) Set Cursor (40) Start Playing Macro (110) Wait (4)	読み取り 読み取り 読み取り 読み取り 書き込み 書き込み 書き込み 読み取り 読み取り 読み取り 読み取り 書き込み 書き込み 読み取り 読み取り 読み取り 書き込み 書き込み 読み取り
Connect Structured Field (120)	Disconnect Structured Field (121) Get Request Completion (125) Read Structured Field (126) Write Structured Field (127)	N/A

表 5. 前提関数およびそれに依存する関数 (続き)

前提呼び出し	関数	アクセス
Read Structured Field (126)	Get Request Completion (125)	N/A
Start Close Intercept (41)	Query Close Intercept (42) Stop Close Intercept (43)	N/A
Start Host Notification (23)	Query Host Update (24) Stop Host Notification (25)	
Start Keystroke Intercept (50)	Get Key (51) Post Intercept Status (52) Stop Keystroke Intercept (53)	N/A
Write Structured Field (127)	Get Request Completion (125)	N/A

表示スペースのロック

共用表示スペースを指定した場合でも、 **Lock Presentation Space API** (60) 関数または **Lock Windows Services API** (61) 関数を使用すれば、アプリケーションは、表示スペースの排他的制御を得ることができます。これらの関数によってロックされた表示スペースを使用したいという、他のアプリケーションからの要求は、キューに入れられます。ロックをかけた側のアプリケーションが表示スペースをアンロックすると、キューの要求は先入れ先出し法 (FIFO) に基づいて処理されます。

表示スペースをロックしたアプリケーションが、 **Unlock** オプションを指定する同じ呼び出し、または **Reset System** (21) 呼び出しを使用してアンロックしない場合、そのアプリケーションが終了するか、またはセッションが停止するまで、ロックは解除されません。

ASCII 略号

ホストのキーボードで入力するキー・ストロークには、対応する ASCII 値があります。キー・ストロークに対する **Get Key** (51) 関数の応答の仕方は、キーが定義されているか、またはキーが ASCII 値または ASCII の略号として定義されているかどうかによって異なります。

1 つのセッションのキーボードで、別のセッションが必要とする一部のコードを作成することができない場合もあります。この一部のコードを表す ASCII 略号は、**Send Key** (3) 関数のデータ・ストリング・パラメーターに入れることができます。

Send Key (3) 関数および **Get Key** (51) 関数によって、ASCII 値または使用可能なキーでは表すことができないキー・ストロークをセッションで交換することができます。キーボードから生成できる 1 組の略号が提供されます。これらの略号によって、ASCII 文字を使用して、ワークステーションのキーボードの特殊ファンクション・キーを表すことができます。

シフトなしキーの略号は、エスケープ文字とその後に続く省略形からなります。また、これはシフト・キー全体、上段シフト、前面 (Alt)、Ctrl にもあてはまります。シフト付きキーの略号は、シフト・キーの略号とその後に続くシフトなしキーの略号からなります。したがって、シフト付きキーの略号は、エスケープ文字、省略形、エスケープ文字、省略形の 4 文字の順序になります。

デフォルトのエスケープ文字は @ です。エスケープ文字の値は、**Set Session Parameters** (9) 関数の ESC=c オプションを指定して他の任意の文字に変更することもできます。ただし、以下の説明では、デフォルトのエスケープ文字を使用しています。

ASCII 文字セットにないシフト標識は、ホスト・アプリケーションに対して、次のように 2 バイトの ASCII の略号で表されます。

上段シフト	@S
Alt (前面)	@A
Ctrl	@r

これらのシフトの標識に対する略号は、アプリケーションで別々に受信されることはありません。同様に、アプリケーションがシフト標識の略号を別々に送信することもありません。シフト標識の略号には、常にシフトなし標識文字または略号が伴います。

使用する省略形によって、特殊キーの略号が覚えやすくなります。最もよく使用するキーには、英字キー・コードを使用しています。例えば、クリア・キーは *C*、タブ・キーは *T* などです。英字の大文字と小文字は、それぞれ異なるキーの省略形となっていることに注意してください。

以下の説明では、これらの関数の使用方法について説明します。

概要

定義済みのすべてのキーは次のいずれかで表されます。

- 1 バイトの ASCII 値。これは 256 のエレメントの ASCII 文字セットの一部です。
- 2 バイト、4 バイト、または 6 バイトの ASCII の略号。

ASCII 文字として定義したキーを表すには、その文字に対応する 1 バイトの ASCII 値を使用します。

関数として定義したキーを表すには、その関数に対応する 2 バイト、4 バイト、または 6 バイトの ASCII の略号を使用します。例えば、後退タブ・キーを表すには、@B を使用します。 PF1 を表すには、@1 を使用します。入力消去を表すには、@A@F を使用します。次のリストを参照してください。

@B	左タブ	@0	カーソル・ホーム	@h	PF17
@C	消去	@1	PF1/F1	@i	PF18
@D	削除	@2	PF2/F2	@j	PF19
@E	実行 (Enter)	@3	PF3/F3	@k	PF20
@F	EOF 消去	@4	PF4/F4	@l	PF21
@H	ヘルプ (PC400)	@5	PF5/F5	@m	PF22
@I	挿入	@6	PF6/F6	@n	PF23
@J	ジャンプ	@7	PF7/F7	@o	PF24
@L	カーソル左移動	@8	PF8/F8	@q	End
@N	改行	@9	PF9/F9	@u	前ページ (PC400)
@O	スペース	@a	PF10/F10	@v	次ページ (PC400)
@P	ページ印刷	@b	PF11/F11	@x	PA1
@R	リセット	@c	PF12/F12	@y	PA2

@T	右タブ	@d	PF13	@z	PA3
@U	カーソル上移動	@e	PF14	@@	@ (at) 記号
@V	カーソル下移動	@f	PF15	@$	カーソル切り替え
@X	DBCS	@g	PF16	@<	バックスペース
@Z	カーソル右移動				

@A@C	テスト (PC400)	@A@e	ピンク (PC/3270)
@A@D	ワード削除	@A@f	緑 (PC/3270)
@A@E	フィールド終了	@A@g	黄 (PC/3270)
@A@F	入力消去	@A@h	青 (PC/3270)
@A@H	システム要求	@A@i	空色 (PC/3270)
@A@I	挿入切り替え	@A@j	白 (PC/3270)
@A@J	カーソル選択	@A@l	ホスト・カラーのリセット (PC/3270)
@A@L	高速カーソル左移動	@A@t	印刷 (ワークステーション)
@A@Q	アテンション	@A@u	ロールアップ (PC400)
@A@R	装置取り消し (DvCnl)	@A@v	ロールダウン (PC400)
@A@T	表示スペース印刷	@A@y	正方向ワード・タブ
@A@U	高速カーソル上移動	@A@z	逆方向ワード・タブ
@A@V	高速カーソル下移動	@A@-	フィールド - (PC400)
@A@Z	高速カーソル右移動	@A@+	フィールド + (PC400)
@A@9	反転表示	@A@<	レコード・バックスペース (PC400)
@A@b	下線 (PC/3270)	@S@E	ホストでの表示スペース印刷 (PC400)
@A@c	反転表示のリセット (PC/3270)	@S@x	複写 (DUP)
@A@d	赤 (PC/3270)	@S@y	フィールド・マーク

注:

1. 最初の表の最初の @ 記号はエスケープ文字を表します。 2 番目の表の 1 つ目の @ 記号と 2 つ目の @ 記号はエスケープ文字です。 @ 記号はデフォルトのエスケープ文字です。エスケープ文字の値は **Set Session Parameters** (9) 関数の ESC=c オプションを使用して変更することができます。

   エスケープ文字を # に変更すると、後退タブ・キー、Home キー、および入力消去キーを表すために使用するリテラル順序はそれぞれ、#B、#0、および #A#F になります。

   また、@ 記号を表すために使用するリテラル順序は、#@ になります。
2. 印刷画面の略号 (つまり、@P または @A@T) を送信する場合は、呼び出しデータ・ストリングの終わりにこの記号を入れてください。
3. 装置取り消しの略号 (つまり、@A@R) を送信する場合には、その略号はエラー・メッセージを伴わずに渡されます。ただし、ローカル・コピーは停止されません。

Get Key (51) 関数

端末オペレーターが ASCII 文字として定義したキーを入力すると、ホスト・アプリケーションはその文字に対応する 1 バイトの ASCII 値を受信します。

オペレーターが関数として定義したキーをタイプすると、ホスト・アプリケーションはその関数に対応する 2 バイト、4 バイト、または 6 バイトの ASCII の略号を

受信します。例えば、**後退タブ・**キーをタイプすると、 @B を受信します。 **PF1** を押すと、@1 を受信します。**入力消去 (ErInp)** を押すと、@A@F を受信します。

オペレーターが定義済みのシフト・キーの組み合わせをタイプすると、ホスト・アプリケーションは、ASCII 文字、または定義済みの文字や関数に対応する 2 バイト、4 バイト、6 バイトのいずれかの ASCII の略号を受信します。

オペレーターが定義されていない個々のキーをタイプすると、 **Get Key** (51) 関数が戻りコード 20 を戻し、ホスト・アプリケーションには何も送信されません。

Get Key (51) 関数は、ホスト・アプリケーションに送信されたすべての文字と略号の前に 2 つの ASCII 文字を付けます。最初の ASCII 文字は、キー・ストロークを送信する先のホスト表示スペースの PSID です。その他の文字は、ASCII、特殊シフトまたは略号の場合には、*A*、*S*、または *M* となります。 101 ページの『戻りパラメーター』を参照してください。

Send Key (3) 関数

ASCII 文字を別のセッションに送信するには、その文字を **Send Key** (3) 関数のデータ・ストリング・パラメーターに入れてください。

ファンクション・キーを別のセッションに送信するには、その関数の ASCII の略号を **Send Key** (3) 関数のデータ・ストリング・パラメーターに入れてください。

Send Key (3) 関数が認識されない略号をホスト・セッションに送信すると、そのキーを拒否する戻りコードが戻されます。

デバッグ

EHLLAPI アプリケーションをデバッグする際の援助機能として、パーソナル・コミュニケーションズのトレース機能を使用することができます。この機構は、すべての EHLLAPI 呼び出し、パラメーター、戻り値、および戻りコードのログを生成します。トレース機能の使用法の詳細については、「*管理者ガイドおよび解説書*」を参照してください。

簡単な EHLLAPI サンプル・プログラム

次の Windows サンプル・アプリケーションでは、文字ストリング ″Hello World!″ がホスト・セッション 'A' の最初の入力フィールドに入力されます。

```
#include <stdlib.h>
#include <stdio.h>
#include <windows.h>
#include "hapi_c.h"

int main(char **argv, int argc) {
  int HFunc, HLen, HRc;
  char HBuff[1];
  struct HLDConnectPS ConnBuff;
  // Send Key string for HOME+string+ENTER:
  char SendString[] = "@0Hello World!@E";

  HFunc = HA_RESET_SYSTEM;
  HLen  = 0;
  HRc   = 0;
  hllapi(&HFunc, HBuff, &HLen, &HRc);
  if (HRc != HARC_SUCCESS) {
```

```
      printf("Unable to access EHLLAPI.¥n");
      return 1;
    }

    HFunc = HA_CONNECT_PS;
    HLen  = sizeof(ConnBuff);
    HRc   = 0;
    memset(&ConnBuff, 0x00, sizeof(ConnBuff));
    ConnBuff.stps_shortname = 'A';
    hllapi(&HFunc, (char *)&ConnBuff, &HLen, &HRc);
    switch (HRc) {
      case HARC_SUCCESS:
      case HARC_BUSY:
      case HARC_LOCKED: // All these are OK
        break;
      case HARC_INVALID_PS:
        printf("Host session A does not exist.¥n");
        return 1;
      case HARC_UNAVAILABLE:
        printf("Host session A is in use by another EHLLAPI application.¥n");
        return 1;
      case HARC_SYSTEM_ERROR:
        printf("System error connecting to session A.¥n");
        return 1;
      default:
        printf("Error connecting to session A.¥n");
        return 1;
    }

    HFunc = HA_SENDKEY;
    HLen  = strlen(SendString);
    HRc   = 0;
    hllapi(&HFunc, SendString, &HLen, &HRc);
    switch (HRc) {
      case HARC_SUCCESS:
        break;
      case HARC_BUSY:
      case HARC_LOCKED:
        printf("Send failed, host session locked or busy.¥n");
        break;
      default:
        printf("Send failed.¥n");
        break;
    }

    HFunc = HA_DISCONNECT_PS;
    HLen  = 0;
    HRc   = 0;
    hllapi(&HFunc, HBuff, &HLen, &HRc);

    printf("EHLLAPI program ended.¥n");
    return 0;
}
```

このアプリケーションを IBM VisualAge C/C++ for Windows コンパイラーで作成するために、次の MAKEFILE ファイルを使用することができます (ソース・ファイル名を SAMPLE.C と想定しています)。

```
all: sample.exe

hlldir = C:¥PCOMWIN¥SAMPLES
hlllib = C:¥PCOMWIN¥SAMPLES

.SUFFIXES: .C .OBJ

.c.obj:
```

```
    icc.exe /Ti /Gh /Gm /Gd /C /I $(hlldir) /Tc $*.c

sample.exe: sample.obj
    ilink.exe /de /noe $(hlllib)¥pcscal32.lib $**

sample.obj: sample.c
```

このアプリケーションは、次のコマンドで作成することができます。

```
nmake /a all
```

標準および拡張インターフェースの考慮事項

特定のプラットフォーム上の標準と拡張の EHLLAPI インターフェースの間に機能上の相違はありません。しかし、他の面で重要な違いがあります。

- 拡張 EHLLAPI インターフェースは、表示スペース ID (PSID) を 1 バイトから 4 バイトに拡張します。現時点では追加バイトは使用されませんが、今後出されるバージョンの拡張 EHLLAPI との互換性を確実にするために、ご使用のアプリケーションで追加バイトを 2 進ゼロに設定してください。
- EHLLAPI 関数との間で受け渡されるメモリー・バッファー内のデータ・エレメントの位置 (オフセット) が異なります。拡張 EHLLAPI のデータ・エレメントは、ダブルワード境界に位置合わせされます。標準 EHLLAPI のデータ・エレメントは、特定の方法で位置合わせされることはありません。 EHLLAPI アプリケーションをコード化して、オフセット (バイト) 値によってバッファー内のデータを設定したり、あるいは取り出したりすることはできません。代わりに、HAPI_C.H ファイルで提供されるデータ構造を使用して、データ・エレメントを設定したり、取り出したりすることができます。これにより、16 ビットと 32 ビットの両方のプログラムについて、正しい位置からデータを設定したり、取り出したりすることが保証されます。

EHLLAPI データ・バッファーを 2 進ゼロで事前に埋めて、HAPI_C.H で提供されるデータ構造を使用すると、ソース・コードを変更することなく、標準または拡張で稼働するようにアプリケーションをコンパイルすることができます。例えば、次のセクションのコードは、標準 EHLLAPI については機能しますが、拡張 EHLLAPI では機能しません。

```
#include "hapi_c.h"
...
int Func, Len, Rc;
char Buff[18];
char SessType;

Func = HA_QUERY_SESSION_STATUS;   // Function
Len  = 18;                        // Buffer length
Rc   = 0;
Buff[0] = 'A'                     // Session to query
hllapi(&Func, Buff, &Len, &Rc);   // Execute function

SessType = Buff[9];               // Get session type
...
```

上記の例は、拡張 EHLLAPI アプリケーションとしてコンパイルした場合は機能しません。その理由は次のとおりです。

- このアプリケーションは、拡張セッション ID バイトをゼロに設定していません。
- この関数のバッファー長は、18 ではなく、20 です。

- セッション・タイプ標識は、データ・バッファー内のオフセット 9 ではなく、オフセット 12 です。

同じ関数を、標準または拡張オペレーションでコンパイルした場合に、正しく機能するように書いたものを以下に示します。変更された行は、> で示しています。

```
   #include "hapi_c.h"
   ...
   int Func, Len, Rc;
>  struct HLDQuerySessionStatus Buff;
   char SessType;

   Func = HA_QUERY_SESSION_STATUS;   // Function
>  Len  = sizeof(Buff);              // Buffer length
   Rc   = 0;
>  memset(&Buff, 0x00, sizeof(Buff));// Zero buffer
>  Buff.qsst_shortname = 'A';        // Session to query
   hllapi(&Func, (char *)&Buff, &Len, &Rc);   // Execute function

>  SessType = Buff.qsst_sestype;     // Get session type
   ...
```

ホストの自動化のシナリオ

ここで示すサンプル・シナリオは、EHLLAPI を使用して簡素化できる操作について、概念的な情報を提供します。シナリオでは、以下の領域で EHLLAPI のプログラム式操作が実行できる任務を示しています。

- 以下のことを含むホスト・システムの操作
  - 検索関数
  - キー・ストロークの送信
- 以下のことを含む分散処理
  - データの抽出
  - ファイル転送
- インターフェースの統合

シナリオ 1. 検索関数

通常のホスト・システム・トランザクションには、次の 4 つのフェーズがあります。

1. トランザクションを開始する。
2. ホスト・システムの応答を待つ。
3. 予期した応答であるかどうかを確認するため応答を分析する。
4. 応答からデータを抽出し、使用する。

プログラム式操作では、一連の EHLLAPI 関数を使用してこれらの処置をまねることができます。ホスト・システム・トランザクションの正しい開始点を決定した後で、プログラム式操作は **Search Presentation Space** (6) 関数を呼び出して、どのキーワード・メッセージまたはプロンプト・メッセージを画面に表示するかを決めることができます。

次に、プログラム式操作は、**Send Key** (3) 関数を使用して、データをホスト・システムのセッションに入力し、ホスト・システム・トランザクションを実行します。次に、プログラム式操作は、以下のことを実行できます。

- **Wait** (4) 関数を使用して、X CLOCK、X []、または X SYSTEM 条件が終了するのを待ちます (あるいは、端末がロックされている場合はキーボード・ロック条件を戻します)。

  キーボードの使用が禁止されている場合、EHLLAPI プログラムは **Copy OIA** (13) 関数を呼び出して、エラー条件についての詳細情報を取得することができます。
- **Search Presentation Space** (6) 関数を使用して、適切な応答を受け取ったかどうかをチェックするために、予期したキーワードを探します。
- **Copy Presentation Space to String** (8) 関数 (または、複数のデータ・アクセス関数のうちのいずれか) を使用して、希望するデータを抽出します。

Search Presentation Space (6) 関数は、端末オペレーターの別のタスクをシミュレートするための重要な関数です。一部のホスト・システムは、応答するまでの間、X CLOCK、X [] または X SYSTEM モードではロック状態にならず、代わりに、すぐにキーボードをアンロックして、オペレーターが他の要求をスタックできるようにします。このような環境では、端末オペレーターは、他の何らかの表示されるプロンプト (画面のタイトルまたはラベルなど) によってデータが戻ったことを知ります。 **Search Presentation Space** (6) 関数により、EHLLAPI プログラムは待機中に表示スペースを検索することができます。また、応答待ちの間に **Pause** (18) 関数を呼び出すと、他の DOS セッションが中央演算処理装置のリソースを共用できるようになります。 **Pause** (18) 関数にはオプションがあり、それによって、EHLLAPI プログラムは、ホスト・システムの更新イベントが生ずるのを待つことができます。

妥当なタイムアウト期間を過ぎてもホスト・システムのイベントが発生しない場合、EHLLAPI プログラムは、カスタマイズされた次のようなエラー・メッセージを呼び出すことができます。

```
No Response From Host.  Retry?
```

このような環境では、表示メッセージのわずかな変更に対してもプログラム式操作を再プログラムする必要があるので、プログラムの改訂は重要な考慮事項になります。

例えば、端末オペレーターが

```
Enter Part Number:
```

というメッセージをプロンプトとして期待している場合に、次のメッセージを作り出すようなアプリケーションの変更があっても、その端末オペレーターは正しく対処できます。

```
Enter Component Number:
```

しかし、プログラム式操作はリテラル・キーワード・ストリングを探すので、メッセージ構文の微妙な変更 (大文字と小文字の変更のような単純なもの) であっても、プログラムは事前にプログラムされたエラー処置をとることになります。

シナリオ 2. キー・ストロークの送信

キー・ストロークをホスト・システムに送信するプログラムを組む際に、注意を要する考慮事項がいくつかあります。アプリケーションの環境によっては、コマンド

の発行は、ストリングをキー入力し Enter (改行) キーを押すのと同じように簡単です。また、複数のフィールドのうちの 1 つにデータを入力できる、もっと複雑な定様式画面を必要とするアプリケーションもあります。このような環境では、表示画面に記入する必要があるキー・ストロークを理解している必要があります。

タブ・キーの略号 (@T; 略号の詳細なリストについては、 21 ページの『概要』) を使用して、フィールド間をスキップすることができます。**Send Key** (3) 関数を使用して、キー・ストロークをフィールドに送信する場合、フィールドの長さと内容を知っている必要があります。フィールドを完全に埋め、次の属性バイトを自動スキップにすると、カーソルは次のフィールドに移動します。次にタブを発行した場合は、さらに別のフィールドにスキップします。

同様に、キー・ストロークでフィールドを完全に埋められなかった場合、以前に入力したデータが残ることがあります。「フィールドの終わりまで消去」(Erase EOF) コマンドを使用して、この残ったデータを消去してください。

シナリオ 3. 分散処理

collaborative と呼ばれる区分に入るアプリケーションもあります。このようなアプリケーションは、単一エンド・ユーザー・インターフェースを提供しますが、その処理は、2 つ以上の異なる物理位置で実行されます。

EHLLAPI のアプリケーションは、ホスト・システムと端末ユーザー間の通信を代行受信することによって、ホスト・システムのアプリケーションと対話することができます。ホスト・システムの表示スペースは、このデータを代行受信するための手段となります。ローカルのアプリケーションは、表示スペースが更新されるたびに、またはオペレーターが AID キーを押したときにはいつでも、通知を受けるように要求することができます。

このワークステーションのアプリケーションは、次のいずれかの方法でホスト・システムのアプリケーションと連携することができます。

- フィールドまたは表示スペース・ベースでは、フィールドをアドレスするコピー関数 (**Copy String to Field** (33) 機能または **Copy Field to String** (34) 機能)、もしくは表示スペースからまたは表示スペースへコピーする機能 (例えば、**Copy String to Presentation Space** (15) 機能または **Copy Presentation Space to String** (8) 機能) のいずれかを使用します。
- キー・ストローク・ベースでは、**Send Key** (3) 関数を使用する。
- 大きいブロック・データに対するファイル・ベースでは、アプリケーションで EHLLAPI ファイル転送関数 (**Send File** (90) 関数または **Receive File** (91) 関数) を使用して、データまたは関数 (ロード・モジュールなど) を転送し、ローカルまたはリモートで処理する。

シナリオ 4. ファイル転送

このシナリオでは、次のように、ファイルの転送を自動化したい場合を想定します。

- 前述の検索シナリオで説明した手順を使用し、ホスト・システムのセッションへログオンして作業を開始する。

- いずれかのコピー関数 (多数の画面のデータをコピーするには非効率) を使用する代わりに、EHLLAPI プログラムでファイル転送関数の **Send File** (90) および **Receive File** (91) を呼び出してデータを転送する。
- 正常に終了した時、
  - **Send File** (90) 関数が実行を終了した場合は、EHLLAPI プログラムは、コピー関数または **Send Key** (3) 関数を使用して、バッチ・ジョブを発行してからログオフする。
  - **Receive File** (91) 関数が実行を終了した場合は、 EHLLAPI プログラムはローカルのアプリケーションを開始する。

シナリオ 5. 自動化

あるアプリケーションが別のアプリケーションにすべてのキー・ストロークを提供することができます。また、ターゲットの宛先にキー・ストロークをキーボードから加えることができます。これを行うために、場合によっては、アプリケーションが宛先アプリケーションまたは表示スペース向けキー・ストローク入力のその他のソースを (**Reserve** (11) 関数を使用して) ロックアウトしてから、 (**Release** (12) 関数を使用して) アンロックしなければならないことがあります。

任意のアプリケーションに提供されるキー・ストロークの発信元は、アプリケーションの設計によって決まります。キー・ストロークの発信元を以下に示します。

- キーボード
- ソース・アプリケーションに組み込まれたデータ
- DOS インターフェースによって検索された 2 次記憶データ
- パーソナル・コミュニケーションズ・インターフェース

どの場合も、宛先アプリケーションに提供されるキー・ストロークは、通常のオペレーターの入力と区別がつきません。

シナリオ 6. キー・ストロークのフィルター操作

フィルターとして機能するアプリケーションは、EHLLAPI から (キーボードまたはソース・アプリケーションから) 送信された、別の宛先アプリケーションにあてたキー・ストロークを代行受信します。このキー・ストロークは、次のように処理することができます。

- 無視する (つまり、削除する)。
- 別のアプリケーションにリダイレクトする。
- 妥当性を検査する。
- 変換する (例えば、大文字から小文字へ)。
- 拡張する (キーボード・マクロによって)。

30 ページの図 1 は、キーボード拡張環境でのキー・ストロークのフローとオブジェクトの概略を表しています。

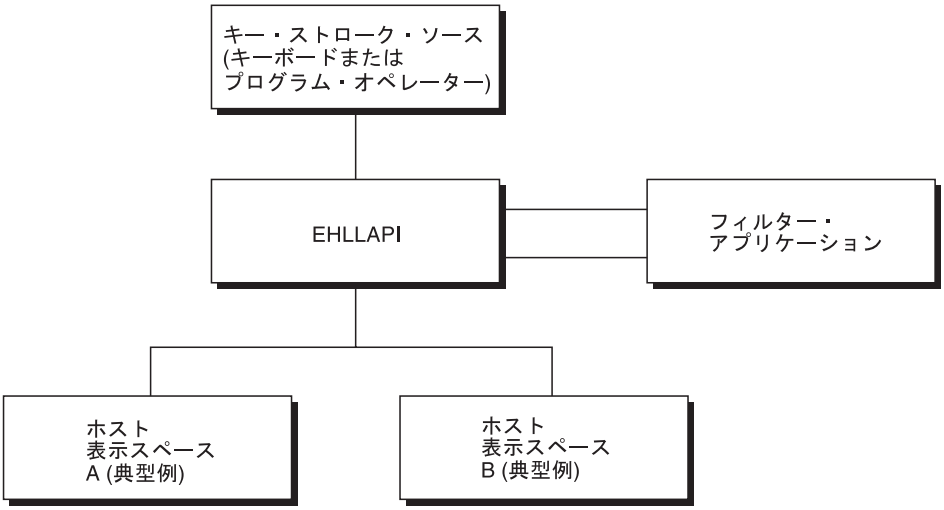

図 1. キー・ストロークのフロー

シナリオ 7. キーボードの拡張

このシナリオでは、**拡張アプリケーション・プログラム**を作成するためにフィルターを使用します。拡張アプリケーション・プログラムとは、キーボードから入力したデータをモニターし、ある特定の方法でそのデータを変更するプログラムです。通常、この種のアプリケーション・プログラムは**キーボード・マクロ**と呼ばれる命令を使用します。この命令は、どのキー・ストロークを探し、どんな変更を加えるかをアプリケーション・プログラムに伝えます。この変更には、キー・ストロークを抑制すること (これによって、宛先アプリケーションではキー・ストロークが送信されなかったように見える)、あるキー・ストロークを別のキー・ストロークに置き換えること、あるいは単一のキー・ストロークを一連のキー・ストロークに置き換えることが含まれます。

EHLLAPI を使用してこの処理を行うために、次のシナリオを作成することができます。

1. EHLLAPI アプリケーション・プログラムで **Connect Presentation Space** (1) 関数を呼び出し、そのキー・ストロークを、フィルターにかける表示スペースに接続する。
2. EHLLAPI プログラムは、次に L オプションを指定して **Start Keystroke Intercept** (50) 関数を呼び出す。これによって、すべてのキー・ストロークが、フィルター操作を行うアプリケーション・プログラムにルーティングされる。
3. フィルター操作を行うアプリケーション・プログラムで、以下のことを行うループを定義できる。
   a. **Get Key** (51) 関数が、宛先表示スペースへ送信中のキー・ストロークのすべてを代行受信する。
   b. フィルター操作を行うアプリケーションはそれぞれのキー・ストロークを調べ、次のようなキーボード・マクロのタスクを実行する。
      - プログラム・コマンドの省略化。この結果、3 つまたは 4 つのキー・ストローク・コマンドを 1 つのキー・ストロークに凝縮することができる。

- コマンドのカスタマイズ。この結果、コマンドが覚えやすくなり、また他のソフトウェア・パッケージとの一貫性を保つことができる。
- 契約書または頻繁に使用される書面の**定形文面**の作成。
- 異なる関数に同じキーを使用する同時実行アプリケーションのためのキーボードの再編成。

例えば、フィルター操作を行うアプリケーションは、 Alt+Y のようなキーの組み合わせを、カーソルを表示スペースの 2 行目の 35 桁に移動し、「テキサス州ダラス XYZ Tool Corporation」というストリングを書き込むコマンドに変換することができます。

c. キー・ストロークが受け入れられない場合、EHLLAPI プログラムは **Post Intercept Status** (52) 関数を使用してビープ音を鳴らす。

4. EHLLAPI プログラムがフィルター・ループを出た後、 **Stop Keystroke Intercept** (53) 関数を呼び出してフィルター操作を終了する。

第 3 章 EHLLAPI 関数

この章では、パーソナル・コミュニケーションズの EHLLAPI の個々の関数の詳細および EHLLAPI プログラム・サンプラーの使用方法について説明します。関数は、関数名によりアルファベット順に並べられています。標準と拡張の両方のインターフェースについて、これらの関数を説明します。

注: 本章では、WinHLLAPI、IBM 標準 32 ビット HLLAPI および 16 ビット EHLLAPI を標準インターフェースと呼び、IBM 拡張 32 ビット EHLLAPI を拡張インターフェースと呼びます。

コード・ページ 1390/1399 および 1137 のためのユニコード・サポート

次の EHLLAPI 関数は、ユニコード・セッションの日本語コード・ページ 1390/1399 とヒンディ語コード・ページ 1137 サポートで使用できます。

- Convert Position または Convert RowCol (1137 のみ)
- Copy Field to String
- Copy Presentation Space
- Copy Presentation Space to String
- Copy String to Field
- Copy String to Presentation Space
- Get Key
- Search Field
- Search Presentation Space
- Send Key
- Set Cursor (1137 のみ)
- Set Session Parameters

日本語コード・ページ 1390/1399 とヒンディ語コード・ページ 1137 の詳細については、それぞれの関数ごとに特定のセクションを参照してください。

注:

1. PCOMM セッションに送信するユニコード文字を含むストリングは、コード・ページ 1390/1399 では `WCHAR *`、コード・ページ 1137 では `char *` の型キャストであるようにしてください。
2. EHLLAPI 1390/1399 ユニコード機能は 3270 および 5250 セッションでのみ選択可能です。EHLLAPI 1137 ユニコード機能は 5250 セッションでのみ選択可能です。

ページ・レイアウトについての規則

すべての EHLLAPI 関数呼び出しは、必要な情報を素早く検索できるように同一の形式で表されています。形式は次のとおりです。

関数名 (関数番号)

前提呼び出し

呼び出しパラメーター

戻りパラメーター

使用上の注意

前提呼び出し

「前提呼び出し」では、その関数に先行して呼び出しておく必要がある関数をリストしています。

呼び出しパラメーター

「呼び出しパラメーター」には、その EHLLAPI 関数を呼び出すためにプログラム内で定義しなければならないパラメーターをリストし、それらのパラメーターを定義する方法を説明しています。関数にパラメーターがない場合は、 *NA* (不適用) と記されます。 **Set Session Parameters** (9) 関数呼び出しで定義されたセッション・パラメーターの値でパラメーターをオーバーライドできる場合は、そのセッション・パラメーター名が示されています。

戻りパラメーター

「戻りパラメーター」には、その EHLLAPI 関数を呼び出した後にプログラムが受け取るはずのパラメーターをリストし、それらのパラメーターをどのように解釈するかを説明します。

使用上の注意

「使用上の注意」には、その関数に影響を与えるセッション・オプションをリストします。また、関数の使用に関する技術情報とアプリケーション開発のヒントも提供します。

EHLLAPI 関数の要約

表 6 は EHLLAPI 関数を要約したものです。

表 6. EHLLAPI 関数の要約

関数	3270	5250	VT
43 ページの『Connect Presentation Space (1)』	YES	YES	YES
94 ページの『Disconnect Presentation Space (2)』	YES	YES	YES
153 ページの『Send Key (3)』	YES	YES	YES
190 ページの『Wait (4)』	YES	YES	YES
66 ページの『Copy Presentation Space (5)』	YES	YES	YES

表 6. EHLLAPI 関数の要約 (続き)

関数	3270	5250	VT
146 ページの『Search Presentation Space (6)』	YES	YES	YES
120 ページの『Query Cursor Location (7)』	YES	YES	YES
74 ページの『Copy Presentation Space to String (8)』	YES	YES	YES
165 ページの『Set Session Parameters (9)』	YES	YES	YES
126 ページの『Query Sessions (10)』	YES	YES	YES
139 ページの『Reserve (11)』	YES	YES	YES
138 ページの『Release (12)』	YES	YES	YES
57 ページの『Copy OIA (13)』	YES	YES	YES
121 ページの『Query Field Attribute (14)』	YES	YES	YES
88 ページの『Copy String to Presentation Space (15)』	YES	YES	YES
113 ページの『Pause (18)』	YES	YES	YES
128 ページの『Query System (20)』	YES	YES	YES
140 ページの『Reset System (21)』	YES	YES	YES
124 ページの『Query Session Status (22)』	YES	YES	YES
180 ページの『Start Host Notification (23)』	YES	YES	YES
123 ページの『Query Host Update (24)』	YES	YES	YES
188 ページの『Stop Host Notification (25)』	YES	YES	YES
141 ページの『Search Field (30)』	YES	YES	YES
97 ページの『Find Field Position (31)』	YES	YES	YES
96 ページの『Find Field Length (32)』	YES	YES	YES
83 ページの『Copy String to Field (33)』	YES	YES	YES
48 ページの『Copy Field to String (34)』	YES	YES	YES
164 ページの『Set Cursor (40)』	YES	YES	YES
176 ページの『Start Close Intercept (41)』	YES	YES	YES
116 ページの『Query Close Intercept (42)』	YES	YES	YES
186 ページの『Stop Close Intercept (43)』	YES	YES	YES
115 ページの『Query Additional Field Attribute (45)』	NO	YES	NO
183 ページの『Start Keystroke Intercept (50)』	YES	YES	YES
100 ページの『Get Key (51)』	YES	YES	YES
114 ページの『Post Intercept Status (52)』	YES	YES	YES
189 ページの『Stop Keystroke Intercept (53)』	YES	YES	YES
109 ページの『Lock Presentation Space API (60)』	YES	NO	NO
111 ページの『Lock Window Services API (61)』	YES	NO	NO
178 ページの『Start Communication Notification (80)』	YES	YES	YES
119 ページの『Query Communication Event (81)』	YES	YES	YES
187 ページの『Stop Communication Notification (82)』	YES	YES	YES
206 ページの『Send File (90)』	YES	YES	NO
136 ページの『Receive File (91)』	YES	YES	NO
37 ページの『Cancel File Transfer (92)』	YES	YES	YES

表 6. EHLLAPI 関数の要約 (続き)

関数	3270	5250	VT
46 ページの『Convert Position または Convert RowCol (99)』	YES	YES	YES
44 ページの『Connect Window Services (101)』	YES	YES	YES
95 ページの『Disconnect Window Service (102)』	YES	YES	YES
129 ページの『Query Window Coordinates (103)』	YES	YES	YES
191 ページの『Window Status (104)』	YES	YES	YES
40 ページの『Change Switch List LT Name (105)』	YES	YES	YES
38 ページの『Change PS Window Name (106)』	YES	YES	YES
186 ページの『Start Playing Macro (110)』	YES	YES	YES
41 ページの『Connect for Structured Fields (120)』	YES	NO	NO
93 ページの『Disconnect from Structured Fields (121)』	YES	NO	NO
117 ページの『Query Communications Buffer Size (122)』	YES	NO	NO
『Allocate Communications Buffer (123)』	YES	NO	NO
99 ページの『Free Communications Buffer (124)』	YES	NO	NO
106 ページの『Get Request Completion (125)』	YES	NO	NO
131 ページの『Read Structured Fields (126)』	YES	NO	NO
195 ページの『Write Structured Fields (127)』	YES	NO	NO

Allocate Communications Buffer (123)

3270	*5250*	*VT*
YES	NO	NO

Allocate Communications Buffer 関数を使用して、オペレーティング・システムからバッファーを取得することができます。バッファー・アドレスは、 **Read Structured Fields** (126) と **Write Structured Fields** (127) の両方の関数に渡す必要があります。

前提呼び出し

この関数には前提呼び出しはありません。

呼び出しパラメーター

	標準インターフェース	拡張インターフェース
関数番号	必ず 123 にしてください。	

	標準インターフェース	拡張インターフェース
データ・ストリング	次の表を参照してください。	
長さ	必ず 6 にしてください。	必ず 8 にしてください。
PS 位置	NA	

呼び出しデータ・ストリングには次のものがあります。

バイト位置		定義
標準	拡張	
1 ～ 2	1 ～ 4	32 ビットまたは 16 ビットのバッファー長。 (0 < size ≤ (64 KB-256 bytes)=X'FF00')
3 ～ 6	5 ～ 8	割り振られるバッファーの 32 ビット・アドレス (戻される)。

戻りパラメーター

戻りコード	説明
0	**Allocate Communications Buffer** 関数は正常終了しました。
2	パラメーター指定でエラーが発生しました。
9	システム・エラーが発生しました。
11	リソースが使用できません。(メモリーが使用できません。)

使用上の注意

1. EHLLAPI は、オペレーティング・システムのメモリー・マネージメントからバッファーを取得し、そのバッファー・アドレスを戻りパラメーター・ストリングに置きます。要求バッファー・サイズ (長さ) もパラメーター・ストリングに渡されます。バッファー・サイズは、1 バイトから 64 KB - 256 バイト (X'FF00' バイト) の長さです。

   バッファー・サイズに関する情報は、『**Query Communications Buffer Size** (122)』を参照してください。
2. この関数で取得したバッファーは、別のプロセス間では共用できません。共用しようとすると、アプリケーションで予測不可能な結果が生じることがあります。
3. EHLLAPI アプリケーションは、**Free Communications Buffer** (124) 関数を発行して、割り振られたメモリーを解放する必要があります。
4. 1 つのアプリケーションに割り振ることのできるバッファーは 10 個が限度です。この限度に達すると、リソースが使用できないことを示す戻りコード (RC=11) が戻されます。
5. **Reset System** (21) 関数は、この関数が割り振ったバッファーを解放します。

Cancel File Transfer (92)

3270	*5250*	*VT*
YES	YES	YES

Cancel File Transfer 関数を使用すると、現行の EHLLAPI が指定されたセッションに対して開始した、 **Send File** または **Receive File** が直ちに戻されます。

前提呼び出し

Send File (90) または **Receive File** (91)

呼び出しパラメーター

	拡張インターフェース
関数番号	必ず 92 にしてください。
データ・ストリング	ホスト表示スペースの 1 文字の短縮名。ホスト接続表示スペースへの更新要求を示すブランクまたは NULL。
長さ	4 が暗黙指定されます。
PS 位置	NA

呼び出しデータ構造には、以下の要素が含まれます。

バイト位置	**定義**
1	表示スペースの 1 文字の短縮名 (PSID)
2 ～ 4	予約済み

戻りパラメーター

戻りコード	**定義**
0	関数は正常終了しました。
1	誤った PSID が指定されました。
8	この PSID に対して、先に **Start Communication Notification** (80) 関数が呼び出されていません。
9	システム・エラーが発生しました。

使用上の注意

Send File (90) および **Receive File** (91) の両方がブロッキング呼び出ししているため、この関数は、常に、異なるスレッド上で発行されなければなりません。

Change PS Window Name (106)

3270	*5250*	*VT*
YES	YES	YES

Change PS Window Name 関数を使用すると、アプリケーションは表示スペース・ウィンドウに新しい名前を指定したり、デフォルト名にリセットすることができます。

前提呼び出し

Connect Window Services (101)

呼び出しパラメーター

	標準インターフェース	拡張インターフェース
関数番号	必ず 106 にしてください。	
データ・ストリング	次の表を参照してください。	
長さ	必ず指定してください (注を参照)。	必ず 68 にしてください。
PS 位置	NA	

注: データ・ストリング長は必ず指定してください (通常、PC/3270 では 3 ～ 63、PC400 では 4 ～ 63、拡張インターフェースでは 68)。

呼び出しデータ・ストリングには次のものがあります。

バイト位置		定義
標準	拡張	
1	1	表示スペースの 1 文字の短縮名 (PSID)
	2 ～ 4	予約済み
2	5	変更要求オプションの値。次のうちの 1 つを選択してください。 • 表示スペース・ウィンドウ名を変更する時は、X'01' • 表示スペース・ウィンドウ名をリセットする時は、X'02'
3 ～ 63	6 ～ 66	終止符バイトを含む 1 (PC/3270 の場合) または 2 (PC400 の場合) から 61 バイトの ASCII ストリング。 ASCII ストリングは必ず NULL 文字で終了してください。このストリングには、NULL 文字の前に少なくとも 1 個の非 NULL 文字が必要です。
	67 ～ 68	予約済み

戻りパラメーター

戻りコード	説明
0	**Change PS Window Name** 関数は正常終了しました。
1	無効なホスト表示スペースの短縮セッション ID が指定されたか、またはホスト表示スペースが接続されていません。
2	パラメーター指定でエラーが発生しました。
9	システム・エラーが発生しました。
12	セッションが停止しました。

使用上の注意

ストリングは、最初に NULL 文字が検出されたところで終了します。 NULL 文字は、指定されたストリングの長さを変更します。 NULL 文字が指定した長さの終わりにない場合、指定した長さのところで最後のバイトが NULL 文字に置き換えら

れ、データ・ストリングの残りの部分は失われます。指定した長さより短いところで NULL 文字が検出された場合、ストリングはそこで切り捨てられ、データ・ストリングの残りの部分は失われます。

アプリケーションが終了する前に表示スペース名のリセットに失敗すると、出口リスト処理がその表示スペース名をリセットします。

Change Switch List LT Name (105)

3270	*5250*	*VT*
YES	YES	YES

Change Switch List LT Name 関数を使用すると、アプリケーションは選択した論理端末 (LT) に対するスイッチ・リストの変更またはリセットができます。アプリケーションは呼び出し時に、スイッチ・リストに挿入する名前を指定する必要があります。

注: この関数は コミュニケーション・マネージャー EHLLAPI との互換性を保つためのもので、**Change PS Window Name** (106) 関数と同じ結果になります。

前提呼び出し

Connect Window Services (101)

呼び出しパラメーター

	標準インターフェース	**拡張インターフェース**
関数番号	必ず 105 にしてください。	
データ・ストリング	次の表を参照してください。	
長さ	通常は 4 〜 63 です。	必ず 68 にしてください。
PS 位置	NA	

呼び出しデータ・ストリングには次のものがあります。

バイト位置		**定義**
標準	拡張	
1	1	表示スペースの 1 文字の短縮名 (PSID)
	2 〜 4	予約済み
2	5	変更要求オプション。次の中から選びます。 • スイッチ・リストの LT 名を変更する時は、X'01' • スイッチ・リストの LT 名をリセットする時は、X'02'
3 〜 63	6 〜 66	終止符バイトを含む 2 から 61 バイトの ASCII ストリング。 ASCII ストリングは必ず NULL 文字で終了してください。このストリングには、NULL 文字の前に少なくとも 1 個の非 NULL 文字が必要です。
	67 〜 68	予約済み

戻りパラメーター

戻りコード	説明
0	**Change Switch List LT Name** 関数は正常終了しました。
1	無効なホスト表示スペースの短縮セッション ID が指定されたか、またはホスト表示スペースが接続されていません。
2	パラメーター指定でエラーが発生しました。
9	システム・エラーが発生しました。
12	セッションが停止しました。

使用上の注意

ストリングは、最初に NULL 文字が検出されたところで終了します。 NULL 文字は、指定されたストリングの長さを変更します。 NULL 文字が指定した長さの終わりにない場合、指定した長さのところで最後のバイトが NULL 文字に置き換えられ、データ・ストリングの残りの部分は失われます。指定した長さより短いところで NULL 文字が検出された場合、ストリングはそこで切り捨てられ、データ・ストリングの残りの部分は失われます。

アプリケーションが終了する前にスイッチ・リスト LT 名のリセットに失敗すると、出口リスト処理がそのスイッチ・リスト LT 名をリセットします。

Connect for Structured Fields (120)

3270	*5250*	*VT*
YES	NO	NO

Connect for Structured Fields 関数を使用すると、アプリケーションは、エミュレーション・プログラムへの接続を確立し、ホスト・アプリケーションとの間で構造化フィールド・データを交換することができます。ワークステーションのアプリケーションは、Query Reply (照会・応答) データ・フィールドを提供し、パラメーター・ストリング内でそれを指していなければなりません。エミュレーターから戻された Destination/Origin ID はアプリケーションに戻されます。

前提呼び出し

この関数には前提呼び出しはありません。

呼び出しパラメーター

	標準インターフェース	拡張インターフェース
関数番号	必ず 120 にしてください。	
データ・ストリング	次の表を参照してください。	
長さ	7 または 11 にしてください。	必ず 16 にしてください。
PS 位置	NA	

呼び出しデータ・ストリングには次のものがあります。

バイト位置		定義
標準	拡張	
1	1	表示スペースの 1 文字の短縮名 (PSID)
	2 ～ 4	予約済み
2 ～ 5	5 ～ 8	Query Reply (照会・応答) データ・バッファーのアドレス
6 ～ 7	9 ～ 10	戻される Destination/Origin 固有の ID(16 ビット・ワード)
	11 ～ 12	予約済み
8 ～ 11	13 ～ 16	この位置にあるデータは EHLLAPI に無視されます。ただし、移行プログラムでこの位置にデータを指定していても、エラーは発生しません。このデータはアプリケーションの移行時に互換性を保つために設けられています。

戻りパラメーター

戻りコード	説明
0	**Connect for Structured Fields** 関数は正常終了しました。
1	指定されたホスト表示スペースの短縮 セッション ID が無効か、またはホスト表示スペースが接続されていません。
2	パラメーター指定でエラーが発生しました。
9	システム・エラーが発生しました。
10	エミュレーション・プログラムはこの関数をサポートしていません。
32	アプリケーションは通信用にこのセッションにすでに接続しています。(正常接続)
39	1 つの DDM セッションがこのセッションにすでに接続しています。

使用上の注意

1. EHLLAPI は Destination/Origin ID (DOID) 自己定義パラメーター (SDP) の Query Reply (照会・応答) バッファーを走査して、Query Reply の DOID フィールドの内容を決定します。この値が X'0000' の場合は、エミュレーターが DOID をアプリケーションに割り当て、EHLLAPI は割り当てられた ID で Query Reply の DOID フィールドを埋めます。アプリケーションが Query Reply の DOID フィールドに指定した値がゼロ以外の値である場合は、ID は前もって割り当てられていないものと見なされ、エミュレーターは指定された値をアプリケーションの DOID として割り当てます。指定された DOID がすでに使用中の場合には、EHLLAPI は戻りコード 2 を戻します。
2. アプリケーションは、Query Reply データ構造をアプリケーションの専用メモリー内に作成する必要があります。 EHLLAPI がサポートする Query Reply データ構造の形式と使用法についての詳細は、367 ページの『付録 A. EHLLAPI がサポートする Query Reply データ構造』を参照してください。
3. Query Reply のデータについて簡単なチェックが実行されます。 ID と構造の長さについてのみ有効かどうかチェックされます。

4. ホスト・セッションごとに接続できる DDM 基本タイプは 1 つだけです。DDM 接続が Destination/Origin ID (DOID) 用に自己定義パラメーター (SDP) をサポートしている場合は、複数接続できます。
5. 戻りコード RC=32 または RC=39 を受信した場合、アプリケーションは選択したセッションにすでに接続しており、表示スペースを注意して使用する必要があります。 SRPI、ファイル転送、および他の EHLLAPI アプリケーションとの競合が生じる可能性があります。

Connect Presentation Space (1)

3270	*5250*	*VT*
YES	YES	YES

Connect Presentation Space 関数は、EHLLAPI アプリケーション・プログラムとホスト表示スペースとの接続を確立します。

前提呼び出し

この関数には前提呼び出しはありません。

呼び出しパラメーター

	標準インターフェース	**拡張インターフェース**
関数番号	必ず 1 にしてください。	
データ・ストリング	ホスト表示スペースの 1 文字の短縮名。	
長さ	1 が暗黙指定されます。	必ず 4 にしてください。
PS 位置	NA	

呼び出しデータ・ストリングには次のものがあります。

バイト位置		**定義**
標準	拡張	
1	1	表示スペースの 1 文字の短縮名 (PSID)
	2 ～ 4	予約済み

戻りパラメーター

Connect Presentation Space 関数は戻りコードを設定して、試行状況、および試行が正常終了した場合はホスト表示スペースの状況を示します。

戻りコード	**説明**
0	**Connect Presentation Space** 関数は正常終了しました。ホスト表示スペースはアンロックされ、入力の準備ができました。
1	無効なホスト表示スペース ID が指定されました。指定されたセッションが存在しないか、または、論理プリンター・セッションです。さらにこの戻りコードは、DDE/EHLLAPI 用の API Setting がオンに設定されていないことを示している場合があります。
4	接続には成功しましたが、ホスト表示スペースが使用中です。

戻りコード	説明
5	接続には成功しましたが、ホスト表示スペースがロックされています (入力禁止)。
9	システム・エラーが発生しました。
11	このリソースは使用できません。ホスト表示スペースは、すでに他のシステム関数が使用しています。

使用上の注意

1. **Connect Presentation Space** 関数は、 CONLOG/CONPHYS セッション・オプションの影響を受けます。
2. 1 つの EHLLAPI アプリケーションが同時に複数の表示スペースに接続することはできません。前提関数として **Connect Presentation Space** 関数を必要とする呼び出しは、現在接続されている表示スペースを使用します。例えば、アプリケーションが表示スペースに A、B、C の順で接続した場合、そのアプリケーションが関数を発行するためには B または A に再度接続しなければなりません。
3. **Connect Presentation Space** を必要とする各スレッドに対応する **Disconnect Presentation Space** (2) があるか、またはそれらのスレッドの 1 つが **Reset System** (21) を発行する必要があります。この機能は全スレッドに影響し、残りの接続をすべて切断します。
4. アプリケーションが共用をサポートし (つまり、それらのアプリケーションが一緒に動作できるように開発されており、予測可能な動作が示されている)、なおかつ **Set Session Parameters** (9) 関数で設定したものと互換性がある読み取り/書き込みアクセス権とキーワードのオプションがある場合、複数の EHLLAPI アプリケーションで 1 つの表示スペースを共用できます。詳細については、 165 ページの『Set Session Parameters (9)』を参照してください。
5. **Connect Presentation Space** 関数と **Start Keystroke Intercept** (50) 関数は共通のサブシステム関数を共用します。このため、あるアプリケーションによって、1 つのセッションに対してこれら 2 つの関数のうちのいずれかを共用するという要求が出され、その要求が正常に実行されると、別のアプリケーションがこれら 2 つの関数についての要求を出した場合に影響を受ける場合があります。例えば、アプリケーション A が セッションに対して、KEY$abcdefgh をキーワードとして Write アクセスで指定した **Connect Presentation Space** を要求し、その要求が正常に実行された場合、アプリケーション B による、あるセッションに対する **Connect Presentation Space** および **Start Keystroke Intercept** についての要求は、両方のアプリケーションで互換性のある読み取り/書き込みオプションが設定されている場合にのみ、正常に実行されます。
6. 論理プリンター・セッションとして定義したセッションには接続できません。詳細については、「*管理者ガイドおよび解説書*」を参照してください。

Connect Window Services (101)

3270	*5250*	*VT*
YES	YES	YES

Connect Window Services 関数を使用すると、アプリケーションは表示スペース・ウィンドウを管理することができます。ウィンドウ・サービスでは、1 つの表示スペースには一度に 1 つの EHLLAPI アプリケーションしか接続できません。

ウィンドウ・サービスでは、1 つの EHLLAPI アプリケーションは同時に複数の表示スペースに接続できます。

前提呼び出し

この関数には前提呼び出しはありません。

呼び出しパラメーター

	標準インターフェース	拡張インターフェース
関数番号	必ず 101 にしてください。	
データ・ストリング	ホスト表示スペースの 1 文字の短縮セッション ID。	
長さ	1 が暗黙指定されます。	必ず 4 にしてください。
PS 位置	NA	

呼び出しデータ・ストリングには次のものがあります。

バイト位置		定義
標準	拡張	
1	1	表示スペースの 1 文字の短縮名 (PSID)
	2 ～ 4	予約済み

戻りパラメーター

戻りコード	説明
0	**Connect Window Services** 関数は正常終了しました。
1	無効なホスト表示スペースの短縮セッション ID が指定されたか、または Sessions Window Services マネージャーが接続されていません。さらにこの戻りコードは、DDE/EHLLAPI 用の API Setting がオンに設定されていないことを示している場合があります。
9	システム・エラーが発生しました。
10	エミュレーション・プログラムはこの関数をサポートしていません。
11	このリソースは使用できません。ホスト表示スペースは、すでに他のシステム関数が使用しています。

使用上の注意

1. 1 つの EHLLAPI アプリケーションは同時に複数の表示スペース・ウィンドウに接続できます。アプリケーションは、接続された複数の表示スペース・ウィンドウを切断しなくても、それらの間で前後に移動することができます。例えば、アプリケーションが表示スペース・ウィンドウ A、B、および C に 接続している場合、同時に A、B、および C すべてにアクセス可能となり、他のアプリケーションからは A、B、C のどれにもアクセスできません。

2. プロセスを実行するには、**Connect Window Services** 関数だけで十分です。ただし、**Connect Window Services** を必要とする各スレッドに対応する **Disconnect Window Services** (102) があるか、またはそれらのスレッドの 1 つが **Reset System** (21) を発行する必要があります。この機能は全スレッドに影響し、残りの接続をすべて切断します。

Convert Position または Convert RowCol (99)

3270	*5250*	*VT*
YES	YES	YES

Convert Position または **Convert RowCol** 関数は、ホスト表示スペースの位置の値をディスプレイの行と桁の座標に変換したり、ディスプレイの行と桁の座標をホスト表示スペースの位置の値に変換します。この関数は、カーソル位置を変更しません。

前提呼び出し

この関数には前提呼び出しはありません。

呼び出しパラメーター

	標準インターフェース	**拡張インターフェース**
関数番号	必ず 99 にしてください。	
データ・ストリング	**Convert Position** 関数に対してはホスト表示スペースの短縮名および P (例えば、AP はセッション A の表示スペース位置を変換します)、あるいは、**Convert RowCol** 関数に対してはホスト表示スペースの短縮名および R (例えば、AR はセッション A の行と桁の座標を変換します)。	
長さ	データ・ストリング・パラメーターの 2 番目の文字として R を指定する場合は行 (Row)。有効な入力の下限値は 1、上限値はホスト表示スペースの構成によって異なります。 48 ページの『使用上の注意』を参照してください。 データ・ストリング・パラメーターの 2 番目の文字として P を指定する場合は適用されません。	
PS 位置	データ・ストリング・パラメーターの 2 番目の文字として R を指定する場合は桁 (Column)。有効な入力の下限値は 1、上限値はホスト表示スペースの構成によって異なり、24 から 43 の間です。 48 ページの『使用上の注意』を参照してください。 データ・ストリング・パラメーターの 2 番目の文字として P を指定する場合は、ホスト表示スペース位置。有効な入力の下限値は 1、上限値はホスト表示スペースの構成によって異なり、1920 から 3564 の間です。 48 ページの『使用上の注意』を参照してください。	

呼び出しデータ・ストリングには次のものがあります。

バイト位置		定義
標準	拡張	
1	1	表示スペースの 1 文字の短縮名 (PSID)
	2 ～ 4	予約済み
2	5	**変換**オプション P または R
	6 ～ 8	予約済み

戻りパラメーター

この関数はデータ・ストリング長および戻りコードを戻します。

データ・ストリング長:

Convert Position 関数 (呼び出しデータ・ストリング内の 2 番目の文字が P) については、1 から 43 (PC/3270 の場合) または 27 (PC400 の場合) が戻されます。この値は、呼び出し PS 位置パラメーターに含まれた PS 位置を含む行の番号です。上限値は、ホスト表示スペースの構成によっては、43 (PC/3270 の場合) または 27 (PC400 の場合) よりも小さくなります。

Convert RowCol 関数 (呼び出しデータ・ストリングの 2 番目の文字が R) の場合、0 という値は、行についての入力値 (呼び出しデータ・ストリング長パラメーター) にエラーがあることを示します。

戻りコード:

Convert Position または RowCol 関数は、4 番目の戻りパラメーターに常に戻りコードがあるという規則から除外されています。この関数に関しては、4 番目のパラメーターに戻される値は状況コードです。この状況コードには、データまたは戻りコードが含まれることがあります。予測不能な結果やエラーを避けるため、アプリケーションはこの状況コードの処理機能を備えている必要があります。

- 4 番目のパラメーターの値が、0、9998、9999 のいずれかである場合は、戻りコードです。
- **Convert Position** 関数 (呼び出しデータ・ストリング内の 2 番目の文字が P) の場合、1 から 132 の間の値は、呼び出し PS 位置パラメーターで渡された PS 位置を含む桁の番号です。上限値は、ホスト表示スペースの構成によっては、132 よりも小さくなります。
- **Convert RowCol** 関数 (呼び出しデータ・ストリング内の 2 番目の文字が R) の場合、1 から 3564 の間の値は、呼び出しデータ・ストリング長パラメーターと PS 位置パラメーターでそれぞれ渡された行と桁の値に対応するホスト表示スペースの位置を表します。上限値は、ホスト表示スペースの構成によっては、3564 よりも小さくなります。

以下の状況コードが定義されています。

状況コード	説明
0	これは無効な PS 位置または桁です。
>0	これは PS 位置または桁です。

状況コード	説明
9998	無効なホスト表示スペース ID が指定されたか、またはシステム・エラーが発生しました。
9999	データ・ストリング内の 2 番目の文字が P でも R でもありません。

使用上の注意

1. 表示スペースを構成する場合は、「*管理者ガイドおよび解説書*」を参照してください。
2. 表示スペースの行や桁の数を知るには、 **Query Session Status** (22) 関数で戻されるデータ・ストリング・パラメーターを調べてください。 124 ページの『Query Session Status (22)』を参照してください。

1137 コード・ページ・サポート

ユニコード機能は 5250 セッションでのみサポートされます。

Convert Position または Convert RowCol は、クラスターの先頭に戻るためにヒンディ語が使用できます。 **Convert Position または Convert RowCol** の使用法は SBCS セッションと同じです。

Copy Field to String (34)

3270	*5250*	*VT*
YES	YES	YES

Copy Field to String 関数は、ホスト接続の表示スペース内のフィールドからストリングへ文字を転送します。

Copy Field to String 関数は、ホスト・ソース表示スペース内の文字を情報交換用日本工業規格コード (ASCII) に変換します。属性バイトやその他の通常 ASCII では表されない文字は、ブランクに変換されます。

前提呼び出し

Connect Presentation Space (1)

呼び出しパラメーター

	標準インターフェース	拡張インターフェース
関数番号	必ず 34 にしてください。	

	標準インターフェース	拡張インターフェース
データ・ストリング	事前に割り振りされたターゲットのデータ・ストリング。Extended Attribute Bytes (EAB) オプションを指定して **Set Session Parameters** (9) 関数を発行した場合、データ・ストリングの長さは少なくともフィールドの長さの 2 倍必要です。 **DBCS の場合のみ**: Extended Attributes Double-byte (EAD) オプションを指定した場合、データ・ストリングの長さは最低でもフィールド長の 3 倍なければなりません。EAB と EAD の両オプションを指定した場合、データ・ストリングの長さは最低でもフィールド長の 4 倍なければなりません。	
長さ	コピーするバイトの数 (データ・ストリングの長さ)	
PS 位置	該当フィールドを識別します。これはコピー先フィールド内のどのバイトの PS 位置でも構いません。コピー関数は常にフィールドの先頭から開始されます。	

戻りパラメーター

この関数は、データ・ストリング、データ・ストリング長、戻りコードを戻します。

データ・ストリング:

ストリングはホスト表示スペース内の識別されたフィールドからのデータを含んでいます。戻されるデータ・ストリングの先頭バイトは、ホスト表示スペース内の識別されたフィールドの先頭バイトです。戻されるデータ・ストリングのバイト数は、次のうちの小さい方によって決まります。

- 呼び出しデータ・ストリング長パラメーターで指定したバイト数。
- ホスト表示スペース内の識別されたフィールドのバイト数。

データ・ストリング長:

戻されるデータの長さ。

戻りコード:

以下のコードが定義されています。

戻りコード	説明
0	**Copy Field to String** 関数は正常終了しました。
1	ユーザー・プログラムが現在ホスト・セッションに接続されていません。
2	パラメーター指定でエラーが発生しました。
6	コピーするデータとコピー先フィールドのサイズが等しくありません。ストリングの長さがコピーされるフィールドよりも短い場合、データは切り捨てられます。
7	ホスト表示スペースの位置が正しくありません。
9	システム・エラーが発生しました。
24	ホスト表示スペースが定様式化されていません。

使用上の注意

1. フィールドの位置および長さに関する情報は、**Find Field Position** (31) 関数および **Find Field Length** (32) 関数を使用して得ることができます。**Copy Field to String** 関数は保護フィールド、無保護フィールドのいずれでも使用できますが、*フィールド形式* のホスト表示スペース内のフィールドに限定されています。
2. 次のいずれかの状態が発生した時、コピーは終了します。
   - フィールドの終わりに到達。
   - コピー先ストリングの長さを超過。
3. **DBCS の場合のみ:** コピー先ストリングが 2 バイト文字の上位バイトで終わる場合、そのバイトはブランクに変換されます。EAD オプションを ON に設定した場合、1 文字につき 3 バイトずつ戻されます。EAB と EAD の両オプションを ON に設定した場合は、1 文字につき 4 バイトずつ戻されます。

   注: フィールドが表示スペースの終わりで折り返している場合、表示スペースの終わりに到達したときに折り返します。
4. **DBCS の場合のみ: Set Session Parameters** (9) 関数の EAD オプションをこの関数と共に使用して、2 バイトの EAD を戻すことができます。EAB オプションを指定せずに EAD オプションを指定した場合、各文字の前に EAD が戻されます。EAB と EAD の両オプションを指定した場合、EAB の前に EAD が戻されます。
5. EAB を戻すには、**Set Session Parameters** (9) 関数の EAB オプションを使用します。EAB は表示スペース上の各文字と関連しており、それぞれの文字の前に戻されます。
6. **Copy Field to String** 関数は、ATTRB/NOATTRB/NULLATTRB、 EAB/NOEAB、XLATE/NOXLATE、 DISPLAY/NODISPLAY、DISPLAY/NODISPLAY、 EAD/NOEAD (DBCS 用のみ)、および NOSO/SPACESO/SO (DBCS 用のみ) のセッション・オプションによる影響を受けます。詳細については、 5 (168 ページ)、 13、 14 (171 ページ)、17 (172 ページ)、および 20、 21 (173 ページ) を参照してください。

   前述のように、さまざまな **Copy** (5、8、および 34) 関数により戻される属性は、**Set Session Parameters** (9) 関数の影響を受けます。関係するセッション設定パラメーターは、以下のような影響を及ぼします。

   セッション設定パラメーター
   COPY 機能への影響

   NOEAB、NOEAD
   属性は戻されません。テキストのみを表示スペースからユーザーのバッファーへコピーします。

   EAB、NOXLATE
   属性は以下の表に定義されたとおりに戻されます。

   EAB、XLATE
   表示スペースの表示に使用される色が戻されます。色は再マップできるので、XLATE と EAB が同時に ON になっている場合、属性の色は **COPY** 関数によって戻される色ではありません。

   EAD 以下の表に示される 2 バイト文字セット属性が戻されます。

戻される文字の属性を以下の表に示します。属性ビット位置は IBM 形式 (つまり、バイトの左端がビット 0) で示してあります。

3270 の文字属性が、ホストからエミュレーターに戻されます。次の表は、EAB および NOXLATE が設定されている場合に適用されます。

ビット位置	意味
0 ～ 1	文字の強調表示 00 = 標準 01 = 明滅 10 = 反転表示 11 = 下線
2 ～ 4	文字の色 (カラー再マップでこの色定義を一時変更できます。) 000 = デフォルト値 001 = 青 010 = 赤 011 = ピンク 100 = 緑 101 = 空色 110 = 黄 111 = 白
5 ～ 6	文字属性 00 = デフォルト値 11 = 2 バイト文字
7	予約済み

5250 の文字属性が、ホストからエミュレーターに戻されます。次の表は、EAB および NOXLATE が設定されている場合に適用されます。

ビット位置	意味
0	反転イメージ 0 = 標準イメージ 1 = 反転イメージ
1	下線 0 = 下線なし 1 = 下線付き
2	明滅 0 = 明滅しない 1 = 明滅する
3	桁分離文字線 0 = 桁分離文字線なし 1 = 桁分離文字線付き
4 ～ 7	予約済み

次の表は、パーソナル・コミュニケーションズの文字カラー属性を示します。次の表は、EAB および XLATE が設定されている場合に適用されます。

ビット位置	意味
0 ～ 3	背景文字の色 0000 = 黒 0001 = 青 0010 = 緑 0011 = シアン 0100 = 赤 0101 = マジェンタ 0110 = 茶 (3270)、黄 (5250) 0111 = 白
4 ～ 7	前景文字の色 0000 = 黒 0001 = 青 0010 = 緑 0011 = シアン 0100 = 赤 0101 = マジェンタ 0110 = 茶 (3270)、黄 (5250) 0111 = 白 1000 = 灰色 1001 = 明るい青 1010 = 明るい緑 1011 = 明るいシアン 1100 = 明るい赤 1101 = 明るいマジェンタ 1110 = 黄 1111 = 白 (高輝度)

- 2 バイト文字セット属性 (DBCS の場合のみ)
  - 第 1 バイト

ビット位置	文字位置	フィールド属性位置
0	2 バイト文字	予約済み
1	2 バイト文字の第 1 バイト	予約済み
2	SO	予約済み

ビット位置	文字位置	フィールド属性位置
3 〜 4	SI (ビット位置 3)	5250 DBCS 関連フィールド ビット位置 7 が 0 のとき 00 = デフォルト値 01 = DBCS のみ 10 = DBCS と SBCS のどちらか一方 11 = DBCS と SBCS が混在 ビット位置 7 が 1 のとき 00 = 予約済み 01 = SO/SI なしの DBCS のみ 10 = 予約済み 11 = 予約済み
5	予約済み	SO/SI 生成可能 (3270 のみ)
6	予約済み	文字属性が存在 (3270 のみ)
7	予約済み	5250 DBCS 関連拡張フィールド 0 = 基本 2 バイト・フィールド 1 = 拡張 2 バイト・フィールド

– 第 2 バイト

ビット位置	文字位置	フィールド属性位置
0	予約済み	左罫線 (3270 のみ)
1	予約済み	上罫線 (3270 のみ)
2	予約済み	右罫線 (3270 のみ)
3	予約済み	下罫線 (3270 のみ)
4	左罫線	左罫線
5	上罫線	上罫線
6 〜 7	予約済み	予約済み

PS/2® モノクローム表示の場合、アプリケーション (ワークステーション) セッション内の文字はグレー表示されます。この機能は、EHLLAPI アプリケーション・セッション内の再マップ済みの色をユーザーに提示して、ホストのアプリケーションの表示スペースに表示されているものを確認できるようにするために必要です。

7. この関数を使用するには、戻りデータ・ストリング・パラメーターを受け入れるメモリーを事前に割り振ります。このメモリーの事前割り振りに必要なステート

メントは、アプリケーションに使用される言語によって異なります。詳細については、 10 ページの『メモリー割り振り』を参照してください。

注: 5250 エミュレーションは、24 行 80 桁の表示スペースをサポートします。コミュニケーション・マネージャー 5250 エミュレーションが 25 番目の行を表示するインスタンスもあります。このようなことが起こるのは、ホストからのエラー・メッセージが表示される場合、またはオペレーターが SysReq キーを選択する場合です。パーソナル・コミュニケーションズは、25 行目の情報をステータス・バーに表示します。**EXTEND_PS** オプションを指定すると、EHLLAPI アプリケーションでコミュニケーション・マネージャー EHLLAPI と同じインターフェースを使用することができ、このような状況が発生した時に有効な表示スペースが拡張されます。

1390/1399 コード・ページ・サポート

ユニコード機能は 3270 および 5250 セッションでのみサポートされます。

ユニコード・セッションでは、ホストのコピー元の表示スペース内の文字をユニコードに変換します。属性バイトは通常ブランクに変換されます。

ユニコード・セッションでは、 XLATE オプション (**Set Session Parameters** 関数を使用して指定できる) はサポートされていません。これは、もしこのオプションが発行されても、EAB は PC カラー・グラフィックス・アダプター (CGA) 形式に変換されないことを意味します。

前提呼び出し: Connect Presentation Space (1)

呼び出しパラメーター:

	標準インターフェース	**拡張インターフェース**
関数番号	必ず 34 にしてください。	
データ・ストリング	事前に割り振りされたターゲットのデータ・ストリング。拡張属性バイト (EAB) オプションを指定して **Set Session Parameters** (9) 関数を発行した場合、データ・ストリングの長さは少なくとも EBCDIC フィールドの長さの 2 倍必要です。	
長さ	ユニコード文字でのコピー先データ・ストリングの長さ。	
PS 位置	該当フィールドを識別します。これはコピー先フィールド内のどのバイトの PS 位置でも構いません。コピー関数は常にフィールドの先頭から開始されます。	

戻りパラメーター: この関数は、データ・ストリング、データ・ストリング長、戻りコードを戻します。

データ・ストリング:
ユニコード・データを含むストリングが戻されます。

データ・ストリング長:
ストリングにコピーされたユニコード文字の数。

戻りコード:
以下のコードが定義されています。

戻りコード	説明
0	**Copy Field to String** 関数は正常終了しました。
1	ユーザー・プログラムが現在ホスト・セッションに接続されていません。
2	パラメーター指定でエラーが発生しました。
6	コピーするデータとコピー先フィールドのサイズが等しくありません。ストリングの長さがコピーされるフィールドよりも短い場合、データは切り捨てられます。
7	ホスト表示スペースの位置が正しくありません。
9	システム・エラーが発生しました。
24	ホスト表示スペースが定様式化されていません。

使用上の注意: 以下のオプションは **Copy Field To String** (34) のユニコード・セッションでサポートされ、DBCS の場合と同じ方法で機能します。

- NOATTRB
- ATTRB
- NULLATTRB
- EAB
- NOEAB
- NOXLATE
- DISPLAY
- NODISPLAY

1137 コード・ページ・サポート

ユニコード機能は 5250 セッションでのみサポートされます。

ユニコード・セッションでは、ホストのコピー元の表示スペース内の文字をユニコードに変換します。属性バイトは通常ブランクに変換されます。

ユニコード・セッションでは、 XLATE オプション (**Set Session Parameters** 関数を使用して指定できる) はサポートされていません。これは、もしこのオプションが発行されても、EAB は PC カラー・グラフィックス・アダプター (CGA) 形式に変換されないことを意味します。

前提呼び出し: Connect Presentation Space (1)

呼び出しパラメーター:

	標準インターフェース	拡張インターフェース
関数番号	必ず 34 にしてください。	
データ・ストリング	事前に割り振りされたターゲットのデータ・ストリング。長さは、表示スペースからコピーされる時に必要な EBCDIC バイト数の 2 倍必要です。拡張属性バイト (EAB) オプションを指定して **Set Session Parameters** (9) 関数を発行した場合、データ・ストリングの長さは少なくとも EBCDIC フィールドの長さの 4 倍必要です。	

	標準インターフェース	拡張インターフェース
長さ	コピー先データ・ストリングのバイト単位の長さ。この長さは、ユニコード・セッションでは最低 2 バイトなければなりません。そうでない場合は、2 のエラー・コードが戻されます。	
PS 位置	該当フィールドを識別します。これはコピー先フィールド内のどのバイトの PS 位置でも構いません。コピー関数は常にフィールドの先頭から開始されます。	

戻りパラメーター: この関数は、データ・ストリング、データ・ストリング長、戻りコードを戻します。

データ・ストリング:
ユニコード・データを含むストリングが戻されます。

データ・ストリング長:
ストリングにコピーされたユニコード文字の数。バイト数を知るには、2 を掛けます。

戻りコード:
以下のコードが定義されています。

戻りコード	説明
0	**Copy Field to String** 関数は正常終了しました。
1	ユーザー・プログラムが現在ホスト・セッションに接続されていません。
2	パラメーター指定でエラーが発生しました。
6	コピーするデータとコピー先フィールドのサイズが等しくありません。ストリングの長さがコピーされるフィールドよりも短い場合、データは切り捨てられます。
7	ホスト表示スペースの位置が正しくありません。
9	システム・エラーが発生しました。
24	ホスト表示スペースが定様式化されていません。

使用上の注意: 以下のオプションは **Copy Field To String** のユニコード・セッションでサポートされ、SBCS の場合と同じ方法で機能します。

- NOATTRB
- ATTRB
- NULLATTRB
- EAB
- NOEAB
- NOXLATE
- DISPLAY
- NODISPLAY

Copy OIA (13)

3270	*5250*	*VT*
YES	YES	YES

Copy OIA 関数は、ホスト接続表示スペースから現在のオペレーター情報域 (OIA) データを戻します。

OIA は画面の下分割線の下側にあり、ワークステーションとホストとの接続に関するセッション状況情報の表示に使用されます。

前提呼び出し

Connect Presentation Space (1)

呼び出しパラメーター

	標準インターフェース	**拡張インターフェース**
関数番号	必ず 13 にしてください。	
データ・ストリング	事前に割り振りされたターゲットのデータ・ストリング	
長さ	103	104
PS 位置	NA	

戻りパラメーター

この関数はデータ・ストリングおよび戻りコードを戻します。

データ・ストリング:

16 ビットの場合は 103 バイトのストリング、32 ビットの場合は 104 バイトのストリング。詳細については、58 ページの『戻り OIA データ・ストリングの形式』を参照してください。

戻りコード:

以下のコードが定義されています。

戻りコード	**説明**
0	OIA データが戻されます。コピー先の表示スペースはロックされていません。
1	ユーザー・プログラムが現在ホスト・セッションに接続されていません。
2	指定されたストリング長に誤りがありました。 OIA データは戻されませんでした。
4	OIA データが戻されます。コピー先の表示スペースは使用中です。
5	OIA データが戻されます。コピー先の表示スペースはロックされています。 (入力禁止)
9	内部システム・エラーが発生しました。 OIA データは戻されませんでした。

使用上の注意

1. OIA グループは、接続したセッションの状況を表すビットで構成されています。グループは、そのグループが意味するホストの関数によって分けられています。(例えば、グループ 8 にはそのセッションで起こり得るすべての入力禁止条件を表すビットが入っています。) 各グループの状態は、高位のビットがより高い優先順位を表すような順序になっています。つまり、ビット 7 はビット 0 より優先されます。そのため、1 つのグループ内で複数の状態が活動状態にあるときは、優先順位の最も高い状態が、そのグループ内の活動状態です。
2. この関数を使用するには、戻りデータ・ストリング・パラメーターを受け入れるメモリーを事前に割り振ります。このメモリーの事前割り振りに必要なステートメントは、アプリケーションに使用される言語によって異なります。詳細については、10 ページの『メモリー割り振り』を参照してください。

戻り OIA データ・ストリングの形式

OIA データ・ストリングには以下の情報が入っています。

バイト位置		定義
標準	拡張	
1	1	OIA フォーマット設定バイト。値は 1 (PC/3270)、9 (PC400)、または 5 (VT) です。
2 ～ 81	2 ～ 81	ホスト・コード・ポイントの OIA イメージ。
82 ～ 103	82 ～ 103	OIA グループの標識の意味。
	104	予約済み。

PC/3270 OIA グループの標識の意味とそのイメージ: OIA イメージ・グループは、ホスト・コード・ポイント内の OIA イメージを含む属性バイトのない 80 バイトの ASCII 文字ストリングから構成されます。 59 ページの図 2 は、ホスト表示スペースで検出される 16 進コードと、それらを表す文字を示しています。戻されたデータは OIA グラフィックス文字に変換されます。 OIA 標識については、「はじめに」を参照してください。

戻されたデータを OIA グラフィックス文字に変換する場合は次のようにします。

1. バイト 2 から 81 に戻されたデータを、画面またはプリンターで表示または印刷します。
2. 出力が表示される装置に適用できるコード・ページ・チャートを使用して、それぞれの文字に対応する 16 進数値を見つけます。
3. 59 ページの図 2 を使用して、ステップ 2 で見つけた 16 進数値のそれぞれに対応する OIA グラフィックス文字を見つけます。

注: グループ 8 (バイト 0) 機械チェック、通信チェック、およびプログラム・チェックのイメージの後には、チェックのタイプに関連する 3 桁の数字があります。

オンラインおよび表示画面の所有権グループのイメージは非 SNA 3274 コントローラー構成のためのものです。SNA の場合、16 進数値 CD は CD (59 ページの図 2 を参照) に変換されます。3174 コントローラーまたは SDLC 接続での実行の場合、16 進数値 X'F4' は X'B2' または X'22' に置き換えられます。強調表示標識

は、“グループ 5 (オフセット 86 強調表示グループ 1” のバイト) に対応する (データ・ストリングの先頭 80 バイトの中にある) イメージです。強調表示標識の後には、X'F9' (明滅)、X'FC' (下線)、 X'D2' (反転表示)、または X'80' (ホスト・デフォルト値) のいずれかが続きます。

後に X'20' が続く短縮セッション ID は 7 桁目にあります。

すべてのグループ・イメージは、メインフレーム・インタラクティブ (MFI) の 16 進コード・ポイントで表されます。

注: OIA イメージ・データ・ストリング位置から 1 を引いたものが、OIA 桁に等しくなります。

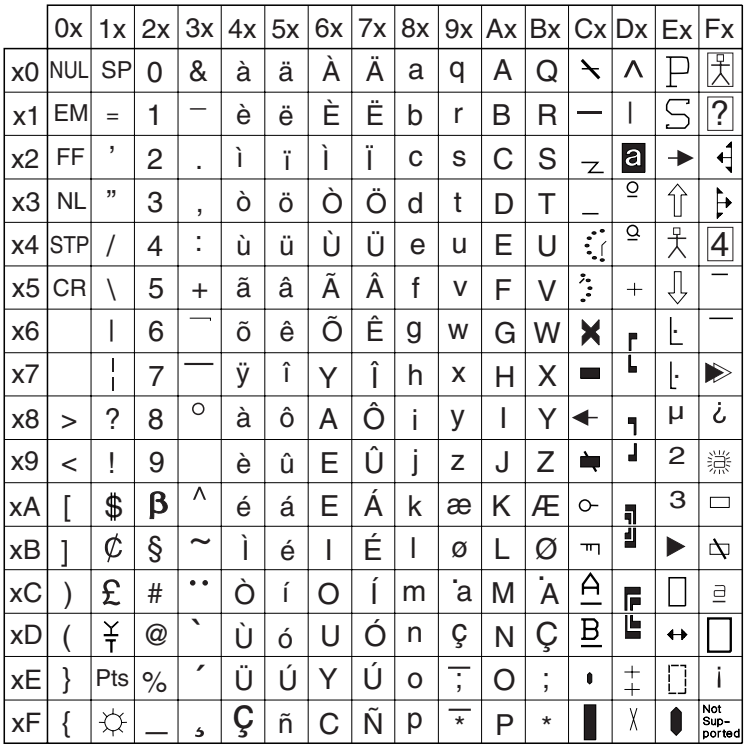

図 2. ホスト表示スペース文字

- グループ 1 (オフセット 82): オンラインおよび画面の所有権

ビット	意味
0 ～ 1	予約済み
2	SSCP-LU セッションが表示画面を所有する
3	LU-LU セッションが表示画面を所有する
4	オンラインであり、所有されていない
5	サブシステム作動可能
6 ～ 7	予約済み

- グループ 2 (オフセット 83): 文字選択

ビット	意味
0	予約済み
1	APL
2	カタカナ (日本のみ)
3	英数字
4 ～ 5	予約済み
6	ひらがな (日本のみ)
7	2 バイト文字

- グループ 3 (オフセット 84): シフト状態

ビット	意味
0	上段シフト
1	数字
2	CAPS
3 ～ 7	予約済み

- グループ 4 (オフセット 85): PSS グループ 1

ビット	意味
0 ～ 7	予約済み

- グループ 5 (オフセット 86): 強調表示グループ 1

ビット	意味
0	オペレーターが選択可能
1	フィールド継承
2 ～ 7	予約済み

- グループ 6 (オフセット 87): 色グループ 1

ビット	意味
0	オペレーターが選択可能
1	フィールド継承
2 ～ 7	予約済み

- グループ 7 (オフセット 88): 挿入

ビット	意味
0	挿入モード
1 ～ 7	予約済み

- グループ 8 (オフセット 89 ～ 93): 入力禁止 (5 バイト)

– バイト 1 (オフセット 89)

ビット	意味
0	リセット不可能なマシン・チェック
1	予約済み
2	マシン・チェック
3	通信チェック
4	プログラム・チェック
5 ～ 7	予約済み

– バイト 2 (オフセット 90)

ビット	意味
0	装置使用中
1	端末待ち
2	マイナス記号
3	マイナス関数
4	入力過多
5 ～ 7	予約済み

– バイト 3 (オフセット 91)

ビット	意味
0 ～ 2	予約済み
3	間違ったデッド・キーの組み合わせ、限定キー
4	不適当な位置
5 ～ 7	予約済み

– バイト 4 (オフセット 92)

ビット	意味
0 ～ 1	予約済み
2	システム待ち
3 ～ 7	予約済み

– バイト 5 (オフセット 93)

ビット	意味
0 ～ 7	予約済み

• グループ 9 (オフセット 94): PSS グループ 2

ビット	意味
0 ～ 7	予約済み

- グループ 10 (オフセット 95): 強調表示グループ 2

ビット	意味
0 〜 7	予約済み

- グループ 11 (オフセット 96): 色グループ 2

ビット	意味
0 〜 7	予約済み

- グループ 12 (オフセット 97): 通信エラー状況メッセージ

ビット	意味
0 〜 6	通信エラー
1 〜 7	予約済み

- グループ 13 (オフセット 98): プリンターの状態

ビット	意味
0 〜 7	予約済み

- グループ 14 (オフセット 99): グラフィックス

ビット	意味
0 〜 7	予約済み

- グループ 15 (オフセット 100): これは予約済みグループです。
- グループ 16 (オフセット 101): 自動キーの再生/記録の状態

ビット	意味
0 〜 7	予約済み

- グループ 17 (オフセット 102): 自動キーの打ち切り/停止の状態

ビット	意味
0 〜 7	予約済み

- グループ 18 (オフセット 103): 拡大状態

ビット	意味
0 〜 7	予約済み

PC400 OIA グループの標識の意味とそのイメージ: 以下の表に、OIA グループの詳細をリストします。

- グループ 1 (オフセット 82): オンラインおよび画面の所有権

ビット	意味	データ・ストリングの開始位置
0 〜 2	予約済み	

ビット	意味	データ・ストリングの開始位置
3	システム使用可能	1
4	予約済み	
5	サブシステム作動可能	
6 ～ 7	予約済み	

- グループ 2 (オフセット 83): 文字選択

ビット	意味	データ・ストリングの開始位置
0 ～ 1	予約済み	
2	カタカナ (日本のみ)	
3	英数字	
4 ～ 5	予約済み	
6	ひらがな (日本のみ)	
7	2 バイト文字	

- グループ 3 (オフセット 84): シフト状態

ビット	意味	データ・ストリングの開始位置
0	予約済み	
1	キーボード・シフト	39
2	CAPS	
3 ～ 6	予約済み	
7	2 バイト文字入力可能	

- グループ 4 (オフセット 85): PSS グループ 1

ビット	意味	データ・ストリングの開始位置
0 ～ 7	予約済み	

- グループ 5 (オフセット 86): 強調表示グループ 1

ビット	意味	データ・ストリングの開始位置
0 ～ 7	予約済み	

- グループ 6 (オフセット 87): 色グループ 1

ビット	意味	データ・ストリングの開始位置
0 ～ 7	予約済み	

- グループ 7 (オフセット 88): 挿入

ビット	意味	データ・ストリングの開始位置
0	挿入モード	68
1 ～ 7	予約済み	

- グループ 8 (オフセット 89 〜 93): 入力禁止 (5 バイト)
  - バイト 1 (オフセット 89)

ビット	意味	データ・ストリングの開始位置
0 〜 7	予約済み	

  - バイト 2 (オフセット 90)

ビット	意味	データ・ストリングの開始位置
0 〜 7	予約済み	

  - バイト 3 (オフセット 91)

ビット	意味	データ・ストリングの開始位置
0 〜 4	予約済み	
5	オペレーターの入力エラー	64
6 〜 7	予約済み	

  - バイト 4 (オフセット 92)

ビット	意味	データ・ストリングの開始位置
0 〜 1	予約済み	
2	システム待ち	64
3 〜 7	予約済み	

  - バイト 5 (オフセット 93)

ビット	意味	データ・ストリングの開始位置
0 〜 7	予約済み	

- グループ 9 (オフセット 94): PSS グループ 2

ビット	意味	データ・ストリングの開始位置
0 〜 7	予約済み	

- グループ 10 (オフセット 95): 強調表示グループ 2

ビット	意味	データ・ストリングの開始位置
0 〜 7	予約済み	

- グループ 11 (オフセット 96): 色グループ 2

ビット	意味	データ・ストリングの開始位置
0 〜 7	予約済み	

- グループ 12 (オフセット 97): 通信エラー状況メッセージ

ビット	意味	データ・ストリングの開始位置
0	通信エラー	
1 ～ 5	予約済み	
7	メッセージ待ち	3

- グループ 13 (オフセット 98): プリンターの状態

ビット	意味	データ・ストリングの開始位置
0 ～ 7	予約済み	

- グループ 14 (オフセット 99): グラフィックス

ビット	意味	データ・ストリングの開始位置
0 ～ 7	予約済み	

- グループ 15 (オフセット 100): これは予約済みグループです。
- グループ 16 (オフセット 101): 自動キーの再生/記録の状態

ビット	意味	データ・ストリングの開始位置
0 ～ 7	予約済み	

- グループ 17 (オフセット 102): 自動キーの打ち切り/停止の状態

ビット	意味	データ・ストリングの開始位置
0 ～ 7	予約済み	

- グループ 18 (オフセット 103): 拡大状態

ビット	意味	データ・ストリングの開始位置
0 ～ 7	予約済み	

VT ホスト OIA グループの標識の意味とそのイメージ: 以下の表に、VT ホスト OIA グループの詳細をリストします。

- グループ 1 (オフセット 82): オンラインおよび画面の所有権

ビット	意味
5	サブシステム作動可能

- グループ 2 (オフセット 83): 文字選択

ビット	意味
0	上段シフト
2	CAPS

- グループ 7 (オフセット 88): 挿入

ビット	意味
0	挿入モード

OIA 行表示の一部の桁には、3270/5250 用に表示されるメッセージとは異なる VT 用のメッセージが表示されます。個々の詳細については次の表を参照してください。

桁	シンボル
1 ～ 7	VT220 7
	VT220 8
	VT100
	VT52
	VTANSI
9 ～ 12	LOCK
61 ～ 64	HOLD

Copy Presentation Space (5)

3270	*5250*	*VT*
YES	YES	YES

Copy Presentation Space 関数は、ホスト接続した表示スペースの内容を EHLLAPI アプリケーション・プログラム内に定義されたデータ・ストリングにコピーします。

Copy Presentation Space 関数は、ホストのコピー元の表示スペース内の文字を ASCII 形式に変換します。属性バイトやその他の通常 ASCII では表されない文字は、ブランクに変換されます。属性バイトをブランクに変換したくない場合には、**Set Session Parameters** (9) 関数の ATTRB オプションを指定して、この変換を一時変更することができます。

前提呼び出し

Connect Presentation Space (1)

呼び出しパラメーター

	標準インターフェース	拡張インターフェース
関数番号	必ず 5 にしてください。	

	標準インターフェース	拡張インターフェース
データ・ストリング	事前にホスト表示スペースのサイズを割り振られたコピー先ストリング。このサイズはホスト表示スペースの構成によって違います。 EAB オプションを指定して **Set Session Parameters** (9) 関数を発行した場合、データ・ストリングの長さは少なくとも表示スペースの長さの 2 倍必要です。 **DBCS の場合のみ:** EAD オプションを指定した場合、データ・ストリングの長さは少なくとも表示スペースの長さの 3 倍必要です。 EAB と EAD の両オプションを指定した場合、データ・ストリングの長さは最低でも表示スペースの長さの 4 倍なければなりません。	
長さ	NA (ホスト表示スペース長が暗黙指定されます)。	
PS 位置	NA	

戻りパラメーター

この関数は、データ・ストリング、データ・ストリング長、戻りコードを戻します。

データ・ストリング:
接続されたホスト表示スペースの内容。

データ・ストリング長:
コピーされたデータの長さ。

戻りコード:
以下のコードが定義されています。

戻りコード	説明
0	ホスト表示スペースの内容はアプリケーション・プログラムにコピーされました。コピー先表示スペースは活動状態にあり、キーボードはアンロックされています。
1	ユーザー・プログラムが現在ホスト・セッションに接続されていません。
4	ホスト表示スペースの内容はコピーされました。接続されたホスト表示スペースはホストからの応答を待っています。
5	ホスト表示スペースはコピーされました。キーボードはロックされています。
9	システム・エラーが発生しました。

使用上の注意

1. EAB を戻すには、**Set Session Parameters** (9) 関数の EAB オプションを使用します。EAB は表示スペース上の各文字と関連しており、それぞれの文字の前に戻されます。
2. **DBCS の場合のみ: Set Session Parameters** (9) 関数の EAD オプションをこの関数と共に使用して、2 バイトの EAD を戻すことができます。EAB オプショ

ンを指定せずに EAD オプションを指定した場合、各文字の前に EAD が戻されます。EAB と EAD の両オプションを指定した場合、EAB の前に EAD が戻されます。

コピーの開始位置が 2 バイト文字の第 2 バイトである場合、または終了位置が 2 バイト文字の第 1 バイトである場合、これらのバイトはブランクに変換されます。

3. **Copy Presentation Space** 関数は以下のセッション・オプションの影響を受けます。
   - ATTRB/NOATTRB/NULLATTRB
   - EAB/NOEAB
   - XLATE/NOXLATE
   - BLANK/NOBLANK
   - DISPLAY/NODISPLAY
   - EAD/NOEAD (DBCS のみ)
   - NOSO/SPACESO/SO (DBCS のみ)
   - EXTEND_PS/NOEXTEND_PS

詳細については、5 (168 ページ)、 13、 14、 15、 17 (172 ページ)、 20、および 21 (173 ページ) を参照してください。

提供されたコピー先データ・ストリングが、要求データを保持するための十分な長さがない場合は、予測できない結果となる可能性があります。

前述のように、さまざまな **Copy** (5、8、および 34) 関数により戻される属性は、**Set Session Parameters** (9) 関数の影響を受けます。関係するセッション設定パラメーターは、以下のような影響を及ぼします。

セッション設定パラメーター
COPY 機能への影響

NOEAB、NOEAD
属性は戻されません。テキストのみを表示スペースからユーザーのバッファーへコピーします。

EAB、NOXLATE
属性は以下の表に定義されたとおりに戻されます。

EAB、XLATE
表示スペースの表示に使用される色が戻されます。色は再マップできるので、XLATE と EAB が同時に ON になっている場合、属性の色は **Copy** 関数によって戻される色ではありません。

EAD 以下の表に示される 2 バイト文字セット属性が戻されます。

NOSO/SPACESO/SO
NOSO を指定した場合、SPACESO として機能します。表示スペースのサイズは変わりません。

戻される文字の属性を以下の表に示します。属性ビット位置は IBM 形式 (つまり、バイトの左端がビット 0) で示してあります。

3270 の文字属性が、ホストからエミュレーターに戻されます。次の表は、EAB および NOXLATE が設定されている場合に適用されます。

ビット位置	意味
0 ～ 1	文字の強調表示 00 = 標準 01 = 明滅 10 = 反転表示 11 = 下線
2 ～ 4	文字の色 (カラー再マップでこの色定義を一時変更できます。) 000 = デフォルト値 001 = 青 010 = 赤 011 = ピンク 100 = 緑 101 = 空色 110 = 黄 111 = 白
5 ～ 6	文字属性 00 = デフォルト値 11 = 2 バイト文字
7	予約済み

5250 の文字属性が、ホストからエミュレーターに戻されます。次の表は、EAB および NOXLATE が設定されている場合に適用されます。

ビット位置	意味
0	反転イメージ 0 = 標準イメージ 1 = 反転イメージ
1	下線 0 = 下線なし 1 = 下線付き
2	明滅 0 = 明滅しない 1 = 明滅する
3	桁分離文字線 0 = 桁分離文字線なし 1 = 桁分離文字線付き
4 ～ 7	予約済み

次の表は、パーソナル・コミュニケーションズの文字カラー属性を示します。次の表は、EAB および XLATE が設定されている場合に適用されます。

ビット位置	意味
0 〜 3	背景文字の色 0000 = 黒 0001 = 青 0010 = 緑 0011 = シアン 0100 = 赤 0101 = マジェンタ 0110 = 茶 (3270)、黄 (5250) 0111 = 白
4 〜 7	前景文字の色 0000 = 黒 0001 = 青 0010 = 緑 0011 = シアン 0100 = 赤 0101 = マジェンタ 0110 = 茶 (3270)、黄 (5250) 0111 = 白 1000 = 灰色 1001 = 明るい青 1010 = 明るい緑 1011 = 明るいシアン 1100 = 明るい赤 1101 = 明るいマジェンタ 1110 = 黄 1111 = 白 (高輝度)

- 2 バイト文字セット属性 (DBCS の場合のみ)
  - 第 1 バイト

ビット位置	文字位置	フィールド属性位置
0	2 バイト文字	予約済み
1	2 バイト文字の第 1 バイト	予約済み
2	SO	予約済み

ビット位置	文字位置	フィールド属性位置
3 ～ 4	SI (ビット位置 3)	5250 DBCS 関連フィールド • ビット位置 7 が 0 のとき 00 = デフォルト値 01 = DBCS のみ 10 = DBCS と SBCS のどちらか一方 11 = DBCS と SBCS が混在 • ビット位置 7 が 1 のとき 00 = 予約済み 01 = SO/SI なしの DBCS のみ 10 = 予約済み 11 = 予約済み
5	予約済み	SO/SI 生成可能 (3270 のみ)
6	予約済み	文字属性が存在 (3270 のみ)
7	予約済み	5250 DBCS 関連拡張フィールド 0 = 基本 2 バイト・フィールド 1 = 拡張 2 バイト・フィールド

– 第 2 バイト

ビット位置	文字位置	フィールド属性位置
0	予約済み	左罫線 (3270 のみ)
1	予約済み	上罫線 (3270 のみ)
2	予約済み	右罫線 (3270 のみ)
3	予約済み	下罫線 (3270 のみ)
4	左罫線	左罫線
5	上罫線	上罫線
6 ～ 7	予約済み	予約済み

PS/2 モノクローム表示の場合、アプリケーション (ワークステーション) セッション内の文字はグレー表示されます。この機能は、EHLLAPI アプリケーション・セッション内の再マップ済みの色をユーザーに提示して、ホストのアプリケーションの表示スペースに表示されているものを確認できるようにするために必要です。

ホスト表示スペースの一部のみをコピーしたい場合は、**Copy Presentation Space to String** (8) 関数を使用します。

この関数を使用するには、戻りデータ・ストリング・パラメーターを受け入れるメモリーを事前に割り振ります。このメモリーの事前割り振りに必要なステートメントは、アプリケーションに使用される言語によって異なります。詳細については、10 ページの『メモリー割り振り』を参照してください。

注: 5250 エミュレーションは、24 行 80 桁の表示スペースをサポートします。コミュニケーション・マネージャー 5250 エミュレーションが 25 番目の行を表

示するインスタンスもあります。このようなことが起こるのは、ホストからのエラー・メッセージが表示される場合、またはオペレーターが SysReq キーを選択する場合です。パーソナル・コミュニケーションズは、25 行目の情報を 24 行目またはステータス・バーに表示します。ステータス・バーに情報が表示される場合は、ステータス・バーがその情報用に構成されている必要があります。ステータス・バーの構成方法の詳細については、「*はじめに*」を参照してください。**EXTEND_PS** オプションによって、EHLLAPI アプリケーションはコミュニケーション・マネージャー EHLLAPI と同じインターフェースを使用することができ、このような状況が発生したときに有効な表示スペースが拡張されます。

1390/1399 コード・ページ・サポート

ユニコード機能は 3270 および 5250 セッションでのみサポートされます。

ユニコード・セッションでは、ホストのコピー元の表示スペース内の文字をユニコードに変換します。属性バイトは通常ブランクに変換されます。

ユニコード・セッションでは、 XLATE オプション (**Set Session Parameters** (9) 関数を使用して指定できる) はサポートされていません。これは、もしこのオプションが発行されても、EAB は PC カラー・グラフィックス・アダプター (CGA) 形式に変換されないことを意味します。

前提呼び出し: Connect Presentation Space (1)

呼び出しパラメーター:

	標準インターフェース	**拡張インターフェース**
関数番号	必ず 5 にしてください。	
データ・ストリング	事前に割り振られたコピー先ユニコード・ストリング。拡張属性バイト (EAB) オプションを指定して **Set Session Parameters** (9) 関数を発行した場合、データ・ストリングの長さは少なくとも表示スペースのサイズの 2 倍必要です。	
長さ	NA (ホスト表示スペース長が暗黙指定されます)。	
PS 位置	NA	

戻りパラメーター: この関数はデータ・ストリングおよび戻りコードを戻します。

データ・ストリング:
表示スペースの内容のユニコード・プレゼンテーションを含むストリングが戻されます。

戻りコード:
以下のコードが定義されています。

戻りコード	**説明**
0	ホスト表示スペースの内容はアプリケーション・プログラムにコピーされました。コピー先表示スペースは活動状態にあり、キーボードはアンロックされています。
1	ユーザー・プログラムが現在ホスト・セッションに接続されていません。

戻りコード	説明
4	ホスト表示スペースの内容はコピーされました。接続されたホスト表示スペースはホストからの応答を待っています。
5	ホスト表示スペースはコピーされました。キーボードはロックされています。
9	システム・エラーが発生しました。

使用上の注意: 以下のオプションは **Copy Presentation Space (5)** のユニコード・セッションでサポートされ、DBCS の場合と同じ方法で機能します。

- NOATTRB
- ATTRB
- NULLATTRB
- EAB
- NOEAB
- NOXLATE
- DISPLAY
- NODISPLAY
- BLANK
- NOBLANK

1137 コード・ページ・サポート

ユニコード機能は 5250 セッションでのみサポートされます。

ユニコード・セッションでは、ホストのコピー元の表示スペース内の文字をユニコードに変換します。属性バイトは通常ブランクに変換されます。

ユニコード・セッションでは、 XLATE オプション (**Set Session Parameters** (9) 関数を使用して指定できる) はサポートされていません。これは、もしこのオプションが発行されても、EAB は PC カラー・グラフィックス・アダプター (CGA) 形式に変換されないことを意味します。

前提呼び出し: Connect Presentation Space (1)

呼び出しパラメーター:

	標準インターフェース	拡張インターフェース
関数番号	必ず 5 にしてください。	
データ・ストリング	事前に割り振られたコピー先ユニコード・データ・ストリング。長さ (バイトで) は、表示スペースのサイズの 2 倍 (バイトで) 必要です。拡張属性バイト (EAB) オプションを指定して **Set Session Parameters** (9) 関数を発行した場合、データ・ストリングの長さは少なくとも表示スペースのサイズの 4 倍必要です。	
長さ	NA (ホスト表示スペース長が暗黙指定されます)。	
PS 位置	NA	

戻りパラメーター: この関数はデータ・ストリングおよび戻りコードを戻します。

データ・ストリング:
表示スペースの内容のユニコード・プレゼンテーションを含むストリングが戻されます。

戻りコード:
以下のコードが定義されています。

戻りコード	説明
0	ホスト表示スペースの内容はアプリケーション・プログラムにコピーされました。コピー先表示スペースは活動状態にあり、キーボードはアンロックされています。
1	ユーザー・プログラムが現在ホスト・セッションに接続されていません。
4	ホスト表示スペースの内容はコピーされました。接続されたホスト表示スペースはホストからの応答を待っています。
5	ホスト表示スペースはコピーされました。キーボードはロックされています。
9	システム・エラーが発生しました。

使用上の注意: 以下のオプションは **Copy Presentation Space (5)** のユニコード・セッションでサポートされ、DBCS の場合と同じ方法で機能します。

- NOATTRB
- ATTRB
- NULLATTRB
- EAB
- NOEAB
- NOXLATE
- DISPLAY
- NODISPLAY
- BLANK
- NOBLANK

Copy Presentation Space to String (8)

3270	*5250*	*VT*
YES	YES	YES

Copy Presentation Space to String 関数は、ホスト接続表示スペースのすべてまたは一部を、EHLLAPI アプリケーション・プログラム内に定義したデータ・ストリングにコピーする場合に使用します。

入力 PS 位置は、ホストの表示スペースのオフセットです。このオフセットは、左上隅 (行 1、桁 1) を位置 1、右下隅を 3564 (ホスト表示スペースの最大表示画面サイズ) とするレイアウトを基にしています。 PS 位置 + (長さ - 1) の値は、ホスト表示スペースの構成サイズを超えることはできません。

Copy Presentation Space to String 関数はホストのコピー元の表示スペース内の文字を ASCII 形式に変換します。属性バイトやその他の通常 ASCII では表されない文字は、ブランクに変換されます。属性バイトをブランクに変換したくない場合には、**Set Session Parameters** (9) 関数の ATTRB オプションを指定して、この変換を一時変更することができます。

前提呼び出し

Connect Presentation Space (1)

呼び出しパラメーター

	標準インターフェース	拡張インターフェース
関数番号	必ず 8 にしてください。	
データ・ストリング	事前にホスト表示スペースのサイズを割り振られたコピー先ストリング。 EAB オプションを指定して **Set Session Parameters** (9) 関数を発行した場合、データ・ストリングの長さは少なくとも表示スペースの長さの 2 倍必要です。 **DBCS の場合のみ:** EAD オプションを指定した場合、データ・ストリングの長さは少なくとも表示スペースの長さの 3 倍必要です。 EAB と EAD の両オプションを指定した場合、データ・ストリングの長さは最低でも表示スペースの長さの 4 倍なければなりません。	
長さ	該当データ・ストリングの長さ。	
PS 位置	コピー先データ・ストリングの第 1 バイトのホスト表示スペース内の位置。	

戻りパラメーター

この関数はデータ・ストリングおよび戻りコードを戻します。

データ・ストリング:
ホスト表示スペースの内容。

戻りコード:
以下のコードが定義されています。

戻りコード	説明
0	ホスト表示スペースの内容はアプリケーション・プログラムにコピーされました。コピー先表示スペースは活動状態にあり、キーボードはアンロックされています。
1	ユーザー・プログラムが現在ホスト・セッションに接続されていません。
2	指定されたストリング長に誤りがありました。あるいは、 (データ・ストリング長 - 1) + PS 位置の合計が、接続されているホスト表示スペースのサイズを超えています。
4	ホスト表示スペースの内容はコピーされました。ホスト表示スペースはホストの応答を待っています。
5	ホスト表示スペースはコピーされました。キーボードはロックされています。

戻りコード	説明
7	ホスト表示スペースの位置が正しくありません。
9	システム・エラーが発生しました。

使用上の注意

1. EAB を戻すには、**Set Session Parameters** (9) 関数の EAB オプションを使用します。EAB は表示スペースの各文字と関連しており、それぞれの文字の後に戻されます。
2. **DBCS の場合のみ: Set Session Parameters** (9) 関数の EAD オプションをこの関数と共に使用して、2 バイトの EAD を戻すことができます。EAB オプションを指定せずに EAD オプションを指定した場合、各文字の前に EAD が戻されます。EAB と EAD の両オプションを指定した場合、EAB の後に EAD が戻されます。

   コピーの開始位置が 2 バイト文字の第 2 バイトである場合、または終了位置が 2 バイト文字の第 1 バイトである場合、これらのバイトはブランクに変換されます。EAD オプションを ON に設定した場合、1 文字につき 3 バイトずつ戻されます。EAB と EAD の両オプションを ON に設定した場合は、1 文字につき 4 バイトずつ戻されます。
3. **Copy Presentation Space to String** 関数は以下のオプションの影響を受けます。
   - ATTRB/NOATTRB/NULLATTRB
   - EAB/NOEAB
   - XLATE/NOXLATE
   - BLANK/NOBLANK
   - DISPLAY/NODISPLAY
   - EAD/NOEAD (DBCS のみ)
   - NOSO/SPACESO/SO (DBCS のみ)
   - EXTEND_PS/NOEXTEND_PS

   詳細については、5 (168 ページ)、 13、 14 (171 ページ)、 15 (171 ページ)、 17 (172 ページ)、 20 および 21 (173 ページ) を参照してください。

   指定されたコピー先のデータ・ストリングが要求されたバイト数を保持するのに十分でない場合、コピー先データ・ストリングの終わりに達した時にコピー関数は正常終了します (RC=0、4、または 5)。

   前述のように、さまざまな **Copy** (5、8、および 34) 関数により戻される属性は、**Set Session Parameters** (9) 関数の影響を受けます。関係するセッション設定パラメーターは、以下のような影響を及ぼします。

   セッション設定パラメーター
   Copy 機能への影響

   NOEAB、NOEAD
   属性は戻されません。テキストのみを表示スペースからユーザーのバッファーへコピーします。

EAB、NOXLATE
属性は以下の表に定義されたとおりに戻されます。

EAB、XLATE
表示スペースの表示に使用される色が戻されます。色は再マップできるので、XLATE と EAB が同時に ON になっている場合、属性の色は **Copy** 関数によって戻される色ではありません。

EAD 以下の表に示される 2 バイト文字セット属性が戻されます。

戻される文字の属性を以下の表に示します。属性ビット位置は IBM 形式 (つまり、バイトの左端がビット 0) で示してあります。

- 3270 の文字属性が、ホストからエミュレーターに戻されます。次の表は、EAB および NOXLATE が設定されている場合に適用されます。

ビット位置	意味
0 ～ 1	文字の強調表示 00 = 標準 01 = 明滅 10 = 反転表示 11 = 下線
2 ～ 4	文字の色 (カラー再マップでこの色定義を一時変更できます。) 000 = デフォルト値 001 = 青 010 = 赤 011 = ピンク 100 = 緑 101 = 空色 110 = 黄 111 = 白
5 ～ 7	予約済み

- 5250 の文字属性が、ホストからエミュレーターに戻されます。次の表は、EAB および NOXLATE が設定されている場合に適用されます。

ビット位置	意味
0	反転イメージ 0 = 標準イメージ 1 = 反転イメージ
1	下線 0 = 下線なし 1 = 下線付き
2	明滅 0 = 明滅しない 1 = 明滅する

ビット位置	意味
3	桁分離文字線 0 = 桁分離文字線なし 1 = 桁分離文字線付き
4 〜 7	予約済み

- VT の文字属性が、ホストからエミュレーターに戻されます。次の表は、EAB および NOXLATE が設定されている場合に適用されます。

ビット位置	意味
0 〜 3	予約済み
4	Bold 1 = オン 0 = オフ
5	下線 1 = オン 1 = オフ
6	明滅 1 = オン 0 = オフ
7	反転表示 0 = オン 1 = オフ

- 次の表は、パーソナル・コミュニケーションズの文字カラー属性を示します。次の表は、EAB および XLATE が設定されている場合に適用されます。

ビット位置	意味
0 〜 3	背景文字の色 0000 = 黒 0001 = 青 0010 = 緑 0011 = シアン 0100 = 赤 0101 = マジェンタ 0110 = 茶 (3270)、黄 (5250) 0111 = 白

ビット位置	意味
4 〜 7	前景文字の色 0000 = 黒 0001 = 青 0010 = 緑 0011 = シアン 0100 = 赤 0101 = マジェンタ 0110 = 茶 (3270)、黄 (5250) 0111 = 白 1000 = 灰色 1001 = 明るい青 1010 = 明るい緑 1011 = 明るいシアン 1100 = 明るい赤 1101 = 明るいマジェンタ 1110 = 黄 1111 = 白 (高輝度)

- 2 バイト文字セット属性
  - 第 1 バイト

ビット位置	文字位置	フィールド属性位置
0	2 バイト文字	予約済み
1	2 バイト文字の第 1 バイト	予約済み
2	SO	予約済み
3 〜 4	SI (ビット位置 3)	5250 DBCS 関連フィールド ビット位置 7 が 0 のとき 00 = デフォルト値 01 = DBCS のみ 10 = DBCS と SBCS のどちらか一方 11 = DBCS と SBCS が混在 ビット位置 7 が 1 のとき 00 = 予約済み 01 = SO/SI なしの DBCS のみ 10 = 予約済み 11 = 予約済み
5	予約済み	SO/SI 生成可能 (3270 のみ)

ビット位置	文字位置	フィールド属性位置
6	予約済み	文字属性が存在 (3270 のみ)
7	予約済み	5250 DBCS 関連拡張フィールド 0 = 基本 2 バイト・フィールド 1 = 拡張 2 バイト・フィールド

– 第 2 バイト

ビット位置	文字位置	フィールド属性位置
0	予約済み	左罫線 (3270 のみ)
1	予約済み	上罫線 (3270 のみ)
2	予約済み	右罫線 (3270 のみ)
3	予約済み	下罫線 (3270 のみ)
4	左罫線	左罫線
5	上罫線	上罫線
6 ～ 7	予約済み	予約済み

PS/2 モノクローム表示の場合、アプリケーション (ワークステーション) セッション内の文字はグレー表示されます。この機能は、EHLLAPI アプリケーション・セッション内の再マップ済みの色をユーザーに提示して、ホストのアプリケーションの表示スペースに表示されているものを確認できるようにするために必要です。

4. この関数を使用するには、戻りデータ・ストリング・パラメーターを受け入れるメモリーを事前に割り振ります。このメモリーの事前割り振りに必要なステートメントは、アプリケーションに使用される言語によって異なります。詳細については、 10 ページの『メモリー割り振り』を参照してください。

注: 5250 エミュレーションは、24 行 80 桁の表示スペースをサポートします。コミュニケーション・マネージャー 5250 エミュレーションが 25 番目の行を表示するインスタンスもあります。このようなことが起こるのは、ホストからのエラー・メッセージが表示される場合、またはオペレーターが SysReq キーを選択する場合です。パーソナル・コミュニケーションズは、25 行目の情報を 24 行目またはステータス・バーに表示します。ステータス・バーに情報が表示される場合は、ステータス・バーがその情報用に構成されている必要があります。ステータス・バーの構成方法の詳細については、「*はじめに*」を参照してください。**EXTEND_PS** オプションによって、EHLLAPI アプリケーションはコミュニケーション・マネージャー EHLLAPI と同じインターフェースを使用することができ、このような状況が発生したときに有効な表示スペースが拡張されます。

1390/1399 コード・ページ・サポート

ユニコード機能は 3270 および 5250 セッションでのみサポートされます。

ユニコード・セッションでは、ホストのコピー元の表示スペース内の文字をユニコードに変換します。属性バイトは通常ブランクに変換されます。

ユニコード・セッションでは、 XLATE オプション (**Set Session Parameters** (9) 関数を使用して指定できる) はサポートされていません。これは、もしこのオプションが発行されても、EAB は PC カラー・グラフィックス・アダプター (CGA) 形式に変換されないことを意味します。

前提呼び出し: Connect Presentation Space (1)

呼び出しパラメーター:

	標準インターフェース	**拡張インターフェース**
関数番号	必ず 8 にしてください。	
データ・ストリング	事前に割り振られたコピー先ユニコード・ストリング。拡張属性バイト (EAB) オプションを指定して **Set Session Parameters** (9) 関数を発行した場合、データ・ストリングの長さは少なくとも表示スペースの長さの 2 倍必要です。	
長さ	ユニコード文字でのコピー先ユニコード・ストリングの長さ。	
PS 位置	コピー先データ・ストリングの第 1 バイトのホスト表示スペース内の位置。	

戻りパラメーター: この関数はデータ・ストリングおよび戻りコードを戻します。

データ・ストリング:
ユニコード・データを含むストリングが戻されます。

戻りコード:
以下のコードが定義されています。

戻りコード	**説明**
0	ホスト表示スペースの内容はアプリケーション・プログラムにコピーされました。コピー先表示スペースは活動状態にあり、キーボードはアンロックされています。
1	ユーザー・プログラムが現在ホスト・セッションに接続されていません。
2	指定されたストリング長に誤りがありました。あるいは、 (データ・ストリング長 - 1) + PS 位置の合計が、接続されているホスト表示スペースのサイズを超えています。
4	ホスト表示スペースの内容はコピーされました。ホスト表示スペースはホストの応答を待っています。
5	ホスト表示スペースはコピーされました。キーボードはロックされています。
7	ホスト表示スペースの位置が正しくありません。
9	システム・エラーが発生しました。

使用上の注意: 以下のオプションは **Copy Presentation Space to String** のユニコード・セッションでサポートされ、DBCS の場合と同じ方法で機能します。

- NOATTRB
- ATTRB
- NULLATTRB
- EAB
- NOEAB
- NOXLATE
- DISPLAY
- NODISPLAY
- BLANK
- NOBLANK

1137 コード・ページ・サポート

ユニコード機能は 5250 セッションでのみサポートされます。

ユニコード・セッションでは、ホストのコピー元の表示スペース内の文字をユニコードに変換します。属性バイトは通常ブランクに変換されます。

ユニコード・セッションでは、 XLATE オプション (**Set Session Parameters** (9) 関数を使用して指定できる) はサポートされていません。これは、もしこのオプションが発行されても、EAB は PC カラー・グラフィックス・アダプター (CGA) 形式に変換されないことを意味します。

前提呼び出し: Connect Presentation Space (1)

呼び出しパラメーター:

	標準インターフェース	**拡張インターフェース**
関数番号	必ず 8 にしてください。	
データ・ストリング	事前に割り振りされたターゲットのデータ・ストリング。長さは、表示スペースからコピーされる時に必要な EBCDIC バイト数の少なくとも 2 倍必要です。拡張属性バイト (EAB) オプションを指定して **Set Session Parameters** (9) 関数を発行した場合、データ・ストリングの長さは表示スペースからコピーされる EBCDIC ストリングの長さの少なくとも 4 倍必要です。	
長さ	コピー先ユニコード・ストリングのバイト単位の長さ。この長さは、ユニコード・セッションでは最低 2 バイトなければなりません。そうでない場合は、2 のエラー・コードが戻されます。	
PS 位置	コピー先データ・ストリングの第 1 バイトのホスト表示スペース内の位置。	

戻りパラメーター: この関数はデータ・ストリングおよび戻りコードを戻します。

データ・ストリング:
ホスト表示スペースの内容。

戻りコード:
以下のコードが定義されています。

戻りコード	説明
0	ホスト表示スペースの内容はアプリケーション・プログラムにコピーされました。コピー先表示スペースは活動状態にあり、キーボードはアンロックされています。
1	ユーザー・プログラムが現在ホスト・セッションに接続されていません。
2	指定されたストリング長に誤りがありました。あるいは、 (データ・ストリング長 - 1) + PS 位置の合計が、接続されているホスト表示スペースのサイズを超えています。
4	ホスト表示スペースの内容はコピーされました。ホスト表示スペースはホストの応答を待っています。
5	ホスト表示スペースはコピーされました。キーボードはロックされています。
7	ホスト表示スペースの位置が正しくありません。
9	システム・エラーが発生しました。

使用上の注意: 以下のオプションは **Copy Presentation Space to String** のユニコード・セッションでサポートされ、SBCS の場合と同じ方法で機能します。

- NOATTRB
- ATTRB
- NULLATTRB
- EAB
- NOEAB
- NOXLATE
- DISPLAY
- NODISPLAY
- BLANK
- NOBLANK

Copy String to Field (33)

3270	*5250*	*VT*
YES	YES	YES

Copy String to Field 関数は、文字ストリングをホスト接続表示スペース内の指定したフィールドへ転送します。この関数は、*フィールド形式* のホスト表示スペースでのみ使用できます。

前提呼び出し

Connect Presentation Space (1)

呼び出しパラメーター

	標準インターフェース	拡張インターフェース
関数番号	必ず 33 にしてください。	
データ・ストリング	ホスト表示スペース内のコピー先フィールドへ転送するデータが入っているストリング。	
長さ	コピー元データ・ストリングの長さ (バイト数)。 EOT モードの場合、このパラメーターは一時変更されます。	
PS 位置	該当フィールドを識別します。これはコピー先フィールド内のどのバイトの PS 位置でも構いません。コピー関数は常にフィールドの先頭から開始されます。	

戻りパラメーター

戻りコード	説明
0	**Copy String to Field** 関数は正常終了しました。
1	ユーザー・プログラムが現在ホスト・セッションに接続されていません。
2	パラメーターが誤っているか、パラメーター長 0 がコピーに指定されました。
5	コピー先フィールドが保護または入力禁止になっているか、あるいは、無効なデータ (フィールド属性など) がコピー先フィールドに送られました。
6	コピーは完了しましたが、データが切り捨てられました。
7	ホスト表示スペースの位置が正しくありません。
9	システム・エラーが発生しました。
24	ホスト表示スペースが定様式化されていません。

使用上の注意

1. **Copy String to Field** 関数は次のオプションの影響を受けます。
   - STRLEN/STREOT
   - EOT
   - EAB/NOEAB
   - XLATE/NOXLATE
   - PUTEAB/NOPUTEAB

   詳細については、1 と 2 (167 ページ)、13、14 (171 ページ)、18 (172 ページ)、20 および 21 (173 ページ) を参照してください。
2. 転送するストリングは、呼び出しデータ・ストリング・パラメーターで指定します。次のいずれかの状況が発生したとき、ストリングは終了します。
   - **Set Session Parameters** (9) 関数を使用して EOT モードを選択していて、テキストの終了 (EOT) 区切り文字が検出された時 (165 ページの『Set Session Parameters (9)』を参照してください)。
   - 指定した長さに達した時 (EOT モードを選択していない場合)。

- フィールド内でフィールド終了が検出された時。

注: フィールドがホスト表示スペースの終わりで折り返している場合、表示スペースの終わりに到達したときに折り返します。

3. キーボードの略号 (**Send Key** (3) 関数を参照) は、 **Copy String to Field** 関数を使用して送信することはできません。
4. 転送されるデータの第 1 バイトは、常に、指定した PS 位置を含むフィールドの先頭に置かれます。
5. **DBCS の場合のみ:** ストリングに 2 バイト文字を含めることができます。

   注: PC400 は、ストリングに SO および SI を追加しません。 DBCS 混用フィールドに 2 バイト文字を含むストリングを書き込む場合は、事前に **Send Key** (3) 関数を使用して、SO および SI を生成し、2 バイト文字を書き込むための区域を作成しておいてください。

   1 つのストリング内に 1 バイト文字と 2 バイト文字が存在する場合、EBCDIC で作成されたデータの方が ASCII で作成されたデータよりも長いので、データが切り捨てられることがあります。このような場合、2 バイト文字の第 1 バイトまたは第 2 バイトだけを書き込むことはしません。

   コピー元ストリングの最後の文字が 2 バイト文字の第 1 バイトである場合、その文字は書き込まれず、長さに加算されません。

   制御文字は、フィールドの状況に応じて、1 バイト文字から 2 バイト文字へ、または 2 バイト文字から 1 バイト文字へ変換されます。 SO と SI の間にある NULL+ 制御文字の対は、1 つの 2 バイト制御文字として扱われます。例えば、以下のストリングは、1 バイト文字フィールドまたは 2 バイト文字フィールドへ、それぞれ表に示されるようにコピーされます。

ストリング	意味	1 バイト文字 フィールド	2 バイト文字 フィールド
X'000C'	(NULL)(FF) X'00'X'0C'	(SB NULL)(SB FF) X'00'X'0C'	(DB NULL)(DB FF) X'0000'X'000C'
X'0E000C0F'	(SO)(DB FF)(SI) X'0E'X'000C'X'0F'	-S エラー	(DB FF) X'000C'
注: SB は 1 バイト文字、DB は 2 バイト文字を意味します。			

注: 5250 エミュレーションは、24 行 80 桁の表示スペースをサポートします。コミュニケーション・マネージャー 5250 エミュレーションが 25 番目の行を表示するインスタンスもあります。このようなことが起こるのは、ホストからのエラー・メッセージが表示される場合、またはオペレーターが SysReq キーを選択する場合です。パーソナル・コミュニケーションズは、25 行目の情報を 24 行目またはステータス・バーに表示します。ステータス・バーに情報が表示される場合は、ステータス・バーがその情報用に構成されている必要があります。ステータス・バーの構成方法の詳細については、「*はじめに*」を参照してください。**EXTEND_PS** オプションによって、EHLLAPI アプリケーションは

コミュニケーション・マネージャー EHLLAPI と同じインターフェースを使用することができ、このような状況が発生したときに有効な表示スペースが拡張されます。

1390/1399 コード・ページ・サポート

ユニコード機能は 3270 および 5250 セッションでのみサポートされます。

STREOT オプションはユニコード・セッションではサポートされていません。詳細については、 165 ページの『Set Session Parameters (9)』を参照してください。

ユニコード・セッションでは、 XLATE オプション (**Set Session Parameters** (9) 関数を使用して指定できる) はサポートされていません。これは、もしこのオプションが発行されても、EAB は PC カラー・グラフィックス・アダプター (CGA) 形式に変換されないことを意味します。

前提呼び出し: Connect Presentation Space (1)

呼び出しパラメーター:

	標準インターフェース	**拡張インターフェース**
関数番号	必ず 33 にしてください。	
データ・ストリング	ホスト表示スペース内のコピー先フィールドへ転送するユニコード・データが入っているストリング。	
長さ	コピー元ユニコード・ストリングの長さ (ユニコード文字数)。 **注:** EOT モードはユニコード・セッションではサポートされていません。したがって、ユニコード・セッションでこの関数が正しく機能するように長さを指定する必要があります。	
PS 位置	該当フィールドを識別します。これはコピー先フィールド内のどのバイトの PS 位置でも構いません。コピー関数は常にフィールドの先頭から開始されます。	

戻りパラメーター:

戻りコード	**説明**
0	**Copy String to Field** 関数は正常終了しました。
1	ユーザー・プログラムが現在ホスト・セッションに接続されていません。
2	パラメーターが誤っているか、パラメーター長 0 がコピーに指定されました。
5	コピー先フィールドが保護または入力禁止になっているか、あるいは、無効なデータ (フィールド属性など) がコピー先フィールドに送られました。
6	コピーは完了しましたが、データが切り捨てられました。
7	ホスト表示スペースの位置が正しくありません。
9	システム・エラーが発生しました。
24	ホスト表示スペースが定様式化されていません。

使用上の注意: 以下のオプションは **Copy String to Field** のユニコード・セッションでサポートされ、DBCS の場合と同じ方法で機能します。

- STRLEN
- EAB
- NOEAB
- NOXLATE
- PUTEAB
- NOPUTEAB

1137 コード・ページ・サポート

ユニコード機能は 5250 セッションでのみサポートされます。

STREOT オプションはユニコード・セッションではサポートされていません。詳細については、 165 ページの『Set Session Parameters (9)』を参照してください。

ユニコード・セッションでは、 XLATE オプション (**Set Session Parameters** (9) 関数を使用して指定できる) はサポートされていません。これは、もしこのオプションが発行されても、EAB は PC カラー・グラフィックス・アダプター (CGA) 形式に変換されないことを意味します。

前提呼び出し: Connect Presentation Space (1)

呼び出しパラメーター:

	標準インターフェース	**拡張インターフェース**
関数番号	必ず 33 にしてください。	
データ・ストリング	ホスト表示スペース内のコピー先フィールドへ転送するユニコード・データが入っているストリング。	
長さ	コピー元ユニコード・ストリングの長さ (バイト数)。長さは、最低 2 バイトでなければなりません。そうでない場合は、2 のエラー・コードが戻されます。 **注:** EOT モードはユニコード・セッションではサポートされていません。したがって、ユニコード・セッションでこの関数が正しく機能するように長さを指定する必要があります。	
PS 位置	該当フィールドを識別します。これはコピー先フィールド内のどのバイトの PS 位置でも構いません。コピー関数は常にフィールドの先頭から開始されます。	

戻りパラメーター:

戻りコード	**説明**
0	**Copy String to Field** 関数は正常終了しました。
1	ユーザー・プログラムが現在ホスト・セッションに接続されていません。
2	パラメーターが誤っているか、パラメーター長 0 がコピーに指定されました。

戻りコード	説明
5	コピー先フィールドが保護または入力禁止になっているか、あるいは、無効なデータ (フィールド属性など) がコピー先フィールドに送られました。
6	コピーは完了しましたが、データが切り捨てられました。
7	ホスト表示スペースの位置が正しくありません。
9	システム・エラーが発生しました。
24	ホスト表示スペースが定様式化されていません。

使用上の注意: 以下のオプションは **Copy String to Field** のユニコード・セッションでサポートされ、SBCS の場合と同じ方法で機能します。

- STRLEN
- EAB
- NOEAB
- NOXLATE
- PUTEAB
- NOPUTEAB

Copy String to Presentation Space (15)

3270	*5250*	*VT*
YES	YES	YES

Copy String to Presentation Space 関数は、ASCII データ・ストリングをホスト表示スペースの PS 位置呼び出しパラメーターで指定した位置に、直接コピーします。

前提呼び出し

Connect Presentation Space (1)

呼び出しパラメーター

	標準インターフェース	拡張インターフェース
関数番号	必ず 15 にしてください。	
データ・ストリング	ホスト表示スペースにコピーする ASCII データ・ストリング。	
長さ	コピー元データ・ストリングの長さ (バイト数)。 EOT モードの場合、このパラメーターは一時変更されます。	
PS 位置	コピーを開始するホスト表示スペースの位置。 1 とホスト表示スペースの構成サイズの間の値です。	

戻りパラメーター

戻りコード	説明
0	**Copy String to Presentation Space** 関数は正常終了しました。
1	ユーザー・プログラムが現在ホスト・セッションに接続されていません。
2	パラメーターが誤っているか、パラメーター長 0 がコピーに指定されました。
5	コピー先表示スペースが保護または入力禁止になっているか、あるいは、無効なデータがコピー先表示スペースに送られました (フィールド属性バイトなど)。
6	コピーは完了しましたが、データが切り捨てられました。
7	ホスト表示スペースの位置が正しくありません。
9	システム・エラーが発生しました。

使用上の注意

1. **Copy String to Presentation Space** 関数は以下のオプションの影響を受けます。
   - STRLEN/STREOT
   - EOT
   - EAB/NOEAB
   - XLATE/NOXLATE
   - PUTEAB/NOPUTEAB
   - EAD/NOEAD (DBCS のみ)
   - NOSO/SPACESO/SO (DBCS のみ)
   - EXTEND_PS/NOEXTEND_PS

   詳細については、1 と 2 (167 ページ)、 13 と 14 (171 ページ) 、18 (172 ページ)、および 20 と 21 (173 ページ) を参照してください。
2. キーボードの略号 (**Send Key** (3) 関数を参照) は、**Copy String to Presentation Space** 関数を使用して送信することはできません。
3. **Set Session Parameters** (9) 関数を使用して EOT モードを選択した場合、テキスト終了 (EOT) 区切り文字が検出されると、ストリングが終了します (165 ページの『Set Session Parameters (9)』を参照してください)。
4. **Send Key** (3) 関数も同じ目的を果たしますが、この関数の方がプロンプトを使って応答し、コマンド入力が迅速です。 **Send Key** (3) 関数はデータをキーボードからタイプする端末オペレーターをエミュレートするので、大量のデータを操作するアプリケーションでは処理速度が遅くなります。 Copy String to Presentation Space 関数は、ホストに対するより迅速な入力パスを提供します。
5. コピー元データ (コピーされるストリング) は、表示スペースのサイズを超えることはできません。
6. **DBCS の場合のみ:** ストリングに 2 バイト文字を含めることができます。

   注: PC400 は、ストリングに SO および SI を追加しません。 DBCS 混用フィールドに 2 バイト文字を含むストリングを書き込む場合は、事前に **Send**

Key (3) 関数を使用して、SO および SI を生成し、2 バイト文字を書き込むための区域を作成しておいてください。

1 つのストリング内に 1 バイト文字と 2 バイト文字が存在する場合、EBCDIC で作成されたデータの方が ASCII で作成されたデータよりも長いので、データが切り捨てられることがあります。 2 バイト文字の第 1 バイトまたは第 2 バイトだけをストリングに書き込む必要がある場合には、ブランクが書き込まれます。

コピー元ストリングの最後の文字が 2 バイト文字の第 1 バイトである場合、その文字は書き込まれず、長さに加算されません。

コピー先表示スペースの最後の文字として書き込まれる文字が SO/SI であるか、または 2 バイト文字の第 1 バイトである場合、その文字は書き込まれず、切り捨てられ、長さに加算されません。

制御文字は、フィールドの状況に応じて、1 バイト文字から 2 バイト文字へ、または 2 バイト文字から 1 バイト文字へ変換されます。 SO と SI の間にある NULL+ 制御文字の対は、1 つの 2 バイト制御文字として扱われます。例えば、以下のストリングは、1 バイト文字フィールドまたは 2 バイト文字フィールドへ、それぞれ表に示されるようにコピーされます。

ストリング	意味	1 バイト文字 フィールド	2 バイト文字 フィールド
X'000C'	(NULL)(FF) X'00'X'0C'	(SB NULL)(SB FF) X'00'X'0C'	(DB NULL)(DB FF) X'0000'X'000C'
X'0E000C0F'	(SO)(DB FF)(SI) X'0E'X'000C'X'0F'	-S エラー	(DB FF) X'000C'
注: SB は 1 バイト文字、DB は 2 バイト文字を意味します。			

注: 5250 エミュレーションは、24 行 80 桁の表示スペースをサポートします。コミュニケーション・マネージャー 5250 エミュレーションが 25 番目の行を表示するインスタンスもあります。このようなことが起こるのは、ホストからのエラー・メッセージが表示される場合、またはオペレーターが SysReq キーを選択する場合です。パーソナル・コミュニケーションズ は、24 行目に常に同じ情報を表示します。**EXTEND_PS** オプションによって、EHLLAPI アプリケーションはコミュニケーション・マネージャー EHLLAPI と同じインターフェースを使用することができ、このような状況が発生したときに有効な表示スペースが拡張されます。

7. 一部のホスト・アプリケーションでは、この関数呼び出しにより、予期せぬ位置にカーソルが移動することがあります。フィールドへの書き込みに関しては、この関数よりも、 SendKey 関数を使用する方が良い選択です。

   注: このことが生じるのは、VT セッションの場合や、ASCII ホストに接続する場合だけです。

1390/1399 コード・ページ・サポート

ユニコード機能は 3270 および 5250 セッションでのみサポートされます。

STREOT オプションはユニコード・セッションではサポートされていません。詳細については、 165 ページの『Set Session Parameters (9)』を参照してください。

ユニコード・セッションでは、 XLATE オプション (**Set Session Parameters** (9) 関数を使用して指定できる) はサポートされていません。これは、もしこのオプションが発行されても、EAB は PC カラー・グラフィックス・アダプター (CGA) 形式に変換されないことを意味します。

前提呼び出し: Connect Presentation Space (1)

呼び出しパラメーター:

	標準インターフェース	**拡張インターフェース**
関数番号	必ず 15 にしてください。	
データ・ストリング	ホスト表示スペースに転送するユニコード・データが入っているストリング。	
長さ	コピー元ユニコード・ストリングの長さ (ユニコード文字数)。 **注:** EOT モードはユニコード・セッションではサポートされていません。したがって、ユニコード・セッションでこの関数が正しく機能するように長さを指定する必要があります。	
PS 位置	コピーを開始するホスト表示スペースの位置。 1 とホスト表示スペースの構成サイズの間の値です。	

戻りパラメーター:

戻りコード	**説明**
0	**Copy String to Presentation Space** 関数は正常終了しました。
1	ユーザー・プログラムが現在ホスト・セッションに接続されていません。
2	パラメーターが誤っているか、パラメーター長 0 がコピーに指定されました。
5	コピー先表示スペースが保護または入力禁止になっているか、あるいは、無効なデータがコピー先表示スペースに送られました (フィールド属性バイトなど)。
6	コピーは完了しましたが、データが切り捨てられました。
7	ホスト表示スペースの位置が正しくありません。
9	システム・エラーが発生しました。

使用上の注意: 以下のオプションは **Copy String to Presentation Space** のユニコード・セッションでサポートされ、DBCS の場合と同じ方法で機能します。

- STRLEN
- EAB
- NOEAB
- NOXLATE
- PUTEAB
- NOPUTEAB

1137 コード・ページ・サポート

ユニコード機能は 5250 セッションでのみサポートされます。

STREOT オプションはユニコード・セッションではサポートされていません。詳細については、 165 ページの『Set Session Parameters (9)』を参照してください。

ユニコード・セッションでは、 XLATE オプション (**Set Session Parameters** (9) 関数を使用して指定できる) はサポートされていません。これは、もしこのオプションが発行されても、EAB は PC カラー・グラフィックス・アダプター (CGA) 形式に変換されないことを意味します。

前提呼び出し: Connect Presentation Space (1)

呼び出しパラメーター:

	標準インターフェース	**拡張インターフェース**
関数番号	必ず 15 にしてください。	
データ・ストリング	ホスト表示スペースに転送するユニコード・データが入っているストリング。	
長さ	コピー元ユニコード・ストリングの長さ (ユニコード文字数)。長さは、最低 2 バイトでなければなりません。そうでない場合は、2 のエラー・コードが戻されます。 **注:** EOT モードはユニコード・セッションではサポートされていません。したがって、ユニコード・セッションでこの関数が正しく機能するように長さを指定する必要があります。	
PS 位置	コピーを開始するホスト表示スペースの位置。 1 とホスト表示スペースの構成サイズの間の値です。	

戻りパラメーター:

戻りコード	**説明**
0	**Copy String to Presentation Space** 関数は正常終了しました。
1	ユーザー・プログラムが現在ホスト・セッションに接続されていません。
2	パラメーターが誤っているか、パラメーター長 0 がコピーに指定されました。
5	コピー先表示スペースが保護または入力禁止になっているか、あるいは、無効なデータがコピー先表示スペースに送られました (フィールド属性バイトなど)。
6	コピーは完了しましたが、データが切り捨てられました。
7	ホスト表示スペースの位置が正しくありません。
9	システム・エラーが発生しました。

使用上の注意: 以下のオプションは **Copy String to Presentation Space** のユニコード・セッションでサポートされ、SBCS の場合と同じ方法で機能します。

- STRLEN
- EAB

- NOEAB
- NOXLATE
- PUTEAB
- NOPUTEAB

Disconnect from Structured Fields (121)

3270	*5250*	*VT*
YES	NO	NO

Disconnect from Structured Fields 関数は、エミュレーション・プログラムと EHLLAPI アプリケーションの間の接続を切断します。システムで終了する前に、EHLLAPI アプリケーションをエミュレーション・プログラムから切断する必要があります。以前に **Connect for Structured Fields** が発行されている場合には、EHLLAPI アプリケーションでこの関数要求を発行する必要があります。

Reset System (21) 関数も未処理の SF 接続を切断します。

前提呼び出し

Connect for Structured Fields (120)

呼び出しパラメーター

	標準インターフェース	**拡張インターフェース**
関数番号	必ず 121 にしてください。	
データ・ストリング	次の表を参照してください。	
長さ	必ず 3 にしてください。	必ず 8 にしてください。
PS 位置	NA	

データ・ストリングの内容

バイト位置		**定義**
標準	拡張	
1	1	表示スペースの 1 文字の短縮名 (PSID)。
	2 ～ 4	予約済み。
2 ～ 3	5 ～ 6	Connect for structured field (120) 関数が戻す固有の Destination/Origin ID
	7 ～ 8	予約済み。

戻りパラメーター

戻りコード	**説明**
0	**Disconnect from Structured Fields** 関数は正常終了しました。
1	無効なホスト表示スペースの短縮セッション ID が指定されたか、またはホスト表示スペースが接続されていません。

戻りコード	説明
2	パラメーター指定でエラーが発生しました。
9	システム・エラーが発生しました。
40	アプリケーションは非同期要求が保留のまま切断されました。

使用上の注意

1. **Disconnect from Structured Fields** 関数が呼び出された時点で、アプリケーションが **Get Request Completion** (125) 関数呼び出しを発行すると、非同期の **Read Structured Fields** (126) や **Write Structured Fields** (127) の関数要求の未処理のものはどれも戻されます。切断呼び出しを発行した後で実行取り消しする場合は、この関数の非同期形式を使用してください。
2. **Reset System** (21) 関数も未処理の非同期要求 (**Get Request Completion** (125) 関数をアプリケーションが検索できなかった要求) を解放します。

Disconnect Presentation Space (2)

3270	*5250*	*VT*
YES	YES	YES

Disconnect Presentation Space 関数は、EHLLAPI アプリケーション・プログラムとホスト表示スペースの間の接続を切断します。また、**Reserve** (11) 関数によりホスト表示スペースが予約済みの場合、それは **Disconnect Presentation Space** 関数の実行時に解放されます。

前提呼び出し

Connect Presentation Space (1)

呼び出しパラメーター

	標準インターフェース	拡張インターフェース
関数番号	必ず 2 にしてください。	
データ・ストリング	NA	
長さ	NA	
PS 位置	NA	

戻りパラメーター

戻りコード	説明
0	**Disconnect Presentation Space** 関数は正常終了しました。
1	ユーザー・プログラムが現在ホスト表示スペースに接続されていません。
9	システム・エラーが発生しました。

使用上の注意

1. **Disconnect Presentation Space** 関数が呼び出されると、ホスト接続表示スペースと対話する関数 (**Send Key** (3)、**Wait** (4)、 **Reserve** (11) および **Release** (12) 関数など) は無効になります。
2. EHLLAPI アプリケーションを終了する前に、接続されているホスト表示スペースを切断する必要があります。
3. **Disconnect Presentation Space** 関数は、セッション・パラメーターのデフォルト値へのリセットを行いません。デフォルト値へのリセットを行うためには、EHLLAPI アプリケーションは **Reset System** (21) 関数を呼び出す必要があります。

Disconnect Window Service (102)

3270	*5250*	*VT*
YES	YES	YES

Disconnect Window Service 関数は、EHLLAPI プログラムと指定されたホスト表示スペース・ウィンドウの間のウィンドウ・サービス接続を切断します。

前提呼び出し

Connect Window Services (101)

呼び出しパラメーター

	標準インターフェース	**拡張インターフェース**
関数番号	必ず 102 にしてください。	
データ・ストリング	次の表を参照してください。	
長さ	1	4
PS 位置	NA	

データ・ストリングの内容

バイト位置		**定義**
標準	拡張	
1	1	表示スペースの 1 文字の短縮名 (PSID)
	2 ～ 4	予約済み

戻りパラメーター

戻りコード	**説明**
0	**Disconnect Window Service** 関数は正常終了しました。
1	ユーザー・プログラムが Window Service に接続されていません。
9	システム・エラーが発生しました。

使用上の注意

Disconnect Window Service 関数を呼び出すと、アプリケーションはその表示スペース・ウィンドウを管理できなくなります。

アプリケーションを終了する前に、プレゼンテーション管理® サービス用に接続されているすべての表示スペースに対して **Disconnect Window Service** 関数を要求する必要があります。アプリケーションがウィンドウ・サービスのための接続をしたまま終了すると、サブシステムがその接続を取り消します。

Find Field Length (32)

3270	*5250*	*VT*
YES	YES	YES

Find Field Length 関数は、接続されている表示スペース内の該当フィールドの長さを戻します。この関数は、*フィールド形式* のホスト表示スペース内のフィールドに限り、保護フィールドや無保護フィールドを検索するのに使用できます。

この関数は、呼び出し PS 位置パラメーターを使用して識別したフィールドに入っている文字の数を戻します。このフィールドには、該当フィールドの先頭文字から次の属性バイトの前の文字までのすべての文字が含まれます。

前提呼び出し

Connect Presentation Space (1)

呼び出しパラメーター

	標準インターフェース	**拡張インターフェース**
関数番号	必ず 32 にしてください。	
データ・ストリング	次の表を参照してください。	
長さ	NA	NA
PS 位置	注を参照。	

注: PS 位置: Find を開始するホスト表示スペース内のフィールドを識別します。これは、**Find** を開始するフィールド内のどのバイトの PS 位置でも構いません。

2 文字の呼び出しデータ・ストリングの内容は次のとおりです。

コード	**説明**
␢␢ または T␢	このフィールド。
P␢	前のフィールド。保護、無保護のいずれでも可。
N␢	次のフィールド。保護または無保護。
NP	次の保護フィールド。
NU	次の無保護フィールド。
PP	前の保護フィールド。
PU	前の無保護フィールド。

注: ␣ 記号は、必要なブランクを表します。

戻りパラメーター

この関数はデータ・ストリング長および戻りコードを戻します。

データ・ストリング長:

次の長さが有効です。

長さ	説明
= 0	戻りコード = 28 の時、フィールドの長さが 0。戻りコード = 24 の時、ホスト表示スペースがフィールド形式ではない。
> 0	ホスト表示スペース内の必要なフィールドの長さ。

戻りコード:

以下のコードが定義されています。

戻りコード	説明
0	**Find Field Length** 関数は正常終了しました。
1	ユーザー・プログラムが現在ホスト・セッションに接続されていません。
2	パラメーター・エラーが発生しました。
7	ホスト表示スペースの位置が正しくありません。
9	システム・エラーが発生しました。
24	該当するフィールドがありません。
28	フィールドの長さが 0 バイトです。

使用上の注意

␣␣ または T␣ が呼び出しデータ・ストリングとして使用される場合を除き、見つかったフィールドが **Find** を開始したフィールドと同じ場合には、戻りコード 24 が戻されます。

Find Field Position (31)

3270	*5250*	*VT*
YES	YES	YES

Find Field Position 関数は、ホスト接続されている表示スペース内の該当フィールドの開始位置を戻します。この関数は、*フィールド形式の* ホスト表示スペース内のフィールドに限り、保護フィールドや無保護フィールドを検索するのに使用できます。

前提呼び出し

Connect Presentation Space (1)

呼び出しパラメーター

	標準インターフェース	拡張インターフェース
関数番号	必ず 31 にしてください。	
データ・ストリング	次の表を参照してください。	
長さ	NA	NA
PS 位置	注を参照。	

注: PS 位置: Find を開始するホスト表示スペース内のフィールドを識別します。これは、**Find** を開始したいフィールド内のどのバイトの PS 位置でも構いません。

2 文字の呼び出しデータ・ストリングの内容は次のとおりです。

コード	説明
␣␣ または T␣	このフィールド。
P␣	前のフィールド。保護、無保護のいずれでも可。
N␣	次のフィールド。保護または無保護。
NP	次の保護フィールド。
NU	次の無保護フィールド。
PP	前の保護フィールド。
PU	前の無保護フィールド。

注: ␣ 記号は、必要なブランクを表します。

戻りパラメーター

この関数はデータ・ストリング長および戻りコードを戻します。

データ・ストリング長:

次の長さが有効です。

長さ	説明
= 0	戻りコード = 28 の時、フィールドの長さが 0。戻りコード = 24 の時、ホスト表示スペースがフィールド形式ではない。
> 0	ホスト表示スペースの起点からの、要求されたフィールドの相対的位置。この位置は、属性バイトの直後になるように定義されています。

戻りコード:

以下のコードが定義されています。

戻りコード	説明
0	**Find Field Position** 関数は正常終了しました。
1	ユーザー・プログラムが現在ホスト・セッションに接続されていません。

戻りコード	説明
2	パラメーター・エラーが発生しました。
7	ホスト表示スペースの位置が正しくありません。
9	システム・エラーが発生しました。
24	該当するフィールドがありません。
28	フィールドの長さが 0 バイトです。

使用上の注意

ｂｂ または Tｂ が呼び出しデータ・ストリングとして使用される場合を除き、見つかったフィールドが **Find** を開始したフィールドと同じ場合には、戻りコード 24 が戻されます。

Free Communications Buffer (124)

3270	*5250*	*VT*
YES	NO	NO

Free Communications Buffer 関数は、アプリケーションがもう必要としないバッファーをメモリー・マネージメントに戻します。終了する前に、アプリケーションはバッファーを解放しなければなりません。

前提呼び出し

Allocate Communications Buffer (123)

呼び出しパラメーター

	標準インターフェース	拡張インターフェース
関数番号	必ず 124 にしてください。	
データ・ストリング	次の表を参照してください。	
長さ	必ず 6 にしてください。	必ず 8 にしてください。
PS 位置	NA	

データ・ストリングの内容

バイト位置		定義
標準	拡張	
1 〜 2	1 〜 4	必ず 0 にしてください。
3 〜 6	5 〜 8	バッファーのアドレス

戻りパラメーター

戻りコード	説明
0	**Free Communications Buffer** 関数は正常終了しました。
2	パラメーター指定でエラーが発生しました。

戻りコード	説明
9	システム・エラーが発生しました。
41	要求されたバッファーは使用中です。

使用上の注意

1. アプリケーションが使用中のバッファーを解放しようとすると、その解放要求は拒否され、戻りコード 41 が戻されます。
2. アプリケーションを終了する前に、**Allocate Communications Buffer** (123) 関数を使用して割り振られたすべての通信バッファーに対して **Free Communications Buffer** (124) 関数を要求する必要があります。
3. **Reset System** (21) 関数を使用することによっても、**Allocate Communications Buffer** (123) 関数で割り振られたバッファーを解放できます。

Get Key (51)

3270	*5250*	*VT*
YES	YES	YES

Get Key 関数を使用して、**Start Keystroke Intercept** (50) 関数によって指定したセッションからのキー・ストロークを EHLLAPI アプリケーション・プログラムで取り出し、そのキー・ストロークを処理、受け入れ、または拒否することができます。この関数をループに入れることにより、ストリングの代行受信に使用できます。

前提呼び出し

Start Keystroke Intercept (50)

呼び出しパラメーター

	標準インターフェース	拡張インターフェース
関数番号	必ず 51 にしてください。	
データ・ストリング	次の表を参照してください。	
長さ	8	12
PS 位置	NA	

データ・ストリングの内容

バイト位置		定義
標準	拡張	
1	1	次のいずれかの値: • 表示スペースの 1 文字の短縮名 (PSID) • ホスト接続表示スペースに対する関数呼び出しを示す、ブランクまたは NULL 文字
	2 ～ 4	予約済み

バイト位置		定義
2 〜 8	5 〜 11	要求されたデータのシンボリック表示用のスペース確保のためのブランク
	12	予約済み

戻りパラメーター

この関数はデータ・ストリングおよび戻りコードを戻します。

データ・ストリング:

次の表を参照してください。

バイト位置		定義
標準	拡張	
1	1	次のいずれかの値: • 表示スペースの 1 文字の短縮名 (PSID) • ホスト接続表示スペースに対する関数呼び出しを示す、ブランクまたは NULL 文字
	2 〜 4	予約済み
2	5	次のいずれかのオプションのコード文字: • ASCII が戻されたことを示す A • キー・ストローク略号を示す M • 特殊略号を示す S
3 〜 8	6 〜 11	この事前割り振りされた 6 バイトのバッファー・スペースは、キー・ストロークのエンキューおよびデキューのために内部的に使用されます。以下の組み合わせが可能: • バイト 3 に ASCII 文字、バイト 4 に X'00' • バイト 3 および 4 に 2 バイト文字 • バイト 3 にエスケープ文字 (@、または機能 9 の ESC=c オプションを使用して指定したその他の文字) およびバイト 4 に 1 バイトの関数の省略文字。 (20 ページの『ASCII 略号』を参照してください。) • バイト 5 〜 8 は、戻された ASCII 略号が 2 バイトより長い場合、バイト 3 および 4 と同様になります (例えば、Attn の ASCII 略号が @A@Q の場合、バイト 5 に @、バイト 6 に Q が入ります)。使用されない場合、バイト 5 〜 8 は 0 (X'00') に設定されます。

説明のため、戻されるデータ・ストリングの例を以下に示します。

注: @ 記号はデフォルトのエスケープ文字です。エスケープ文字の値は、**Set Session Parameters** (9) 関数の ESC=c オプションを使用して ASCII で表される任意のキー・ストロークに設定できます。このオプションを使用して、エスケープ文字を別の文字に変更している場合は、以下の例の中の @ 記号をその変更している文字に置き換えて考えてください。

16 ビット・インターフェースの場合

EAt E は表示スペースの短縮名です。キー・ストロークは ASCII として戻され (A)、戻されるキーは小文字の t です (バイト 4 ～ 8 = X'00')。

EM@2 E は表示スペースの短縮名です。キー・ストロークは略号として戻され、戻されるキーは PF2 です (バイト 5 ～ 8 = X'00')。

32 ビット・インターフェースの場合

E␣␣␣At

E は表示スペースの短縮名です。キー・ストロークは ASCII として戻され (A)、戻されるキーは小文字の t です (バイト 7～11 = X'00')。

E␣␣␣ M@2

E は表示スペースの短縮名です。キー・ストロークは略号として戻され、戻されるキーは PF2 です (バイト 8 ～ 11 = X'00')。

戻りコード:

以下のコードが有効です。

戻りコード	説明
0	**Get Key** 関数は正常終了しました。
1	無効な表示スペースが指定されました。
5	**Start Keystroke Intercept** (50) 関数で AID 専用オプションが指定されました。EHLLAPI が表示スペースに無効なキーを書き込もうとする時には、このセッション・タイプでは非 AID キーは禁止されます。
8	この表示スペースに対して、先に **Start Keystroke Intercept** (50) 関数が呼び出されていません。
9	システム・エラーが発生しました。
20	定義されていないキーの組み合わせがタイプされました。
25	要求されたキー・ストロークは入力キューで使用できません。
31	キー・ストロークのキューがオーバーフローし、キー・ストロークが失われました。

使用上の注意

1. **Get Key** 関数で戻りコード 31 が発生する場合、以下のどちらかを行ってください。
   - **Start Keystroke Intercept** (50) 関数の呼び出しデータ・ストリング長パラメーターの値を増やしてください。
   - **Get Key** 関数をもっと頻繁に実行してください。

   代行受信されたキー・ストロークは、バッファー中の 3 バイトを使用します。次の代行受信されたキー・ストロークは、隣接する 3 バイトに置かれます。**Get Key** 関数がキー・ストロークを取り出す (先入れ先出し、FIFO) と、そのキー・ストロークが使用していた 3 バイトは他のキー・ストロークで使用できるようになります。バッファーのサイズを増やすか、バッファーからキー・ストロークを取り出す頻度を増やすことによって、バッファーのオーバーフローを避けることができます。

PC/3270 において、戻りコード 31 を除去する他の方法は、再開モードで PC/3270 エミュレーターを操作することです。

2. **Send Key** (3) 関数を使用して、元のキー・ストロークと EHLLAPI アプリケーションに必要な他のキー・ストロークの両方を、ホスト接続表示スペースに渡すことができます。
3. キー・ストロークは非同期に受信され、**Start Keystroke Intercept** (50) 関数を使用して EHLLAPI アプリケーション・プログラムで指定したキー・ストロークのキューに入れられます。
4. **Get Key** 関数は read コマンドと同様に機能します。キー・ストロークが使用可能な場合、キー・ストロークはユーザーのアプリケーションで用意したデータ域に読み込まれます。
5. セッションに対するフィールド・サポートの場合、アプリケーションは Enter キーなどの AID キーだけが必要かもしれません。この場合には、**Start Keystroke Intercept** (50) 関数のオプション・コードを `D` (AID キー専用を表す) に設定します。
6. この関数を使用するには、戻りデータ・ストリング・パラメーターを受け入れるメモリーを事前に割り振ります。このメモリーの事前割り振りに必要なステートメントは、アプリケーションに使用される言語によって異なります。詳細については、10 ページの『メモリー割り振り』を参照してください。

1390/1399 コード・ページ・サポート

ユニコード機能は 3270 および 5250 セッションでのみサポートされます。

セッション・オプション ESC はユニコード・セッションではサポートされていません。このオプションを使用してユニコード文字を ESC 文字として設定することはできません。デフォルトの ESC 文字 @ をユニコード・セッションで使用してください。詳細については、165 ページの『Set Session Parameters (9)』を参照してください。

前提呼び出し: Start Keystroke Intercept (50)

呼び出しパラメーター:

	標準インターフェース	**拡張インターフェース**
関数番号	必ず 51 にしてください。	
データ・ストリング	次の表を参照してください。	
長さ	8	12
PS 位置	NA	

データ・ストリングの内容:

バイト位置		**定義**
標準	拡張	

バイト位置		定義
1	1	次のいずれかの値: • 表示スペースの 1 文字の短縮名 (PSID) • ホスト接続表示スペースに対する関数呼び出しを示す、ブランクまたは NULL 文字
	2 ～ 4	予約済み
2 ～ 8	5 ～ 11	要求されたデータのシンボリック表示用のスペース確保のためのブランク
	12	予約済み

戻りパラメーター: この関数はデータ・ストリングおよび戻りコードを戻します。

データ・ストリング:

32 ビット・インターフェースについては、次の表を参照してください。

バイト位置	定義
1	次のいずれかの値: • 表示スペースの 1 文字の短縮名 (PSID) • ホスト接続表示スペースに対する関数呼び出しを示す、ブランクまたは NULL 文字
2 ～ 4	予約済み
5	U はユニコード・セッションのオプション・コード文字です。
6 ～ 11	これらのバイトの定義は DBCS セッションの場合と似ています。異なる点は、オプション・コード文字が U の場合はユニコード文字の値がバイト 6 と 7 に保管される点です。DBCS セッションでは、オプション・コード文字が A の場合に ASCII 文字の値はバイト 3 に保管され、バイト 4 には 0X'00' が入っています。

戻りコード:

以下のコードが有効です。

戻りコード	説明
0	**Get Key** 関数は正常終了しました。
1	無効な表示スペースが指定されました。
5	**Start Keystroke Intercept** (50) 関数で AID 専用オプションが指定されました。EHLLAPI が表示スペースに無効なキーを書き込もうとする時には、このセッション・タイプでは非 AID キーは禁止されます。
8	この表示スペースに対して、先に **Start Keystroke Intercept** (50) 関数が呼び出されていません。
9	システム・エラーが発生しました。
20	定義されていないキーの組み合わせがタイプされました。
25	要求されたキー・ストロークは入力キューで使用できません。
31	キー・ストロークのキューがオーバーフローし、キー・ストロークが失われました。

1137 コード・ページ・サポート

ユニコード機能は 5250 セッションでのみサポートされます。

セッション・オプション ESC はユニコード・セッションではサポートされていません。このオプションを使用してユニコード文字を ESC 文字として設定することはできません。デフォルトの ESC 文字 @ をユニコード・セッションで使用してください。詳細については、165 ページの『Set Session Parameters (9)』を参照してください。

前提呼び出し: Start Keystroke Intercept (50)

呼び出しパラメーター:

	標準インターフェース	拡張インターフェース
関数番号	必ず 51 にしてください。	
データ・ストリング	次の表を参照してください。	
長さ	8	12
PS 位置	NA	

データ・ストリングの内容:

バイト位置		定義
標準	拡張	
1	1	次のいずれかの値: • 表示スペースの 1 文字の短縮名 (PSID) • ホスト接続表示スペースに対する関数呼び出しを示す、ブランクまたは NULL 文字
	2 ～ 4	予約済み
2 ～ 8	5 ～ 11	要求されたデータのシンボリック表示用のスペース確保のためのブランク
	12	予約済み

戻りパラメーター: この関数はデータ・ストリングおよび戻りコードを戻します。

データ・ストリング:

32 ビット・インターフェースについては、次の表を参照してください。

バイト位置	定義
1	次のいずれかの値: • 表示スペースの 1 文字の短縮名 (PSID) • ホスト接続表示スペースに対する関数呼び出しを示す、ブランクまたは NULL 文字
2 ～ 4	予約済み
5	U はユニコード・セッションのオプション・コード文字です。

バイト位置	定義
6 ～ 11	これらのバイトの定義は SBCS セッションの場合と似ています。異なる点は、オプション・コード文字が U の場合はユニコード文字の値がバイト 6 と 7 に保管される点です。 DBCS セッションでは、オプション・コード文字が A の場合に ASCII 文字の値はバイト 3 に保管され、バイト 4 には 0X'00' が入っています。

戻りコード:

以下のコードが有効です。

戻りコード	説明
0	**Get Key** 関数は正常終了しました。
1	無効な表示スペースが指定されました。
5	**Start Keystroke Intercept** (50) 関数で AID 専用オプションが指定されました。EHLLAPI が表示スペースに無効なキーを書き込もうとする時には、このセッション・タイプでは非 AID キーは禁止されます。
8	この表示スペースに対して、先に **Start Keystroke Intercept** (50) 関数が呼び出されていません。
9	システム・エラーが発生しました。
20	定義されていないキーの組み合わせがタイプされました。
25	要求されたキー・ストロークは入力キューで使用できません。
31	キー・ストロークのキューがオーバーフローし、キー・ストロークが失われました。

Get Request Completion (125)

3270	*5250*	*VT*
YES	NO	NO

Get Request Completion 関数によって、アプリケーションは EHLLAPI に対して前回発行した非同期の関数要求の状況を判断して、データ・ストリングを再び使用する前に関数パラメーターのリストを取得できます。この関数は、**Read Structured Fields** (126) や **Write Structured Fields** (127) などの関数を前回呼び出した際に、ユーザーが非同期の (A) 完了を指定した場合に限り有効です。

Get Request Completion 関数が必要な非同期要求はそれぞれ、非同期要求から固有の ID を戻します。アプリケーションはこの ID を保管する必要があります。この ID は、必要な要求を識別するために **Get Request Completion** 関数が使用する ID です。ユーザーは、この関数を使用する際に、以下の 3 つの要求オプションを指定できます。

1. 関数の要求 ID およびブランク以外のセッション短縮名を指定することにより、アプリケーションは特定の非同期関数要求を照会または待機できます。
2. X'0000' という要求 ID とブランク以外のセッションの短縮名を指定することにより、アプリケーションは特定のセッションの最初に完了する非同期関数要求を照会または待機できます。

前提呼び出し

Connect Structured Fields (120) および **Allocate Communications Buffer** (123)

および

Read Structured Fields (126) または **Write Structured Fields** (127)

呼び出しパラメーター

	標準インターフェース	拡張インターフェース
関数番号	必ず 125 にしてください。	
データ・ストリング	次の表を参照してください。	
長さ	必ず 14 にしてください。	必ず 24 にしてください。
PS 位置	NA	

データ・ストリングの内容

バイト位置		定義
標準	拡張	
1	1	表示スペースの 1 文字の短縮名 (PSID)
	2 〜 4	予約済み
2	5	N または W。N=NOWAIT (非待機) が要求されました。W=WAIT (待機) が要求されました。
	6 〜 8	予約済み
3 〜 4	9 〜 10	関数要求 ID
5 〜 6	11 〜 12	予約済み
7 〜 10	13 〜 16	予約済み
11 〜 12	17 〜 20	予約済み
13 〜 14	21 〜 24	予約済み

Get Request Completion 関数の動作は、パラメーター・ストリングの 2 番目の文字により異なります。その文字は以下のいずれかです。

N 非待機オプション: 特定の要求 ID が指定され、その関数が完了している場合、戻りコード 0 および 108 ページの『戻りパラメーター』に定義された完了済みデータ・ストリングと共に、アプリケーションに制御が戻されます。要求 ID に 0 が指定され、該当する非同期関数が完了している場合、戻りコード 0 および 108 ページの『戻りパラメーター』に定義された完了済みデータ・ストリングと共に、アプリケーションに制御が戻されます。

W 待機オプション: 特定の要求 ID が指定され、関数が完了していない場合、呼び出しは関数が完了するのを待ってからアプリケーションに戻ります。要求 ID に 0 が指定され、該当する非同期関数が完了していない場合、呼び出しは関数が完了するのを待ってから呼び出し側のアプリケーションに戻ります。アプリケーションに戻った時点で、戻りコードの値は 0 になり、データ・ストリングは 108 ページの『戻りパラメーター』に定義されたとおりに完成されます。

戻りパラメーター

バイト位置		定義
標準	拡張	
5 ～ 6	11 ～ 12	完了した非同期関数の関数番号 (126 または 127)(戻り)
7 ～ 10	13 ～ 16	完了した非同期関数呼び出しのデータ・ストリングのアドレス (アプリケーションは要求が完了するまでデータ・ストリングを再度使用することができません)。 (戻り)
11 ～ 12	17 ～ 20	完了した非同期関数呼び出しのデータ・ストリングの長さ。 (戻り)
13 ～ 14	21 ～ 24	完了した非同期関数呼び出しの戻りコード。 (戻り)

戻りコード	説明
0	**Get Request Completion** 関数は正常終了しました。
2	パラメーター指定でエラーが発生しました。
9	システム・エラーが発生しました。
38	要求された関数は完了しませんでした。
42	該当する要求がありません。

戻りコード 38 と 42 には、次のような相違点があります。

1. 戻りコード 38
   a. 特定の要求 ID とセッションが要求された場合、セッションと ID の両方とも検出されたにもかかわらず、要求が保留されている (完了状態でない)。
   b. 要求 ID ゼロと特定のセッションが要求された場合、指定されたセッションに保留されている要求があるが、それらの要求は満たされて (完了して) いない。
   c. 要求 ID ゼロとブランク・セッションが要求された場合、保留されている要求が見つかったが、どれも要求が満たされて (完了して) いない。
2. 戻りコード 42
   a. 特定の要求 ID とセッションが要求された場合、特定の要求 ID が保留状態でも完了状態でも検出されない。
   b. 要求 ID ゼロと特定のセッションが要求された場合、その特定のセッションに保留要求または完了済み要求が含まれていない。
   c. 要求 ID ゼロとブランク・セッションが要求された場合、保留要求も完了済み要求も検出されなかった。

使用上の注意

1. この関数は、ユーザーが **Read Structured Fields** または **Write Structured Fields** などの関数を前回呼び出した際に、非同期完了 (Asynchronous の A) を指定した場合に限り有効です。
2. 戻りコードが 0 の場合、アプリケーションは、要求された非同期関数の完了に関する情報が入っている、戻されたデータ・ストリングを検査する必要があります。

Lock Presentation Space API (60)

3270	*5250*	*VT*
YES	NO	NO

Lock Presentation Space API 関数によって、アプリケーションは、他の Windows 32 ビット・アプリケーションに対して、表示スペース・ウィンドウの排他制御を獲得したり解放したりすることができます。ロックされている間、他のアプリケーションは表示スペース・ウィンドウに接続することはできません。

ロックを指定したこの関数が正常に処理されると、他の EHLLAPI アプリケーションから要求された EHLLAPI 表示スペース・ウィンドウ関数は、この関数を要求したアプリケーションが表示スペースをアンロックするまで、待機させられます。ロックを行ったアプリケーションからの要求は、通常どおり処理されます。

前提呼び出し

Connect to Presentation Space (1)

呼び出しパラメーター

	標準インターフェース	**拡張インターフェース**
関数番号	必ず 60 にしてください。	
データ・ストリング	次の表を参照してください。	
長さ	必ず 3 にしてください。	必ず 8 にしてください。
PS 位置	NA	

データ・ストリングの内容

バイト位置		**定義**
標準	拡張	
1	1	表示スペースの 1 文字の短縮名 (PSID)。
	2 ～ 4	予約済み。
2	5	次のいずれかを示します。 • L - API をロックする場合。 • U - API をアンロックする場合
3	6	次のいずれかを示します。 • R - 表示スペースがすでにアプリケーションによってロックされているときに戻る場合。 • Q - 表示スペースがすでにアプリケーションによってロックされているときに、ロック要求をキューに入れる場合。
	7 ～ 8	予約済み。

戻りパラメーター

戻りコード	説明
0	**Lock Presentation Space API** 関数は正常終了しました。
1	無効なホスト表示スペースの短縮セッション ID が指定されたか、またはホスト表示スペースが接続されていません。
2	パラメーター指定でエラーが発生しました。
9	システム・エラーが発生しました。
43	API がすでに他の EHLLAPI アプリケーションによってロックされていたか (ロック要求の場合)、または API がロックされていませんでした (アンロック要求の場合)。

使用上の注意

以下の EHLLAPI 関数は、ロックされている間は待機させられます。

- **Send Key** (3)
- **Copy Presentation Space** (5)
- **Search Presentation Space** (6)
- **Copy Presentation Space to String** (8)
- **Release** (11)
- **Reserve** (12)
- **Query Field Attribute** (14)
- **Copy String to Presentation Space** (15)
- **Search Field** (30)
- **Find Field Position** (31)
- **Find Field Length** (32)
- **Copy String to Field** (33)
- **Copy Field to String** (34)
- **Set Cursor** (40)
- **Send File** (90)
- **Receive File** (91)
- 先行の **Set Sessions Parameter** (9) 関数呼び出しで CONPHYS パラメーターが設定されている **Connect to Presentation Space** (1)

これらの待機させられた要求は、ロックが解除されるまでサービスされません。ロックが解除されると、待機させられた要求は、先入れ先出し法 (FIFO) の順序で処理されます。上記以外の EHLLAPI 関数は、ロックされていない場合と同じように実行されます。ロックを要求したアプリケーションは、以下のいずれかの方法で、表示スペース・ウィンドウをアンロックします。

- まだロックを所有している間に、表示スペースとの接続を切断する。
- まだロックを所有している間に、**Reset System** (21) 関数を発行する。
- まだロックを所有している間に、アプリケーションを停止する。
- セッションを停止する。

- アンロック・オプションを指定した **Lock Presentation Space API** を正常に発行する。

アプリケーションを終了する前に、**Lock Presentation Space API** 関数を使用してロックしていたすべての表示スペース・ウィンドウをアンロックする必要があります。アプリケーションが未処理のロックをそのままにして終了するか、**Reset System** (21) 関数または **Disconnect Presentation Space** (2) 関数が発行されると、ロックは解放されます。

アプリケーションでは、短時間に限り、しかも表示スペースの排他的使用が必要なときにのみ、表示スペースをロックすることをお勧めします。

Lock Window Services API (61)

3270	*5250*	*VT*
YES	NO	NO

Lock Window Services API 関数によって、アプリケーションは、他の Windows 32 ビット・アプリケーションに対して、表示スペース・ウィンドウの排他制御を獲得したり解放したりすることができます。ロックされている間、他のアプリケーションは表示スペース・ウィンドウに接続することはできません。

ロックを指定したこの関数が正常に処理されると、他の EHLLAPI アプリケーションから要求された EHLLAPI 表示スペース・ウィンドウ関数は、この関数を要求したアプリケーションが表示スペースをアンロックするまで、待機させられます。ロックを行ったアプリケーションからの要求は、通常どおり処理されます。

前提呼び出し

Connect Window Services (101)

呼び出しパラメーター

	標準インターフェース	**拡張インターフェース**
関数番号	必ず 61 にしてください。	
データ・ストリング	次の表を参照してください。	
長さ	必ず 3 にしてください。	必ず 8 にしてください。
PS 位置	NA	

データ・ストリングの内容

バイト位置		**定義**
標準	拡張	
1	1	表示スペースの 1 文字の短縮名 (PSID)。
	2 ～ 4	予約済み。
2	5	次のいずれかを示します。 • L - API をロックする場合。 • U - API をアンロックする場合

バイト位置		定義
3	6	次のいずれかを示します。 • R - 表示スペースがすでにアプリケーションによってロックされているときに戻る場合。 • Q - 表示スペースがすでにアプリケーションによってロックされているときに、ロック要求をキューに入れる場合。
5 〜 6	11 〜 12	完了した非同期関数の関数番号 (126 または 127)(戻り)
	7 〜 8	予約済み。

戻りパラメーター

戻りコード	説明
0	**Lock Window Services API** 関数は正常終了しました。
1	無効なホスト表示スペースの短縮セッション ID が指定されたか、またはホスト表示スペースが接続されていません。
2	パラメーター指定でエラーが発生しました。
9	システム・エラーが発生しました。
38	要求された関数は完了しませんでした。
43	API がすでに他の EHLLAPI アプリケーションによってロックされていたか (ロック要求の場合)、または API がロックされていませんでした (アンロック要求の場合)。

使用上の注意

以下の EHLLAPI 関数は、ロックされている間は待機させられます。

- **Window Status** (104)
- **Change Switch List Name** (105)
- **Change PS Window Name** (106)

これらの待機させられた要求は、ロックが解除されるまでサービスされません。ロックが解除されると、待機させられた要求は、先入れ先出し法 (FIFO) の順序で処理されます。

ロックを要求したアプリケーションは、以下のいずれかの方法で、表示スペース・ウィンドウをアンロックします。

- UNLOCK オプションを指定した **Lock Window Services API** を正常に発行する。
- まだロックを所有している間に、表示スペースとの接続を切断する。
- まだロックを所有している間に、**Reset System** (21) 関数を発行する。
- まだロックを所有している間に、アプリケーションを停止する。
- セッションを停止する。

アプリケーションを終了する前に、**Lock Window Services API** 関数を使用してロックしていたすべての表示スペース・ウィンドウをアンロックする必要があります。アプリケーションがロックを未処理のまま終了すると、サブシステムはそのロックを解放します。

アプリケーションでは、短時間に限り、しかも表示スペースの排他的使用が必要なときにのみ、表示スペースをロックすることをお勧めします。

Pause (18)

3270	*5250*	*VT*
YES	YES	YES

Pause 関数は、指定した時間だけ待機します。この機能をタイミング・ループ の代わりに使用して、イベントの発生を待機することができます。 **Pause** 関数は、先に **Start Host Notification** (23) 関数が呼び出され、IPAUSE オプションが選択されている場合、ホストのイベントにより終了できます。

前提呼び出し

この関数には前提呼び出しはありません。

呼び出しパラメーター

	標準インターフェース	**拡張インターフェース**
関数番号	必ず 18 にしてください。	
データ・ストリング	NA	
長さ	2 分の 1 秒単位で待機時間を指定します	
PS 位置	NA	

戻りパラメーター

戻りコード	**定義**
0	待機時間が終了しました。
9	内部システム・エラーが発生しました。時間の結果は予測できません。
26	ホスト・セッションの表示スペースまたは OIA がすでに更新されています。詳細情報を得るには、**Query Host Update** (24) 関数を使用してください。

使用上の注意

1. **Set Session Parameters** (9) 関数を使用して、FPAUSE オプションか IPAUSE オプションを選択すると、この関数を呼び出すときの停止時間の長さに影響を与えます。詳しくは、6 (168 ページ) を参照してください。

2. 呼び出しデータ・ストリング長パラメーターに入力される値は、 **Pause** 関数が待機する時間の最大数 (2 分の 1 秒単位) です。 20 秒の休止の場合、呼び出しデータ・ストリング長パラメーターに 16 進数値の 0028 (10 進数で 40) を渡す必要があります。
3. IPAUSE オプションを使用し、Pause 関数の値がゼロの場合、Pause 関数は最大 2400 (2 分の 1 秒単位) 待機します (これより前に割り込みが発生しない場合)。 FPAUSE オプションを使用し、Pause 関数の値がゼロの場合は、この関数は即座に戻ります。
4. IPAUSE オプションを使用する時は、ホストのイベントにより一時停止の条件が満たされたら、**Query Host Update** (24) 関数を呼び出して次の **Pause** 関数の前にキューを消去してください。 **Pause** 関数は、**Query Host Update** (24) 関数が完了するまで、継続して保留状態のイベントの条件を満たします。
5. **Pause** 関数の実際の最大値は 2400 です。 **Pause** 関数は次のようなタスクには使用しないでください。
   - 非常に長い期間の遅延 (例えば、数時間に渡るものなど)。
   - システム時刻クロックの検査前および EHLLAPI プログラム実行の続行前の、適切な時間 (20 分) を超える遅延。
   - **Pause** 関数で作成される時間間隔は近似値であるため、高機能タイマーを必要とするアプリケーションを使う場合。
   - ループ内での時間間隔ゼロの設定。
6. IPAUSE 設定および割り込み可能休止は、指定されたホスト表示スペース (PS) またはオペレーター情報域 (OIA) が更新されたかどうかを EHLLAPI アプリケーションが判別できるようにします。以下の 3 つの関数を使用します。
   - **Start Host Notification** (23)
   - **Query Host Update** (24)
   - **Stop Host Notification** (25)

   Start 関数が呼び出されるときに IPAUSE を使用して、ホスト表示スペースまたは OIA (あるいはこの両方) が更新を受信するまで、アプリケーションを待機させることができます。受信が完了し、アプリケーションが変更点を判別するために **Query** 関数を発行できるようになった時点で、**Pause** は終了します。続いて、アプリケーションは **Search Presentation Space** (6) を発行し、予期した更新が行われたかどうか調べます。

Post Intercept Status (52)

3270	*5250*	*VT*
YES	YES	YES

Post Intercept Status 関数は、**Get Key** (51) 関数によって得られたキー・ストロークが受け入れられたか拒否されたかをパーソナル・コミュニケーションズ・エミュレーターに通知します。アプリケーションがキー・ストロークを拒否した場合には、**Post Intercept Status** 関数はビープ音を出します。

前提呼び出し

Start Keystroke Intercept (50)

呼び出しパラメーター

	標準インターフェース	拡張インターフェース
関数番号	必ず 52 にしてください。	
データ・ストリング	次の表を参照してください。	
長さ	必ず 2 にしてください。	必ず 8 にしてください。
PS 位置	NA	

呼び出しデータ・ストリングには次のものがあります。

バイト位置		定義
標準	拡張	
1	1	次のいずれかの値: • 表示スペースの 1 文字の短縮名 • ホスト接続表示スペースに対する関数呼び出しを示す、ブランクまたは NULL 文字
	2 ～ 4	予約済み
2	5	次のいずれかを示します。 • A — キー・ストロークの受け入れ • R — キー・ストロークの拒否
	6 ～ 8	予約済み。

戻りパラメーター

戻りコード	説明
0	**Post Intercept Status** 関数は正常終了しました。
1	無効な表示スペースが指定されました。
2	無効なセッション・オプションが指定されました。
8	この表示スペース ID に対して、先に **Start Keystroke Intercept** (50) 関数が呼び出されていません。
9	システム・エラーが発生しました。

Query Additional Field Attribute (45)

3270	*5250*	*VT*
NO	YES	NO

Query Additional Field Attribute 関数はホスト表示スペースの入力位置がある 5250 フィールドについての追加情報を戻します。この情報は定義済み構造の形式でデータ・ストリング・パラメーターに戻されます。

前提呼び出し

Connect Presentation Space (1)

呼び出しパラメーター

	標準インターフェース	拡張インターフェース
関数番号	必ず 45 にしてください。	
データ・ストリング	8 バイト長の文字ストリング。	
長さ	8 が暗黙指定されます。	
PS 位置	コピー先フィールドを識別します。これはコピー先フィールド内のどのバイトの PS 位置でも構いません。	

呼び出しデータ・ストリングには次のものがあります。

バイト位置	定義
1-8	予約済み

戻りパラメーター

この関数はデータ・ストリングおよび戻りコードを戻します。

データ・ストリング:

関数は以下のデータ・ストリングを戻します。

バイト位置	定義
1-6	予約済み
7 〜 8	以下を戻す、2 つの 8 ビット符号なし文字。 • フィールドが RTL であれば R、またフィールドが LTR であれば L。 • フィールドが大文字であれば U、フィールドが標準のケース・フィールドであれば L。

戻りコード:

以下の戻りコードが定義されています。

戻りコード	説明
0	**Query Additional Field Attribute** 関数は正常終了しました。
1	ユーザー・プログラムが現在ホスト・セッションに接続されていません。
7	ホスト表示スペースの位置が正しくありません。
9	この位置ではフィールドが検出できませんでした。
24	フィールドが定様式化されていません。

Query Close Intercept (42)

3270	*5250*	*VT*
YES	YES	YES

Query Close Intercept 関数を使用して、アプリケーションはクローズ・オプションが選択されたかどうかを識別できます。

前提呼び出し

Start Close Intercept (41)

呼び出しパラメーター

	標準インターフェース	拡張インターフェース
関数番号	必ず 42 にしてください。	
データ・ストリング	次の表を参照してください。	
長さ	必ず 1 にしてください。	必ず 4 にしてください。
PS 位置	NA	

呼び出しデータ・ストリングには次のものがあります。

バイト位置		定義
標準	拡張	
1	1	ホスト表示スペースの 1 文字の短縮セッション ID、またはホスト接続されているセッションへの照会要求を示すブランクあるいは NULL。
	2 ～ 4	予約済み

戻りパラメーター

戻りコード	説明
0	クローズ割り込みは生じませんでした。
1	プレゼンテーション・ソースが正しくありません。
2	パラメーター指定でエラーが発生しました。
8	このホスト表示スペースに対して、先に **Start Close Intercept** (41) 関数が呼び出されていません。
9	システム・エラーが発生しました。
12	セッションが停止しました。
26	最後の Query Close Intercept 呼び出し後にクローズ・インターセプトが発生しました。

Query Communications Buffer Size (122)

3270	*5250*	*VT*
YES	NO	NO

Query Communications Buffer Size 関数を使用して、アプリケーションは、エミュレーション・プログラムがサポートするバッファーの最大サイズおよび最適サイズを判別することができます。

前提呼び出し

この関数には前提呼び出しはありません。

呼び出しパラメーター

	標準インターフェース	拡張インターフェース
関数番号	必ず 122 にしてください。	
データ・ストリング	次の表を参照してください。	
長さ	必ず 9 にしてください。	必ず 20 にしてください。
PS 位置	NA	

呼び出しデータ・ストリングには次のものがあります。

バイト位置		定義
標準	拡張	
1	1	表示スペースの 1 文字の短縮名 (PSID)
	2 ～ 4	予約済み
2 ～ 3	5 ～ 8	サポートされるインバウンド・バッファーの最適サイズを示す 16 ビット/32 ビットのフィールド (戻される値)
4 ～ 5	9 ～ 12	サポートされるインバウンド・バッファーの最大サイズを示す 16 ビット/32 ビットのフィールド (戻される値)
6 ～ 7	13 ～ 16	サポートされるアウトバウンド・バッファーの最適サイズを示す 16 ビット/32 ビットのフィールド (戻される値)
8 ～ 9	17 ～ 20	サポートされるアウトバウンド・バッファーの最大サイズを示す 16 ビット/32 ビットのフィールド (戻される値)

戻りパラメーター

戻りコード	説明
0	**Query Communications Buffer Size** 関数は正常終了しました。
1	無効なホスト表示スペースの短縮セッション ID が指定されたか、またはホスト表示スペースが接続されていません。
2	パラメーター指定でエラーが発生しました。
9	システム・エラーが発生しました。
10	エミュレーション・プログラムはこの関数をサポートしていません。

使用上の注意

1. この関数を使用するようにユーザーに要求する方法はありません。これは必須の関数ではなく、アプリケーションはあらゆるシステムで実行するように調整できます。

2. 戻されるバッファー・サイズは、実際にメディアを通じて送信されるレコードのサイズを表しています。 DDM 接続の場合、**読み取り構造化フィールド**と**書き込み構造化フィールド**のデータ・バッファーに与えられる 8 バイトのヘッダーは取り除かれ、構造化フィールドの AID 値を含む 1 バイトが接頭部になります。アプリケーションは、データ・バッファー内の実際のデータ・サイズ (8 バイトのヘッダーを含めない) と、**Query Communications Buffer Size** 機能が戻すバッファー・サイズから 1 バイト引いたものを比較する必要があります。Destination/Origin 接続の場合も、**読み取り構造化フィールド**と**書き込み構造化フィールド**のデータ・バッファーに与えられる 8 バイトが取り除かれ、9 バイトがデータの接頭部になります。アプリケーションは、データ・バッファー内の実際のデータ・サイズ (8 バイトのヘッダーを含めない) と、**Query Communications Buffer Size** 機能が戻すバッファー・サイズから 9 バイト引いたものを比較する必要があります。
3. 最大バッファー・サイズは、ワークステーションのハードウェアがサポートするバイトの最大数、およびエミュレーターがサポートするバイトの最大数を表します。この最大バッファー・サイズは、ホスト側でも少なくともこれらの最大サイズを受け入れるように構成されていれば、使用することができます。
4. 戻される最適バッファー・サイズはワークステーションのハードウェアとエミュレーターの両方がサポートする最適バイト数を表しています。ネットワーク構成によっては、下限値をこの最適バッファー・サイズ未満に設定しているものもあります。このような場合、構造化フィールドのサポートには、エミュレーター構成プロファイルのデータ転送バッファー・サイズの一時変更値が使用されます。エミュレーターの構成プロファイル内のバッファー・サイズの一時変更値は、**Query Communications Buffer Size** に反映されます。

Query Communication Event (81)

3270	*5250*	*VT*
YES	YES	YES

Query Communication Event 関数を使用して、 EHLLAPI プログラムは、通信イベントが発生したかどうかを判別することができます。

前提呼び出し

Start Communication Notification (80)

呼び出しパラメーター

	拡張インターフェース
関数番号	必ず 81 にしてください。
データ・ストリング	1 文字のホスト表示スペースの短縮名、またはホスト接続されている表示スペースへの更新要求を示すブランクまたは NULL。
長さ	4 が暗黙指定されます。
PS 位置	NA

呼び出しデータ構造には、以下の要素が含まれます。

バイト位置	定義
1	表示スペースの 1 文字の短縮名 (PSID)
2 ～ 4	予約済み

戻りパラメーター

戻りコード	定義
0	関数は正常終了しました。
1	誤った PSID が指定されました。
8	この PSID に対して、先に **Start Communication Notification** (80) 関数が呼び出されていません。
9	システム・エラーが発生しました。
21	指示された PSID が接続されました。
22	指示された PSID が切断されました。

Query Cursor Location (7)

3270	*5250*	*VT*
YES	YES	YES

Query Cursor Location 関数は、カーソル位置を戻すことにより、ホスト接続表示スペース内のカーソルの位置を示します。

前提呼び出し

Connect Presentation Space (1)

呼び出しパラメーター

	標準インターフェース	拡張インターフェース
関数番号	必ず 7 にしてください。	
データ・ストリング	NA	
長さ	NA	
PS 位置	NA	

戻りパラメーター

この関数はデータ・ストリング長および戻りコードを戻します。

データ・ストリング長:
カーソルのあるホスト表示スペースの位置

戻りコード:
以下のコードが定義されています。

戻りコード	説明
0	**Query Cursor Location** 関数は正常終了しました。
1	ユーザー・プログラムが現在ホスト・セッションに接続されていません。
9	システム・エラーが発生しました。

Query Field Attribute (14)

3270	*5250*	*VT*
YES	YES	YES

Query Field Attribute 関数は、ホスト表示スペースの入力位置があるフィールドの属性バイトを戻します。この情報は、戻りデータ・ストリング長パラメーターに戻されます。

PC/3270 については、次のことに注意してください。

- 画面が不定様式の場合、戻りデータ・ストリング長パラメーターには 0 が設定されます。
- 属性バイトは C0 (16 進数) 以上です。

前提呼び出し

Connect Presentation Space (1)

呼び出しパラメーター

	標準インターフェース	拡張インターフェース
関数番号	必ず 14 にしてください。	
データ・ストリング	NA	
長さ	NA	
PS 位置	コピー先フィールドを識別します。これはコピー先フィールド内のどのバイトの PS 位置でも構いません。	

戻りパラメーター

この関数はデータ・ストリング長および戻りコードを戻します。

データ・ストリング長:
画面が定様式の場合には属性値、画面が不定様式の場合には 0 となります。

戻りコード:
以下のコードが定義されています。

戻りコード	説明
0	**Query Field Attribute** 機能は正常終了しました。
1	ユーザー・プログラムが現在ホスト・セッションに接続されていません。

戻りコード	説明
7	ホスト表示スペースの位置が正しくありません。
9	システム・エラーが発生しました。
24	属性バイトがないか、またはホスト表示スペースが定様式化されていません。

使用上の注意

戻されるフィールド属性の定義を以下の表に示します。ビット位置は、バイトの左端をビット 0 とする IBM 形式で表されます。

- 3270 フィールド属性:

ビット位置	意味
0 〜 1	共に 1。フィールド属性バイト。
2	保護/無保護 0 = 無保護データ・フィールド 1 = 保護フィールド
3	A/N 0 = 英数字データ 1 = 数値データのみ
4 〜 5	I/SPD 00 = 通常輝度、pen 検出不能 01 = 通常輝度、pen 検出可能 10 = 高輝度、pen 検出可能 11 = 非表示、pen 検出不能
6	予約済み
7	MDT 0 = フィールドは変更されていません。 1 = フィールドは変更されました。

- 5250 フィールド属性:

ビット位置	意味
0	フィールド属性フラグ 0 = フィールド属性フラグなし 1 = フィールド属性フラグあり
1	可視性 0 = 非表示 1 = 表示
2	保護/無保護 0 = 無保護データ・フィールド 1 = 保護フィールド

ビット位置	意味
3	輝度 0 = 通常輝度 1 = 高輝度
4 ～ 6	フィールド・タイプ 000 = 英数字データ: 全文字使用可。 001 = 英字のみ: 大文字と小文字、コンマ、ピリオド、ハイフン、ブランク、Dup キーが使用可。 010 = 数字シフト: 数字に対する自動シフト 011 = 数字のみ: 0 ～ 9 の数字、コンマ、ピリオド、プラス、マイナス、ブランク、Dup キーが使用可。 101 = 数字のみ: 0 ～ 9 の数字、Dup キーが使用可。 110 = 磁気ストライプ読み取り装置データのみ 111 = 符号付き数字: 0 ～ 9 の数字、プラス、マイナス、Dup キーが使用可。
7	MDT 0 = フィールドは変更されていません。 1 = フィールドは変更されました。

Query Host Update (24)

3270	*5250*	*VT*
YES	YES	YES

Query Host Update 関数により、プログラム式操作は、次のいずれかにより、ホストがホスト表示スペースまたは OIA を変更したかどうか判別できます。

- **Start Host Notification** (23) 関数が呼び出された (**Query Host Update** 関数に対する初回呼び出し時のみ)
- **Query Host Update** 関数に対する前提呼び出し (初回呼び出しを除く、**Query Host Update** 関数に対するすべての呼び出し)

前提呼び出し

Start Host Notification (23)

呼び出しパラメーター

	標準インターフェース	拡張インターフェース
関数番号	必ず 24 にしてください。	
データ・ストリング	1 文字のホスト表示スペースの短縮名、またはホスト接続されている表示スペースへの更新要求を表すブランクあるいは NULL。	
長さ	1 が暗黙指定されます。	4 が暗黙指定されます。
PS 位置	NA	

呼び出しデータ・ストリングには次のものがあります。

バイト位置		定義
標準	拡張	
1	1	表示スペースの 1 文字の短縮名 (PSID)
	2 ～ 4	予約済み

戻りパラメーター

戻りコード	定義
0	最後の呼び出し以降、更新されていません。
1	ホストの無効な表示スペースが指定されました。
8	このホスト表示スペース ID に対して、先に **Start Host Notification** (23) 関数が呼び出されていません。
9	システム・エラーが発生しました。
21	OIA が更新されました。
22	表示スペースが更新されました。
23	OIA とホスト表示スペースが更新されました。
44	印刷は、プリンター・セッションで完了しています。

使用上の注意

該当する表示スペースをデータ・ストリングに指定しておくことが必要です。ただし、更新の検査のためにホスト表示スペースに接続する必要はありません。

Query Session Status (22)

3270	*5250*	*VT*
YES	YES	YES

Query Session Status 関数は、セッション特有の情報を得るために使用します。

前提呼び出し

この関数には前提呼び出しはありません。

呼び出しパラメーター

	16 ビット	32 ビット
関数番号	必ず 22 にしてください。	

	16 ビット	32 ビット
データ・ストリング	該当表示スペースの短縮セッション ID (1 バイト) に、戻りデータ用の 17 バイトを加えた、18/20 バイトのストリング。バイト位置 1 には、次のものを入れることができます。 1. ホスト接続の表示スペースに対する要求を示すブランクまたは NULL 文字。 2. キーボード・オーナーの表示スペースに対する要求を示す * (アスタリスク)。	
長さ	必ず 18 にしてください。	必ず 20 にしてください。
PS 位置	NA	

戻りパラメーター

この関数はデータ・ストリングおよび戻りコードを戻します。

バイト位置		定義
標準	拡張	
1	1	表示スペースの 1 文字の短縮名 (PSID)
	2 ～ 4	予約済み
2 ～ 9	5 ～ 12	セッションの正式名 (プロファイル名と同じ。またはプロファイルが設定されていない場合は、短縮名と同じ)
10	13	**セッション タイプ** **D** 3270 表示装置 **E** 3270 印刷装置 **F** 5250 表示装置 **G** 5250 印刷装置 **H** ASCII VT
11	14	セッションの特性。以下のセッション特性ビットを含む 2 進数で表されています。 **ビット 0** EAB 0: セッションには基本属性があります。 1: セッションには拡張属性があります。 **ビット 1** PSS 0: セッションはプログラム・シンボルをサポートしていません。 1: セッションはプログラム・シンボルをサポートしています。 **ビット 2 ～ 7** 予約済み
12 ～ 13	15 ～ 16	ホスト表示スペースの行数。 2 進数で表されます。
14 ～ 15	17 ～ 18	ホスト表示スペースの桁数。 2 進数で表されます。
16 ～ 17	19 ～ 20	ホスト・コード・ページ数。 2 進数で表されます。
18		予約済み

戻りコード:

以下のコードが定義されています。

戻りコード	説明
0	**Query Session Status** 関数は正常終了しました。
1	ホストの無効な表示スペースが指定されました。
2	ストリング長が正しくありません。
9	システム・エラーが発生しました。

使用上の注意

1. この関数を使用するには、戻りデータ・ストリング・パラメーターを受け入れるメモリーを事前に割り振ります。このメモリーの事前割り振りに必要なステートメントは、アプリケーションに使用される言語によって異なります。詳細については、10 ページの『メモリー割り振り』を参照してください。

Query Sessions (10)

3270	*5250*	*VT*
YES	YES	YES

Query Sessions 関数により、各ホスト・セッションを記述している 16 バイト (標準インターフェースの場合は 12 バイト) のデータ・ストリングが戻されます。

前提呼び出し

この関数には前提呼び出しはありません。

呼び出しパラメーター

関数	説明	
	標準インターフェース	拡張インターフェース
関数番号	必ず 10 にしてください。	
データ・ストリング	16*n* (16 ビットの場合は 12*n*) バイト長以上の事前割り振りストリング (*n* =セッション数)。	
長さ	12*n* バイト	16*n* バイト
PS 位置	NA	

注: データ・ストリング長がセッション数に合わない場合、戻りコードは 2 になります。

戻りパラメーター

この関数は、データ・ストリング、データ・ストリング長、戻りコードを戻します。

データ・ストリング:

戻されるデータ・ストリングの長さは、16*n* バイト (標準インターフェースの場合は 12*n* バイト) です。ここで、*n* はホスト・セッションの数です。

記述子は、データ・ストリングと各セッション・タイプ、およびホスト・セッションの表示スペースのサイズと連結されています。

16 バイト (標準インターフェースの場合は 12 バイト) の各セッションの記述子の形式は次のとおりです。

バイト位置		定義
標準	拡張	
1	1	表示スペースの 1 文字の短縮名 (PSID)
	2 ～ 4	予約済み
2 ～ 9	5 ～ 12	セッションの正式名 (プロファイル名と同じ。またはプロファイルが設定されていない場合は、短縮名と同じ)
10	13	接続タイプ H=host
	14	予約済み
11 ～ 12	15 ～ 16	ホスト表示スペースのサイズ (表示形式ではなく、2 進数)。セッション・タイプが印刷セッションの場合、この値は 0 です。

データ・ストリング長:

開始されるホスト・セッションの数。

戻りコード:

以下のコードが定義されています。

戻りコード	説明
0	**Query Sessions** 関数は正常終了しました。
2	ストリング長が正しくありません。
9	システム・エラーが発生しました。

使用上の注意

1. アプリケーション・プログラムが RC=2 または RC=0 を受信した場合、活動セッションの数はデータ・ストリング長のフィールドに戻されます。この数を使用して、アプリケーション・プログラムは必要な最小ストリング長を判別することができます。
2. **Query Sessions** 関数は、CFGSIZE/NOCFGZISE セッション・オプション (詳細は 16 (172 ページ) を参照) および EXTEND_PS/NOEXTEND_PS オプション (詳細は 22 (173 ページ) を参照) の影響を受けます。

注:

1. 5250 セッションに対して **Set Session Parameters**(9) に NOCFGSIZE が設定された場合、 **Query Sessions**(10) から戻されるバイト位置 11 および 12 の表示スペースのサイズの値は、EXTEND_PS と NOEXTEND_PS のどちらを選択するかによって異なります。
2. EXTEND_PS が **Set Session Parameters**(9) で設定されたときは、**Query Sessions**(10) から戻される表示スペースのサイズに、メッセージ行がある場合にはそのサイズが含まれます。

3. NOEXTEND_PS が設定された場合には、メッセージ行の有無にかかわらず、メッセージ行のサイズは含まれません。 25 行 80 桁の表示スペースの場合、このサイズは 1920 または 2000 になります。

Query System (20)

3270	*5250*	*VT*
YES	YES	YES

EHLLAPI プログラムでは、**Query System** 関数を使用して、パーソナル・コミュニケーションズのサポート・レベルおよびシステム関連のその他の値を判別することができます。この関数は該当するシステム・データを含むストリングを戻します。この情報のほとんどは、ユーザーが戻りコード 9 (システム・エラーが発生しました) を受け取った後、IBM サポート・センターに連絡したときにサービス・コーディネーターが使用するためのものです。

この戻りストリングのバイトは『戻りパラメーター』で定義されています。

前提呼び出し

この関数には前提呼び出しはありません。

呼び出しパラメーター

	標準インターフェース	**拡張インターフェース**
関数番号	必ず 20 にしてください。	
データ・ストリング	事前に割り振られた 35 バイトのストリング	36 バイト
長さ	必ず 35 にしてください。	必ず 36 にしてください。
PS 位置	NA	

戻りパラメーター

この関数はデータ・ストリングおよび戻りコードを戻します。

データ・ストリング:

35 バイト (16 ビットの場合) または 36 バイト (32 ビットの場合) のデータ・ストリングが戻されます。バイトは以下のように定義されます。

バイト位置		**定義**
標準	拡張	
1	1	EHLLAPI バージョン番号
2 ～ 3	2 ～ 3	EHLLAPI レベル番号
4 ～ 9	4 ～ 9	予約済み
10 ～ 12	10 ～ 12	予約済み
13	13	ハードウェア・ベース、 U=判別不能
14	14	プログラム・タイプ、P=IBM パーソナル・コミュニケーションズ

バイト位置		定義
15 ～ 16	15 ～ 16	予約済み
17 ～ 18	17 ～ 18	2 バイトの ASCII 値によるパーソナル・コミュニケーションズのバージョン/レベル
19	19	予約済み
20 ～ 23	20 ～ 23	予約済み
24 ～ 27	24 ～ 27	予約済み
28 ～ 29	28 ～ 29	予約済み
	30	予約済み
30 ～ 31	31 ～ 32	2 バイトの 2 進数で表される国別コード
33 ～ 35	34 ～ 36	予約済み

戻りコード

以下のコードが定義されています。

戻りコード	説明
0	**Query System** 関数は正常終了し、データ・ストリングが戻されました。
1	EHLLAPI がロードされていません。 (PC/3270 のみ)
2	ストリング長が正しくありません。 (PC/3270 のみ)
9	システム・エラーが発生しました。

使用上の注意

この関数を使用するには、戻りデータ・ストリング・パラメーターを受け入れるメモリーを事前に割り振ります。詳細については、10 ページの『メモリー割り振り』を参照してください。

Query Window Coordinates (103)

3270	*5250*	*VT*
YES	YES	YES

Query Window Coordinates 関数は、ホスト表示スペースのウィンドウに対する座標を要求します。ウィンドウは座標レベルで戻されます。

注: (0,0) はウィンドウの最上部左方を示しています。

前提呼び出し

Connect Window Services (101)

呼び出しパラメーター

	標準インターフェース	拡張インターフェース
関数番号	必ず 103 にしてください。	
データ・ストリング	ホスト表示スペースの 1 文字の短縮セッション ID。	

	標準インターフェース	拡張インターフェース
長さ	17 が暗黙指定されます。	20 が暗黙指定されます。
PS 位置	NA	

呼び出しデータ・ストリングには次のものがあります。

バイト位置		定義
標準	拡張	
1	1	次のいずれかの値: • 表示スペースの 1 文字の短縮名 (PSID) • 現在接続されている表示スペースに対する関数呼び出しを示す、ブランクまたは NULL 文字
	2 ～ 4	予約済み
2 ～ 17	5 ～ 20	予約済み

戻りパラメーター

この関数はデータ・ストリングおよび戻りコードを戻します。

バイト位置		定義
標準	拡張	
1	1	次のいずれかの値: • 表示スペースの 1 文字の短縮セッション ID • 現在接続されている表示スペースに対する関数呼び出しを示す、ブランクまたは NULL 文字
	2 ～ 4	予約済み
2 ～ 17	5 ～ 20	以下を戻す 32 ビット符号なし整数。
2 ～ 5	5 ～ 8	XLeft — デスクトップ・ウィンドウに対する長方形ウィンドウの左部 X 座標の 32 ビット符号なし整数 (単位: ペル)
6 ～ 9	9 ～ 12	YBottom — デスクトップ・ウィンドウに対する長方形ウィンドウの下部 Y 座標の 32 ビット符号なし整数 (単位: ペル)
10 ～ 13	13 ～ 15	XRight — デスクトップ・ウィンドウに対する長方形ウィンドウの右部 X 座標の 32 ビット符号なし整数 (単位: ペル)
14 ～ 17	16 ～ 20	YTop — デスクトップ・ウィンドウに対する長方形ウィンドウの上部 Y 座標の 32 ビット符号なし整数 (単位: ペル)

戻りコード:

以下のコードが定義されています。

戻りコード	説明
0	**Query Window Coordinates** 関数は正常終了しました。

戻りコード	説明
1	ユーザー・プログラムが現在ホスト・セッションに接続されていません。
9	システム・エラーが発生しました。
12	セッションが停止しました。

Read Structured Fields (126)

3270	*5250*	*VT*
YES	NO	NO

Read Structured Fields 関数を使用して、アプリケーションはホスト・アプリケーションから構造化フィールド・データを読み取ります。呼び出しで S (同期) を指定した場合、**Read Structured Fields** が完了するまで、アプリケーションに制御が戻されます。呼び出しで A (非同期) を指定した場合、その呼び出しの直後に、アプリケーションに制御が戻されます。呼び出しで M (非同期メッセージ・モード) を指定した場合、その呼び出しの直後に、アプリケーションに制御が戻されます。アプリケーションは、このメッセージを待ちます。いずれの場合 (S、A、または M) でも、アプリケーションにより、ホストからのデータが入るバッファー・アドレスが設定されます。

この関数が非同期に正常終了するためには、以下のステートメントを適用します。

パラメーター・リストの戻りコード・フィールドには、要求された入出力の結果が含まれていない場合もあります。戻りコードが 0 でない場合、要求は失敗しています。アプリケーションでは、戻りコードに応じて、適切な処置を行わなければなりません。

この要求に対する戻りコードが 0 である場合、アプリケーションで、この関数呼び出しと共に戻される要求 ID を使用して、 **Get Request Completion** 関数呼び出しを発行し、その要求 ID に関連する関数が終了したかどうかを判別します。 **Get Request Completion** 関数呼び出しによって、以下の情報が戻されます。

1. 関数要求 ID
2. 非同期要求からのデータ・ストリング・アドレス
3. データ・ストリングの長さ
4. 完了した関数の戻りコード

前提呼び出し

Connect for Structured Fields (120) および **Allocate Communication Buffer** (123)

呼び出しパラメーター

	標準インターフェース	拡張インターフェース
関数番号	必ず 126 にしてください。	
データ・ストリング	次の表を参照してください。	

	標準インターフェース	拡張インターフェース
長さ	8、10、14 のいずれかにしてください。	20
PS 位置	NA	

呼び出しデータ・ストリングには次のものがあります。

バイト位置		定義
標準	拡張	
1	1	表示スペースの 1 文字の短縮名 (PSID)。
	2 ～ 4	予約済み。
2	5	S、A、M のいずれか **S =** 同期。制御は、読み取りが満たされるまでアプリケーションに戻りません。 **A =** 非同期。制御は、アプリケーション (イベント・オブジェクト待機可能) に即時に戻されます。 **M =** 非同期。制御は、アプリケーション (メッセージ待機可能) に即時に戻されます。
	6	予約済み。
3 ～ 4	7 ～ 8	2 バイトの Destination/Origin ID
5 ～ 8	9 ～ 12	データが読み取られる 4 バイトのバッファー・アドレス。 **Allocate Communications Buffer** (123) 関数を使用してバッファーを取得する必要があります。
9 ～ 10	13 ～ 16	予約済み。
11 ～ 12	17 ～ 20	バイト位置 2 に M を指定した場合、メッセージを受け取るウィンドウのウィンドウ・ハンドルを設定してください。メッセージは、RegisterWindowMessage (“PCSHLL”) の戻り値 (0 以外) です。
13 ～ 14		この位置にあるデータは EHLLAPI で無視されます。ただし、移行プログラムでこの位置にデータを指定していても、エラーは発生しません。このデータはアプリケーションの移行時に互換性を保つために設けられています。

戻りパラメーター

この関数はデータ・ストリングおよび戻りコードを戻します。

データ・ストリング:

A (非同期) を位置 5 (標準インターフェースの場合は位置 2) に指定して、関数が正常終了した場合、以下のデータ・ストリングが戻されます。

バイト位置		定義
標準	拡張	

バイト位置		定義
9 ～ 10	13 ～ 14	2 バイトの関数要求 ID。この関数要求 ID は、この関数呼び出しの完了を判別するために **Get Request Completion** (125) 関数で使用します。
	15 ～ 16	予約済み。
	17 ～ 20	EHLLAPI によってイベント・オブジェクト・アドレスが戻される 4 バイト値。アプリケーションは、このイベント・オブジェクトを待ちます。イベント・オブジェクトがクリアされたときに、アプリケーションは **Get Request Completion** (125) 関数呼び出しを発行しなければなりません (32 ビットの場合のみ)。

注: イベント・オブジェクト・アドレスは、非同期要求が正常終了するたびに戻されます。イベント・オブジェクトは再使用しないでください。要求ごとに新しいイベント・オブジェクトが戻され、そのイベント・オブジェクトはその要求の間だけ有効です。

データ・ストリング:

"M" (非同期メッセージ・モード) をバイト位置 5 (16 ビットのアプリケーションの場合は 2) に指定して、関数が正常終了した場合、以下のデータ・ストリングが戻されます。

バイト位置		定義
9 ～ 10	13 ～ 14	2 バイトの関数要求 ID。この関数要求 ID は、この関数呼び出しの完了を判別するために **Get Request Completion (125)** 関数で使用します。
	15 ～ 16	予約済み。
11 ～ 12	17 ～ 18	非同期メッセージ・モードのタスク ID。
	19 ～ 20	予約済み。

注: 関数が正常終了した場合、アプリケーション・ウィンドウはメッセージを受け取ります。メッセージは、RegisterWindowMessage(PCSHLL) の戻り値です。wParam パラメーターには、関数呼び出しによって戻されたタスク ID が入ります。 lParam パラメーターの HIWORD には戻りコード 0 (関数の正常終了) が、LOWORD には関数番号 126 が入ります。

戻りコード:

以下のコードが定義されています。

戻りコード	説明
0	**Read Structured Fields** 関数は正常終了しました。
1	無効なホスト表示スペースの短縮セッション ID が指定されたか、またはホスト表示スペースが接続されていません。
2	パラメーター指定でエラーが発生しました。
9	システム・エラーが発生しました。
11	リソースが使用できません。(メモリーが使用できません。)

戻りコード	説明
35	要求はリジェクトされました。ホストからのアウトバウンド送信が取り消されました。
36	要求はリジェクトされました。ホストと接続できません。
37	関数は正常終了しましたが、ホストでのインバウンド送信が使用できません。

使用上の注意

1. 戻りコード 35。これは、ホストからのアウトバウンド送信が取り消された後に **Read Structured Fields** または **Write Structured Fields** が要求された場合に戻されます。アプリケーションで修正処置を行う必要があります。
2. 戻りコード 36。この場合、アプリケーションをエミュレーション・プログラムから切断してから再接続して、ホストとの通信を再確立する必要があります。アプリケーションで修正処置を行う必要があります。
3. 戻りコード 37。これはホスト・インバウンドが使用不可の場合に戻されます。**Read Structured Fields** 関数は正常に要求されました。
4. EHLLAPI では、アプリケーションごとに 20 までの非同期要求を未処理にすることができます。 20 を超える非同期要求が試行されると、リソースが使用不可であることを示す戻りコード (RC=11) が戻されます。
5. IBM グローバル・ネットワーク®接続を使用する場合、非同期要求の最大数は 10 です。

構造化フィールド・データには、ホストからのアプリケーション構造化フィールドが含まれています。構造化フィールド・データがアプリケーションに達する前に、EHLLAPI が構造化フィールド・ヘッダーを除去します。

構造化フィールド・データの形式は以下のとおりです。

オフセット	長さ	内容
0	1 ワード	X'0000'
2	1 ワード	m (メッセージの長さ。すなわち、メッセージ中のデータのバイト数。この数は 8 バイトを含むメッセージ・ヘッダーの接頭部を除いたものです)。EHLLAPI がこの値を戻します。
4	1 ワード	n (バッファー・サイズ。すなわち、8 バイトのメッセージ・ヘッダーを除くデータ・バッファーの長さを示します)。この値は、アプリケーションが設定する必要があります。
6	1 ワード	X'C000'
8	8 バイト	先頭の (または唯一の) 構造化フィールド・メッセージの長さ。
10	1 バイト	構造化フィールド・メッセージの長さ以外の最初のバイト。
		. . .
m+7	1 バイト	構造化フィールド・メッセージの末尾のバイト。

バイト 0 から 7 は、バッファー・ヘッダーです。最初のこれらの 8 バイトは、エミュレーション・プログラムで使用されます。バッファーのユーザー・セッションは、オフセット 8 からです。バイト 8 および 9 は、最初の構造化フィールド (構造化フィールド・メッセージには複数の構造化フィールドを含むことができます) 内のバイト数が入っています (バイト 8 および 9 に対する 2 バイトなど)。バイト 8 から *m* +7 は、構造化フィールド・メッセージがホスト (複数の構造化フィールドを入れることができます) から戻される場所です。

使用中のアプリケーションは、オフセット 0 のワードを 0 に設定して、完全なバッファーを提供しなければなりません。バッファー長は、オフセット 4 のワードになければなりません。オフセット 6 のワードは X'C000' にする必要があります。エミュレーション・プログラムは、データ・メッセージをオフセット 8 から入れ、メッセージの長さをオフセット 2 のワードに入れます。バッファー長は EHLLAPI により妨害されません。

同期要求: Read Structured Fields が同期要求 (データ・ストリング中の S オプション) された際には、制御は要求が満たされた後に限りアプリケーションに戻されます。アプリケーションでは以下のことを想定できます。

- 戻りコードが正しい。
- 通信バッファー (読み取りバッファー) 内のデータが正しい。
- ホストが今後 **Read Structured Fields** 要求を処理しない。

非同期要求: Read Structured Fields が非同期要求 (データ・ストリング中の A オプション) された際には、アプリケーションでは以下のことを*想定できません*。

- 戻りコードが正しい。
- 通信バッファー (読み取りバッファー) 内のデータが正しい。
- ホストが今後 **Read Structured Fields** 要求を処理しない。

非同期要求がなされた際には、EHLLAPI は以下の値を戻します。

- 16 ビット要求 ID (データ・ストリングのバイト位置 13 ～ 14) (標準インターフェースの場合は 9 ～ 10)
- イベント・オブジェクトのアドレス (データ・ストリングのバイト位置 17 ～ 20)

これらは、非同期の **Read Structured Fields** 呼び出しを完了させるために使用します。

非同期の **Read Structured Fields** 関数呼び出しの結果を判別するためには、次のステップを行わなければなりません。

- EHLLAPI 戻りコードが 0 でない場合、要求は失敗しています。したがって、非同期要求は行われていません。アプリケーションでは、再び呼び出しを試みる前に、適切な処置を行わなければなりません。
- 戻りコードが 0 ならば、アプリケーションは、 **Get Request Completion** (125) 関数または **Wait For Single Object** を使用してイベント・オブジェクトが通知状態になるまで待たなければなりません。イベント・オブジェクトは再使用しないでください。イベント・オブジェクトは、**Read Structured Fields** 関数呼び出しから **Get Request Completion** (125) 関数呼び出し完了までの間で有効です。
- イベント・オブジェクトが通知状態になったら、**Get Request Completion** (125) 関数に対する呼び出しで要求 ID パラメーターとして戻された 16 ビットの要求

ID を使用してください。 **Get Request Completion** (125) 関数呼び出しから戻されたデータ・ストリングには、**Read Structured Fields** 関数呼び出しの最終的な戻りコードが含まれています。

Read Structured Fields が非同期要求 (データ・ストリング中の M オプション) された際には、アプリケーションでは以下のことを想定できません。

- 戻りコードが正しい。
- 通信バッファー (読み取りバッファー) 内のデータが正しい。
- ホストが今後 **Read Structured Fields** 要求を処理しない。

M オプションを指定して非同期要求がなされた際には、EHLLAPI は以下の値を戻します。

- 16 ビット要求 ID (データ・ストリングのバイト位置 13 〜 14) (標準インターフェースの場合は 9 〜 10)
- 非同期メッセージ・モードのタスク ID (データ・ストリングのバイト位置 17 〜 18) (標準インターフェースの場合は 11 〜 12)

これらは、非同期の **Read Structured Fields** 呼び出しを完了させるために使用します。

Receive File (91)

3270	*5250*	*VT*
YES	YES	NO

Receive File 関数は、ファイルをホストのセッションからワークステーションのセッションへ転送する際に使用します。この機能の使用方法は、PC/3270 における RECEIVE コマンドと同様です。 **Receive File** 関数は EHLLAPI アプリケーション・プログラムによって呼び出すことができます。

前提呼び出し

この関数には前提呼び出しはありません。

呼び出しパラメーター

	標準インターフェース	**拡張インターフェース**
関数番号	必ず 91 にしてください。	
データ・ストリング	以下の例を参照してください。	
長さ	データ・ストリングの長さ (バイト数)。 EOT モードの場合、このパラメーターは一時変更されます。	

1 バイト文字セット (SBCS) のデータ・ストリングの例を以下に示します。

3270 セッション

- VM/CMS ホスト・システムからファイルを受信する場合

  pc_filename [*id:*]*fn ft* [*fm*] [(*option*]
- MVS™/TSO ホスト・システムからファイルを受信する場合

pc_filename[*id:*]*dataset*[(*member*)] [/*password*] [*option*]

- CICS® ホスト・システムからファイルを受信する場合

  pc_filename [*id:*]*host_filename* [(*option*]

5250 セッション

- iSeries、eServer i5、またはSystem i5 ホスト・システムからファイルを受け取る場合

  pc_filename [*id:*]*library file member* [*option*]

2 バイト文字セット (DBCS) のデータ・ストリングの例を以下に示します。

3270 セッション

- VM/CMS ホスト・システムからファイルを受信する場合

  pc_filename [*id:*]*fn ft* [*fm*] [(*option*]

- MVS/TSO ホスト・システムからファイルを受信する場合

  pc_filename [*id:*]*dataset*[(*member*)] [/*password*]
  [(*option*]

- CICS ホスト・システムからファイルを受信する場合

  pc_filename [*id:*]*host_filename* [(*option*]

5250 セッション

- iSeries、eServer i5、または System i5 ホスト・システムからファイルを受け取る場合

  pc_filename [*id:*]*library file member* [*option*]

注: [ ] で囲まれたパラメーターはオプションです。指定できるオプションは、次の表のとおりです。

ホスト・システム	共通オプション
VM/CMS	ASCII、ASCII、CRLF、APPEND、TIME n、CLEAR、NOCLEAR、PROGRESS、QUIET
MVS/TSO	ASCII、ASCII、CRLF、APPEND、TIME (n)、CLEAR、NOCLEAR、PROGRESS、QUIET、AVBLOCK|TRACKS|CYLINDERS
CICS	ASCII、ASCII、CRLF、NOCRLF、BINARY、TIME n、CLEAR、NOCLEAR、PROGRESS、QUIET
i5/OS™ または OS/400®	ASCII、ASCII、CRLF、APPEND、TIME n、CLEAR、NOCLEAR、PROGRESS、QUIET
注: ASCII は日本の DBCS セッションでのみ有効であり、ASCII は、その他すべての SBCS セッションと DBCS セッションで有効です。 指定されたその他のオプションは、ホスト転送プログラムに渡されます。ホスト側のファイル転送プログラムは、これらを使用するか、無視するか、エラーを戻すかのいずれかです。ホスト転送プログラムのマニュアルで、サポートされるオプションの完全なリストを調べてください。	

戻りパラメーター

戻りコード	説明
2	パラメーター・エラーが発生したか、または EHLLAPI バッファーに対して長すぎる (255 バイトを超過) データ・ストリング長が指定されました。ファイル転送が失敗しました。
3	ファイル転送は完了しました。
4	ファイル転送は完了し、レコードは分割されました。
9	システム・エラーが発生しました。
27	Cancel ボタンまたはタイムアウト (**Set Session Parameter** (9) 関数により設定) によってファイル転送は終了しました。
101	ファイル転送 (CICS への/CICS からの転送) は正常に終了しました。

システム内、またはユーザーのデータ・ストリングの指定方法に問題がある場合には、戻りコード 2 または 9 が戻されます。

他の戻りコードも受信される場合があります。これらの戻りコードは、ホスト転送プログラムにより生成されるメッセージ番号に関連しています。CICS ホスト転送プログラムに転送する場合、その戻りコードから 100 を減算するとメッセージの数値部分を得ることができます。例えば、戻りコードが 101 の場合には、ホストによりメッセージ番号 INW0001 が出されたことを示します。他のホスト転送プログラムの場合には、そのメッセージの数字部分のみ使用してください。例えば、戻りコードが 34 の場合には、ホストによりメッセージ番号 TRANS34 が出されたことを示します。特定のメッセージの意味については、ご使用のホスト転送プログラムの資料を参照してください。

EHLLAPI により報告されたオペレーティング・システムのエラー・コードは、300 より大きい数字です。エラー・コードを判別するには、300 を引いてから、オペレーティング・システムの資料でその戻りコードを調べてください。

使用上の注意

1. **Set Session Parameters** (9) 関数での 4 つのパラメーター・セットがこの関数に関連しています。それらは STRLEN/STREOT、EOT=c、QUIET/NOQUIET および TIMEOUT=c/TIMEOUT=0 の各セッション・オプションです。詳細については、 1、 2 (167 ページ)、 7 および 8 (168 ページ) を参照してください。
2. **Receive File** 関数を実行時にパスを指定しない場合、受信されたファイルは現行のサブディレクトリーに保管されます。このサブディレクトリーは、ユーザーのアプリケーションが実行されているディレクトリーです。

Release (12)

3270	*5250*	*VT*
YES	YES	YES

Release 関数は、**Reserve** (11) 関数により予約されている表示スペースに関連するキーボードをアンロックします。

前提呼び出し

Connect Presentation Space (1)

呼び出しパラメーター

	標準インターフェース	拡張インターフェース
関数番号	必ず 12 にしてください。	
データ・ストリング	NA	
長さ	NA	
PS 位置	NA	

戻りパラメーター

戻りコード	説明
0	**Release** 関数は正常終了しました。
1	ユーザー・プログラムが現在ホスト・セッションに接続されていません。
9	システム・エラーが発生しました。

使用上の注意

Release (11) 関数で予約されているホストの表示スペースに対して **Reserve** 関数を使用しないと、**Reset System** (21) 関数または **Disconnect Presentation Space** (2) 関数を呼び出すか、EHLLAPI アプリケーション・プログラムを終了させるまで、ユーザーはセッションからロックアウトされます。

Reserve (11)

3270	*5250*	*VT*
YES	YES	YES

Reserve 関数は、端末オペレーターからのブロック入力に対するホスト接続表示スペースに関連するキーボードをロックします。

予約済みのホスト表示スペースは、次のいずれかが発生するまでロック状態を維持します。

- 新しいセッションに対する **Connect** (1) 関数の実行
- **Disconnect Presentation Space** (2) 関数の実行
- **Release** (12) 関数の実行
- **Reset System** (21) 関数の実行
- **Start Keystroke Intercept** (50) 関数の実行
- EHLLAPI アプリケーション・プログラムの終了

前提呼び出し

Connect Presentation Space (1)

呼び出しパラメーター

	標準インターフェース	拡張インターフェース
関数番号	必ず 11 にしてください。	
データ・ストリング	NA	
長さ	NA	
PS 位置	NA	

戻りパラメーター

戻りコード	説明
0	**Reserve** 関数は正常終了しました。
1	ユーザー・プログラムが現在ホスト・セッションに接続されていません。
5	表示スペースの使用が禁止されています。
9	システム・エラーが発生しました。

使用上の注意

1. EHLLAPI アプリケーション・プログラムで一連のトランザクションをホストへ送信するとき、アプリケーションの処理が終了するまで、そのセッションに対するアクセスを阻止する必要があります。
2. この関数でキーボードがロックされている間にユーザーが行ったキーボード入力はキューに入り、セッション終了後に処理されます。
3. この関数は、マウス入力とキーボード入力の両方をロックします。マウス入力またはキーボード入力を使用可能にするためには、アプリケーション・プログラムで表示スペースをアンロックする必要があります。

Reset System (21)

3270	*5250*	*VT*
YES	YES	YES

Reset System 関数は、EHLLAPI を開始状態に初期化し直します。セッションのパラメーター・オプションはデフォルト値にリセットされます。イベント通知は停止します。予約済みのホスト・セッションは解放されます。ホスト表示スペースは切断されます。キー・ストロークの代行受信は使用不可になります。

Reset System 関数は、初期化時またはプログラム終了時にシステムの既知の初期化状態にリセットするために使用できます。

前提呼び出し

この関数には前提呼び出しはありません。

呼び出しパラメーター

	標準インターフェース	拡張インターフェース
関数番号	必ず 21 にしてください。	
データ・ストリング	NA	
長さ	NA	
PS 位置	NA	

戻りパラメーター

戻りコード	定義
0	**Reset System** 関数は正常終了されました。
1	EHLLAPI がロードされていません。
9	システム・エラーが発生しました。

使用上の注意

PC/3270 では、EHLLAPI がロードされているかどうかを確認するために、この関数を使用できます。アプリケーション開始時にこの関数を呼び出し、戻りコードが 1 かどうかを確認してください。

Search Field (30)

3270	*5250*	*VT*
YES	YES	YES

Search Field 関数は、指定されたストリングの存在について、接続されたホストの表示スペースのフィールドを検索します。ターゲット・ストリングが検出された場合、この関数はホスト表示スペースの先頭から数えたストリングの位置を 10 進数で戻します。 (例えば、24 行 80 桁の表示スペースでは、行 1 桁 1 の位置は番号 *1* になり、行 5 桁 1 の位置は番号 *321* になります。)

この関数は、*フィールド形式* のホスト表示スペース内のフィールドに限り、保護フィールドまたは無保護フィールドを検索するために使用することができます。

注: フィールドがホスト表示スペースの終わりで折り返している場合、表示スペースの終わりに到達したときに折り返します。

前提呼び出し

Connect Presentation Space (1)

呼び出しパラメーター

	標準インターフェース	拡張インターフェース
関数番号	必ず 30 にしてください。	
データ・ストリング	検索するデータ・ストリング。	

	標準インターフェース	拡張インターフェース
長さ	該当データ・ストリングの長さ。 EOT モードの場合、このパラメーターは一時変更されます。	
PS 位置	該当フィールドを識別します。 SRCHALL に対しては、このパラメーターは該当フィールド内のどのバイトの PS 位置でも構いません。 SRCHFROM に対しては、このパラメーターは SRCHFRWD 検索の開始位置または SRCHBKWD の終了位置を示します。 3 の使用上の注意を参照してください。	

戻りパラメーター

この関数はデータ・ストリング長および戻りコードを戻します。

データ・ストリング長:

以下のコードが定義されています。

長さ	説明
= 0	ストリングが検出されませんでした。
> 0	示されたホスト表示スペースの位置でストリングが検出されました。

戻りコード:

以下のコードが定義されています。

戻りコード	説明
0	**Search Field** 関数は正常終了しました。
1	ユーザー・プログラムが現在ホスト・セッションに接続されていません。
2	パラメーター・エラーです。ストリング長が 0 か、または EOT モードが指定されましたが、呼び出しデータ・ストリング内に EOT 文字がありません。
7	ホスト表示スペースの位置が正しくありません。
9	システム・エラーが発生しました。
24	検索ストリングが検出されなかったか、またはホストの表示スペースが定様式化されていません。

使用上の注意

1. **Set Session Parameters** (9) 関数での 4 つのパラメーター・セットがこの関数に関連しています。それらは SRCHALL/SRCHFROM、STRLEN/STREOT、SRCHFRWD/SRCHBKWD、および EOT=c の各セッション・オプションです。詳細については、 1 (167 ページ) 〜 4 (167 ページ) を参照してください。
2. **Set Session Parameters** (9) 関数を使用して、ユーザーは検索をフィールドの先頭から終わりに向かって実行するのか (SRCHFRWD) またはその逆方向に実行するのか (SRCHBKWD) を決定することができます。
3. **Search Field** 関数は通常フィールド内全体をチェックします (SRCHALL がデフォルト・モードです)。しかし、機能 9 を使用して SRCHFROM を指定すること

ができます。このモードでは、呼び出し PS 位置パラメーターは、該当フィールドの識別以外のことも実行します。つまり、検索の開始点または終了点も指定します。

- SRCHFRWD オプションが有効な場合、指示されたストリングの検索を、指定された PS 位置から開始し、フィールドの終わりに向かって実行します。
- SRCHBKWD オプションが有効な場合、指示されたストリングの検索を、フィールドの終わりから開始し、指定された PS 位置に向かって逆方向に実行します。該当ストリングが検出されない時には、検索は呼び出し PS 位置パラメーターで指定された PS 位置で停止します。

4. **DBCS の場合のみ:** 指定した検索関数の開始位置が 2 バイト文字の第 2 バイトである場合、検索は SRCHFRWD のときは次の文字から、SRCHBKWD の時はその文字から開始されます。指定したストリングの最後の文字が 2 バイト文字の第 1 バイトである場合、その文字は検索対象となりません。

   検索時には、表示スペース内では SO と SI の対は無視されます。 2 バイト制御文字を検索するには、その文字の前後に SO (X'0E') と SI (X'0F') を付けてください。例えば、データ・ストリング内の X'0E000C0F' は 2 バイト文字 FF (X'000C') として扱われます。

注: 5250 エミュレーションは、24 行 80 桁の表示スペースをサポートします。コミュニケーション・マネージャー 5250 エミュレーションが 25 番目の行を表示するインスタンスもあります。このようなことが起こるのは、ホストからのエラー・メッセージが表示される場合、またはオペレーターが SysReq キーを選択する場合です。パーソナル・コミュニケーションズは、25 行目の情報を 24 行目またはステータス・バーに表示します。ステータス・バーに情報が表示される場合は、ステータス・バーがその情報用に構成されている必要があります。ステータス・バーの構成方法の詳細については、「*はじめに*」を参照してください。**EXTEND_PS** オプションによって、EHLLAPI アプリケーションはコミュニケーション・マネージャー EHLLAPI と同じインターフェースを使用することができ、このような状況が発生したときに有効な表示スペースが拡張されます。

1390/1399 コード・ページ・サポート

ユニコード機能は 3270 および 5250 セッションでのみサポートされます。

STREOT オプションはユニコード・セッションではサポートされていません。詳細については、 165 ページの『Set Session Parameters (9)』を参照してください。

前提呼び出し: Connect Presentation Space (1)

呼び出しパラメーター:

	標準インターフェース	**拡張インターフェース**
関数番号	必ず 30 にしてください。	
データ・ストリング	検索対象のターゲット・ユニコード・ストリング。	

	標準インターフェース	拡張インターフェース
長さ	ユニコード文字でのターゲット・ユニコード・ストリングの長さ。 **注:** EOT モードはユニコード・セッションではサポートされていません。したがって、ユニコード・セッションでこの関数が正しく機能するように長さを指定する必要があります。	
PS 位置	該当フィールドを識別します。 SRCHALL に対しては、このパラメーターは該当フィールド内のどのバイトの PS 位置でも構いません。 SRCHFROM に対しては、このパラメーターは SRCHFRWD 検索の開始位置または SRCHBKWD の終了位置を示します。 3 (142 ページ) の使用上の注意を参照してください。	

戻りパラメーター: この関数はデータ・ストリング長および戻りコードを戻します。

データ・ストリング長:

以下のコードが定義されています。

長さ	説明
= 0	ストリングが検出されませんでした。
> 0	示されたホスト表示スペースの位置でストリングが検出されました。

戻りコード:

以下のコードが定義されています。

戻りコード	説明
0	**Search Field** 関数は正常終了しました。
1	ユーザー・プログラムが現在ホスト・セッションに接続されていません。
2	パラメーター・エラーです。ストリング長が 0 か、または EOT モードが指定されましたが、呼び出しデータ・ストリング内に EOT 文字がありません。
7	ホスト表示スペースの位置が正しくありません。
9	システム・エラーが発生しました。
24	検索ストリングが検出されなかったか、またはホストの表示スペースが定様式化されていません。

使用上の注意: 以下のオプションは **Search Field** のユニコード・セッションでサポートされ、DBCS の場合と同じ方法で機能します。

- STRLEN
- SRCHALL
- SRCHFROM
- SRCHFRWD
- SRCHBKWD

1137 コード・ページ・サポート

ユニコード機能は 5250 セッションでのみサポートされます。

STREOT オプションはユニコード・セッションではサポートされていません。詳細については、 165 ページの『Set Session Parameters (9)』を参照してください。

前提呼び出し: Connect Presentation Space (1)

呼び出しパラメーター:

	標準インターフェース	拡張インターフェース
関数番号	必ず 30 にしてください。	
データ・ストリング	検索対象のターゲット・ユニコード・ストリング。	
長さ	バイト単位でのターゲット・ユニコード・ストリングの長さ。 **注:** EOT モードはユニコード・セッションではサポートされていません。したがって、ユニコード・セッションでこの関数が正しく機能するように長さを指定する必要があります。	
PS 位置	該当フィールドを識別します。 SRCHALL に対しては、このパラメーターは該当フィールド内のどのバイトの PS 位置でも構いません。 SRCHFROM に対しては、このパラメーターは SRCHFRWD 検索の開始位置または SRCHBKWD の終了位置を示します。 3 (142 ページ) の使用上の注意を参照してください。	

戻りパラメーター: この関数はデータ・ストリング長および戻りコードを戻します。

データ・ストリング長:

以下のコードが定義されています。

長さ	説明
= 0	ストリングが検出されませんでした。
> 0	示されたホスト表示スペースの位置でストリングが検出されました。

戻りコード:

以下のコードが定義されています。

戻りコード	説明
0	**Search Field** 関数は正常終了しました。
1	ユーザー・プログラムが現在ホスト・セッションに接続されていません。
2	パラメーター・エラーです。ストリング長が 0 か、または EOT モードが指定されましたが、呼び出しデータ・ストリング内に EOT 文字がありません。
7	ホスト表示スペースの位置が正しくありません。
9	システム・エラーが発生しました。

戻りコード	説明
24	検索ストリングが検出されなかったか、またはホストの表示スペースが定様式化されていません。

使用上の注意: 以下のオプションは **Search Field** のユニコード・セッションでサポートされ、SBCS の場合と同じ方法で機能します。

- STRLEN
- SRCHALL
- SRCHFROM
- SRCHFRWD
- SRCHBKWD

Search Presentation Space (6)

3270	*5250*	*VT*
YES	YES	YES

Search Presentation Space 関数により、 EHLLAPI プログラムは指定されたストリングの存在についてホストの表示スペースをチェックします。

前提呼び出し

Connect Presentation Space (1)

呼び出しパラメーター

	標準インターフェース	拡張インターフェース
関数番号	必ず 6 にしてください。	
データ・ストリング	検索するデータ・ストリング。	
長さ	該当データ・ストリングの長さ。 EOT モードの場合、このパラメーターは一時変更されます。	
PS 位置	ホスト・プレゼンテーション内の、検索を開始する (SRCHFRWD オプション) 位置、または検索を終了する (SRCHBKWD オプション) 位置。SRCHALL (デフォルト) モードの場合、このパラメーターは一時変更されます。	

戻りパラメーター

この関数はデータ・ストリング長および戻りコードを戻します。

データ・ストリング長:

以下のコードが定義されています。

長さ	説明
= 0	ストリングが検出されませんでした。
> 0	示されたホスト表示スペースの位置でストリングが検出されました。

戻りコード:

以下のコードが定義されています。

戻りコード	説明
0	**Search Presentation Space** 関数は正常終了しました。
1	ユーザー・プログラムが現在ホスト・セッションに接続されていません。
2	パラメーター指定でエラーが発生しました。
7	ホスト表示スペースの位置が正しくありません。
9	システム・エラーが発生しました。
24	検索ストリングが検出されませんでした。

使用上の注意

1. **Set Session Parameters** (9) 関数での 4 つのパラメーター・セットがこの関数に関連しています。それらは SRCHALL/SRCHFROM、STRLEN/STREOT、SRCHFRWD/SRCHBKWD、および EOT=c の各セッション・オプションです。詳細については、 1 (167 ページ) 〜 4 (167 ページ) を参照してください。
2. **Set Session Parameters** (9) 関数を使用して、 SRCHBKWD を指定することができます。このオプションが有効である場合、検索操作はストリングの*最後* の出現位置を検出します。
3. **Search Presentation Space** 関数は、通常、全ホスト表示スペースをチェックします。しかし、**Set Session Parameters** (9) 関数を使用して SRCHFROM を指定することができます。このモードのとき、PS 位置パラメーターは検索の開始位置または検索の終了位置を指定します。
   - SRCHFRWD オプションが有効な場合、指示されたストリングの検索を、指定された PS 位置から開始し、ホスト表示スペースの終わりに向かって実行します。
   - SRCHBKWD オプションが有効な場合、指示されたストリングの検索を、表示スペースの終わりから開始し、指定された PS 位置に向かって逆方向に実行します。該当ストリングが検出されない時には、検索は呼び出し PS 位置パラメーターで指定された PS 位置で停止します。
4. ホスト表示スペース内に複数回出現する可能性のあるキーワードを検索する場合には SRCHFROM オプションも使用できます。
5. **Search Presentation Space** 関数は、ホスト表示スペースが使用可能かどうかの判別にも使用できます。ユーザーの EHLLAPI アプリケーション・プログラムがデータ送信の前に特定のプロンプトまたはメッセージを要求している場合、**Search Presentation Space** 関数を使用して、処理続行の前にプロンプトまたはメッセージをチェックすることができます。
6. **DBCS の場合のみ:** 指定した検索関数の開始位置が 2 バイト文字の第 2 バイトである場合、検索は SRCHFRWD のときは次の文字から、SRCHBKWD の時はその文字から開始されます。指定したストリングの最後の文字が 2 バイト文字の第 1 バイトである場合、その文字は検索対象となりません。

検索時には、表示スペース内では SO と SI の対は無視されます。 2 バイト制御文字を検索するには、その文字の前後に SO (X'0E') と SI (X'0F') を付けてください。例えば、データ・ストリング内の X'0E000C0F' は 2 バイト文字 FF (X'000C') として扱われます。

注: 5250 エミュレーションは、24 行 80 桁の表示スペースをサポートします。コミュニケーション・マネージャー 5250 エミュレーションが 25 番目の行を表示するインスタンスもあります。このようなことが起こるのは、ホストからのエラー・メッセージが表示される場合、またはオペレーターが SysReq キーを選択する場合です。パーソナル・コミュニケーションズは、25 行目の情報を 24 行目またはステータス・バーに表示します。ステータス・バーに情報が表示される場合は、ステータス・バーがその情報用に構成されている必要があります。ステータス・バーの構成方法の詳細については、「*はじめに*」を参照してください。**EXTEND_PS** オプションによって、EHLLAPI アプリケーションはコミュニケーション・マネージャー EHLLAPI と同じインターフェースを使用することができ、このような状況が発生したときに有効な表示スペースが拡張されます。

1390/1399 コード・ページ・サポート

ユニコード機能は 3270 および 5250 セッションでのみサポートされます。

STREOT オプションはユニコード・セッションではサポートされていません。詳細については、 165 ページの『Set Session Parameters (9)』を参照してください。

前提呼び出し: Connect Presentation Space (1)

呼び出しパラメーター:

	標準インターフェース	**拡張インターフェース**
関数番号	必ず 6 にしてください。	
データ・ストリング	検索対象のターゲット・ユニコード・ストリング。	
長さ	ユニコード文字でのターゲット・ユニコード・ストリングの長さ。 **注:** EOT モードはユニコード・セッションではサポートされていません。したがって、ユニコード・セッションでこの関数が正しく機能するように長さを指定する必要があります。	
PS 位置	ホスト・プレゼンテーション内の、検索を開始する (SRCHFRWD オプション) 位置、または検索を終了する (SRCHBKWD オプション) 位置。SRCHALL (デフォルト) モードの場合、このパラメーターは一時変更されます。	

戻りパラメーター: この関数はデータ・ストリング長および戻りコードを戻します。

データ・ストリング長:

以下のコードが定義されています。

長さ	**説明**
= 0	ストリングが検出されませんでした。

長さ	説明
> 0	示されたホスト表示スペースの位置でストリングが検出されました。

戻りコード:

以下のコードが定義されています。

戻りコード	説明
0	**Search Presentation Space** 関数は正常終了しました。
1	ユーザー・プログラムが現在ホスト・セッションに接続されていません。
2	パラメーター指定でエラーが発生しました。
7	ホスト表示スペースの位置が正しくありません。
9	システム・エラーが発生しました。
24	検索ストリングが検出されませんでした。

使用上の注意: 以下のオプションは **Search Presentation Space** (6) のユニコード・セッションでサポートされ、DBCS の場合と同じように機能します。

- STRLEN
- SRCHALL
- SRCHFROM
- SRCHFRWD
- SRCHBKWD

1137 コード・ページ・サポート

ユニコード機能は 5250 セッションでのみサポートされます。

STREOT オプションはユニコード・セッションではサポートされていません。詳細については、 165 ページの『Set Session Parameters (9)』を参照してください。

前提呼び出し: Connect Presentation Space (1)

呼び出しパラメーター:

	標準インターフェース	拡張インターフェース
関数番号	必ず 6 にしてください。	
データ・ストリング	検索対象のターゲット・ユニコード・ストリング。	
長さ	バイト単位でのターゲット・ユニコード・データ・ストリングの長さ。 **注:** EOT モードはユニコード・セッションではサポートされていません。したがって、ユニコード・セッションでこの関数が正しく機能するように長さを指定する必要があります。	
PS 位置	ホスト・プレゼンテーション内の、検索を開始する (SRCHFRWD オプション) 位置、または検索を終了する (SRCHBKWD オプション) 位置。SRCHALL (デフォルト) モードの場合、このパラメーターは一時変更されます。	

戻りパラメーター: この関数はデータ・ストリング長および戻りコードを戻します。

データ・ストリング長:

以下のコードが定義されています。

長さ	説明
= 0	ストリングが検出されませんでした。
> 0	示されたホスト表示スペースの位置でストリングが検出されました。

戻りコード:

以下のコードが定義されています。

戻りコード	説明
0	**Search Presentation Space** 関数は正常終了しました。
1	ユーザー・プログラムが現在ホスト・セッションに接続されていません。
2	パラメーター指定でエラーが発生しました。
7	ホスト表示スペースの位置が正しくありません。
9	システム・エラーが発生しました。
24	検索ストリングが検出されませんでした。

使用上の注意: 以下のオプションは **Search Presentation Space** (6) のユニコード・セッションでサポートされ、SBCS の場合と同じように機能します。

- STRLEN
- SRCHALL
- SRCHFROM
- SRCHFRWD
- SRCHBKWD

Send File (90)

3270	*5250*	*VT*
YES	YES	NO

Send File 関数は、EHLLAPI を実行しているワークステーションのセッションからホストのセッションへファイルを転送するときに使用します。

前提呼び出し

この関数には前提呼び出しはありません。

呼び出しパラメーター

	標準インターフェース	拡張インターフェース
関数番号	必ず 90 にしてください。	

	標準インターフェース	拡張インターフェース
データ・ストリング	以下の例を参照してください。	
長さ	該当データ・ストリングの長さ。 EOT モードの場合、このパラメーターは一時変更されます。	
PS 位置	必ず 0 にしてください。	

SBCS のデータ・ストリングの例を以下に示します。

3270 セッション

- VM/CMS ホスト・システムへファイルを送信する場合:

  pc_filename [*id:*]*fn ft* [*fm*] [(*option*]

- MVS/TSO ホスト・システムへファイルを送信する場合:

  pc_filename [*id:*]*dataset*[(*member*)] [*/password*] [*option*]

- CICS ホスト・システムへファイルを送信する場合:

  pc_filename [*id:*]*host_filename* [(*option*]

5250 セッション

- iSeries、eServer i5、または System i5 ホスト・システムへファイルを送信する場合:

  pc_filename [*id:*]*library file member* [*option*]

DBCS のデータ・ストリングの例を以下に示します。

3270 セッション

- VM/CMS ホスト・システムへファイルを送信する場合:

  pc_filename [*id:*]*fn ft* [*fm*] [(*option*]

- MVS/TSO ホスト・システムへファイルを送信する場合:

  pc_filename [*id:*]*dataset*[(*member*)] [*/password*]
  [(*option*]

- CICS ホスト・システムへファイルを送信する場合:

  pc_filename [*id:*]*host_filename* [(*option*]

5250 セッション

- iSeries、eServer i5、または System i5 ホスト・システムへファイルを送信する場合:

  pc_filename [*id:*]*library file member* [*option*]

注: [ ] で囲まれたパラメーターはオプションです。指定できるオプションは、次の表のとおりです。オプションについて詳しくは、「*管理者ガイドおよび解説書*」を参照してください。

ホスト・システム	共通オプション
VM/CMS	ASCII、ASCII、CRLF、APPEND、LRECL n、RECFM vlf、TIME n、CLEAR、NOCLEAR、PROGRESS、QUIET

ホスト・システム	共通オプション
MVS/TSO	ASCII、JISCII、CRLF、APPEND、LRECL (n)、RECFM (v|flu)、TIME (n)、CLEAR、NOCLEAR、PROGRESS、QUIET、BLKSIZE (n)、SPACE (n[,m])、AVBLOCK|TRACKS|CYLINDERS
CICS	ASCII、ASCII、CRLF、BINARY、TIME n、CLEAR、NOCLEAR、PROGRESS、QUIET
i5/OS または OS/400	ASCII、ASCII、CRLF、APPEND、SRC、LRECL n、TIME n、CLEAR、NOCLEAR、PROGRESS、QUIET

注:

ASCII は日本の DBCS セッションでのみ有効であり、ASCII は、その他すべての SBCS セッションと DBCS セッションで有効です。
注: 時刻を指定すると、その値が Set Session パラメーターの値に取って代わります。
注:

指定されたその他のオプションは、ホスト転送プログラムに渡されます。ホスト側のファイル転送プログラムは、これらを使用するか、無視するか、エラーを戻すかのいずれかです。ホスト転送プログラムのマニュアルで、サポートされるオプションの完全なリストを調べてください。

戻りパラメーター

戻りコード	説明
2	パラメーター・エラーが発生したか、または EHLLAPI バッファーに対して長すぎる (255 バイトを超過) データ・ストリング長が指定されました。ファイル転送が失敗しました。
3	ファイル転送は完了しました。
4	ファイル転送は完了し、レコードは分割されました。
5	ワークステーション・ファイル名が有効でなかったか、見つかりませんでした。ファイル転送が取り消されました。
9	システム・エラーが発生しました。
27	Cancel ボタンまたはタイムアウト (**Set Session Parameter** (9) 関数により設定) によってファイル転送は終了しました。
101	ファイル転送 (CICS への/CICS からの転送) は正常に終了しました。

システム内、またはユーザーのデータ・ストリングの指定方法に問題がある場合には、戻りコード 2 または 9 が戻されます。

他の戻りコードも受信される場合があります。これらの戻りコードは、ホスト転送プログラムにより生成されるメッセージ番号に関連しています。CICS ホスト転送プログラムに転送する場合、その戻りコードから 100 を減算するとメッセージの数値部分を得ることができます。例えば、戻りコードが 101 の場合には、ホストによりメッセージ番号 INW0001 が出されたことを示します。他のホスト転送プログラムの場合には、そのメッセージの数字部分のみ使用してください。例えば、戻りコードが 34 の場合には、ホストによりメッセージ番号 TRANS34 が出されたことを示します。特定のメッセージの意味については、ご使用のホスト転送プログラムの資料を参照してください。

EHLLAPI により報告されたオペレーティング・システムのエラー・コードは、300 より大きい数字です。エラー・コードを判別するには、300 を引いてから、オペレーティング・システムの資料でその戻りコードを調べてください。

使用上の注意

1. **Set Session Parameters** (9) 関数での 4 つのパラメーター・セットがこの関数に関連しています。それらは QUIET/NOQUIET、STRLEN/STREOT、TIMEOUT=c/TIMEOUT=0、および EOT=c の各セッション・オプションです。詳細については、 1 と、 2 (167 ページ)、 7 および 8 (168 ページ) の項目を参照してください。

Send Key (3)

3270	*5250*	*VT*
YES	YES	YES

Send Key 関数は、1 つのキー・ストロークまたはキー・ストロークのストリングをホスト表示スペースへ送信する際に使用します。

呼び出しデータ・ストリング・パラメーターで、送信するキー・ストロークのストリングを定義します。キー・ストロークは、端末オペレーターが入力したかのように、宛先セッションには見えます。 Enter などのアテンション ID (AID) キーもすべて送信できます。ホストの入力保護フィールドまたは数字専用フィールドはすべて、それに対応した取り扱いが必要です。

前提呼び出し

Connect Presentation Space (1)

呼び出しパラメーター

	標準インターフェース	**拡張インターフェース**
関数番号	必ず 3 にしてください。	
データ・ストリング	キー・ストロークのストリング。最大 255 バイト。英大文字と英小文字の ASCII 文字は、そのまま表されます。ファンクション・キーおよびシフト・ファンクション・キーは、略号によって表されます。 155 ページの『キーボードの略号』を参照してください。	
長さ	送信元データ・ストリングの長さ。 EOT モードの場合、このパラメーターは一時変更されます。	
PS 位置	NA	

戻りパラメーター

戻りコード	**説明**
0	キー・ストロークは送信され、状況は正常です。
1	ユーザー・プログラムが現在ホスト・セッションに接続されていません。

戻りコード	説明
2	無効なパラメーターが EHLLAPI に渡されました。
4	ホスト・セッションは使用中であったため、どのキー・ストロークも送信できませんでした。
5	送信先セッションへの入力が禁止または拒否されたため、キー・ストローク全部を送信することはできませんでした。
9	システム・エラーが発生しました。

使用上の注意

1. **Set Session Parameters** (9) 関数のパラメーターがこの関数に関連しています。それらは AUTORESET/NORESET、 STRLEN/STREOT、EOT=c、ESC=c、および RETRY/NORETRY の各セッション・オプションです。詳細については、 1 と 2 (167 ページ)、 9 と 10 (169 ページ)、および 19 (172 ページ) の項目を参照してください。
2. キーボードがロックされていたり、使用中の時は、キー・ストロークをホスト・セッションへ送信することはできません。 **Wait** (4) 関数でキーボードの状態をチェックすることができます。
3. ホストが使用中の場合、入力が拒否されることがあります。
4. データ・ストリングの長さは、デフォルトのデータ・ストリング長パラメーターを使用して明示的に定義する必要があります。しかし、**Set Session Parameters** (9) 関数の EOT=c オプションで暗黙的に定義することもできます。

   データ・ストリング長を明示的に定義する場合 (1 を参照)、アプリケーションが渡すデータ・ストリング長パラメーターの値を計算しなければなりません。この計算では、`@E` などの複合キー・ストロークは 2 バイト、 `@A@C` などの複合キー・ストロークは 4 バイトとします。
5. 特殊制御キーを送信するには、複合文字によるコーディング法を使用します。このコーディング法では、1 つのキー・ストロークを示すために 2 から 4 文字の ASCII 文字が使用されています。 1 番目と 3 番目の文字は常にエスケープ文字です。 2 番目と 4 番目の文字は常にキー・コードです。

   シーケンス `LOGON ABCDE` の後に Enter (改行) キーを押す場合は、ストリングを `LOGON ABCDE@E` とコーディングします。これらのキー・コードの完全なリストは、 155 ページの『キーボードの略号』にあります。

   この複合コーディング法により、複雑な 16 進キー・コードを使用せずに必要なすべてのキー・ストローク・コードの ASCII ストリング表現が可能になります。

   デフォルトのエスケープ文字は `@` です。エスケープ文字の値は、**Set Session Parameters** (9) 関数の ESC=c オプションを使用して、別の文字に変更することができます。
6. パフォーマンスを向上させる必要があるユーザーは、 **Send Key** (3) 関数でストロークを送信するよりも、 **Copy String to Field** (33) または **Copy String to Presentation Space** (15) 関数を使用してください。ただし、**Send Key** (3) 関数だけが特殊制御キーを送信することができます。

7. この関数のパフォーマンスを向上させるためには、 **Set Session Parameters (9)** のセッション・オプション 10 (169 ページ) (NORESET オプション) を参照してください。

   NORESET を指定しない場合には、リセットの略号がキー・ストローク・ストリングに接頭部として追加されます。したがって、入力禁止以外のリセット可能な状況はすべてリセットされます。

   NORESET オプションは **Reset System** (21) 関数と同じではありません。

8. AID キーを含むキー・ストローク・ストリングは複数のパスを介してホストに送信されます。それぞれのパスは、最初の AID キー前までの (またはそれを含む) ストリングを送信します。 EHLLAPI は、ストリング長および各パスの開始位置を調整します。ホストのアプリケーション・プログラムによっては、任意のキー・ストロークが AID キー処理のために失われることがあります。したがって、複数の AID キーを含むキー・ストロークのリストを送信することはお勧めできません。

9. `@P` (印刷) または `@A@T` (表示スペースの印刷) 処理中、その表示スペースを更新する要求はすべて拒否されます。印刷要求の最中に表示スペースが使用中状態だったり、割り込み要求が発生した場合には、略号 @A@R (装置リセット - 表示スペースの印刷の取り消し) が要求を取り消し、状況をリセットします。

キーボードの略号

キーボードの略号はワークステーションのキーボードの特殊ファンクション・キーを表す ASCII 文字を提供します。この省略形コードにより、これらの特殊キーが覚えやすくなります。最もよく使用されるキーは英字キー・コードです。例えば、**Clear** キーは *C*、**Tab** キーは *T* などです。

表 7 は英大文字を使用する略号の一覧です。

表 7. 英大文字による略号

略号	意味	3270	5250	VT
@B	左タブ	YES	YES	NO
@C	消去	YES	YES	NO
@D	削除	YES	YES	NO
@E	実行 (Enter)	YES	YES	NO
@F	EOF 消去	YES	YES	NO
@H	ヘルプ	NO	YES	NO
@I	挿入	YES	YES	NO
@J	ジャンプ (フォーカスを設定)	YES	YES	NO
@L	カーソル左移動	YES	YES	YES
@N	改行	YES	YES	YES
@O	スペース	YES	YES	YES
@P	ページ印刷	YES	YES	YES
@R	リセット	YES	YES	NO
@T	右タブ	YES	YES	YES

表 7. 英大文字による略号 (続き)

略号	意味	3270	5250	VT
@U	カーソル上移動	YES	YES	YES
@V	カーソル下移動	YES	YES	YES
@X*	DBCS (予約済み)	YES	YES	NO
@Z	カーソル右移動	YES	YES	YES

表 8 は数字または英小文字を使用する略号の一覧です。

表 8. 数字または英小文字による略号

略号	意味	3270	5250	VT
@0	カーソル・ホーム	YES	YES	NO
@1	PF1/F1	YES	YES	NO
@2	PF2/F2	YES	YES	NO
@3	PF3/F3	YES	YES	NO
@4	PF4/F4	YES	YES	NO
@5	PF5/F5	YES	YES	NO
@6	PF6/F6	YES	YES	YES
@7	PF7/F7	YES	YES	YES
@8	PF8/F8	YES	YES	YES
@9	PF9/F9	YES	YES	YES
@a	PF10/F10	YES	YES	YES
@b	PF11/F11	YES	YES	YES
@c	PF12/F12	YES	YES	YES
@d	PF13	YES	YES	YES
@e	PF14	YES	YES	YES
@f	PF15	YES	YES	YES
@g	PF16	YES	YES	YES
@h	PF17	YES	YES	YES
@i	PF18	YES	YES	YES
@j	PF19	YES	YES	YES
@k	PF20	YES	YES	YES
@l	PF21	YES	YES	NO
@m	PF22	YES	YES	NO
@n	PF23	YES	YES	NO
@o	PF24	YES	YES	NO
@q	End	YES	YES	NO
@u	前ページ	NO	YES	NO
@v	次ページ	NO	YES	NO
@x	PA1	YES	YES	NO
@y	PA2	YES	YES	NO

表 8. 数字または英小文字による略号 (続き)

略号	意味	3270	5250	VT
@z	PA3	YES	YES	NO

表 9 は @A と @ 英大文字 (A〜Z) キーの組み合わせを使用する略号の一覧です。

表 9. @A と @ 英大文字による略号

略号	意味	3270	5250	VT
@A@C	テスト	NO	YES	NO
@A@D	ワード削除	YES	YES	NO
@A@E	フィールド終了	YES	YES	NO
@A@F	入力消去	YES	YES	NO
@A@H	システム要求	YES	YES	NO
@A@I	挿入切り替え	YES	YES	NO
@A@J	カーソル選択	YES	YES	NO
@A@L	高速カーソル左移動	YES	YES	NO
@A@Q	アテンション	YES	YES	NO
@A@R	装置の取り消し (表示スペース印刷取り消し)	YES	YES	NO
@A@T	表示スペース印刷	YES	YES	YES
@A@U	高速カーソル上移動	YES	YES	NO
@A@V	高速カーソル下移動	YES	YES	NO
@A@Z	高速カーソル右移動	YES	YES	NO

表 10 は、@A と @ 数字、または @A と @ 英小文字 (a〜z) キーの組み合わせを使用する略号の一覧です。

表 10. @A と @ 英小文字による略号

略号	意味	3270	5250	VT
@A@9	反転表示	YES	YES	NO
@A@b	下線	YES	NO	NO
@A@c	反転表示のリセット	YES	NO	NO
@A@d	赤	YES	NO	NO
@A@e	ピンク	YES	NO	NO
@A@f	緑	YES	NO	NO
@A@g	黄	YES	NO	NO
@A@h	青	YES	NO	NO

表 10. @A と @ 英小文字による略号 (続き)

略号	意味	3270	5250	VT
@A@i	空色	YES	NO	NO
@A@j	白	YES	NO	NO
@A@l	ホスト・カラーのリセット	YES	NO	NO
@A@t	印刷 (ワークステーション)	YES	YES	NO
@A@y	正方向ワード・タブ	YES	YES	NO
@A@z	逆方向ワード・タブ	YES	YES	NO

表 11 は @A と @ 特殊文字の組み合わせを使用する略号の一覧です。

表 11. @A と @ 特殊文字による略号

略号	意味	3270	5250	VT
@A@-	フィールド -	NO	YES	NO
@A@+	フィールド +	NO	YES	NO
@A@<	レコード・バックスペース	NO	YES	NO

表 12 は @S と @ 英文字の組み合わせを使用する略号の一覧です。

表 12. @S (シフト) と @ 英文字による略号

略号	意味	3270	5250	VT
@S@E	ホストでの表示スペース印刷	NO	YES	NO
@S@x	複写 (DUP)	YES	YES	NO
@S@y	フィールド・マーク	YES	YES	NO

DBCS の場合のみ: 表 13 は @X と @ 数字または @X と @ 英小文字 (a〜z) の組み合わせを使用する略号の一覧です。

表 13. @X と @ 英小文字による略号 (DBCS の場合のみ)

略号	意味	3270	5250	VT
@X@1	SO/SI 表示	YES	YES	NO
@X@5	SO/SI 生成	NO	YES	NO
@X@6	属性表示	NO	YES	NO
@X@7	文字前進	NO	YES	NO
@X@c	縦破線	NO	YES	NO

VT の場合のみ: 表 14 は、 @M と @ 数字または @M と @ 英小文字 (a ～ z) の組み合わせを使用する略号です。

表 14. @M、@Q、および @ 英小文字を使用した略号 (VT の場合のみ)

略号	意味	3270	5250	VT
@M@0	VT テン・キー 0	NO	NO	YES
@M@1	VT テン・キー 1	NO	NO	YES
@M@2	VT テン・キー 2	NO	NO	YES
@M@3	VT テン・キー 3	NO	NO	YES
@M@4	VT テン・キー 4	NO	NO	YES
@M@5	VT テン・キー 5	NO	NO	YES
@M@6	VT テン・キー 6	NO	NO	YES
@M@7	VT テン・キー 7	NO	NO	YES
@M@8	VT テン・キー 8	NO	NO	YES
@M@9	VT テン・キー 9	NO	NO	YES
@M@-	VT テン・キー - (負符号)	NO	NO	YES
@M@,	VT テン・キー , (コンマ)	NO	NO	YES
@M@.	VT テン・キー . (ピリオド)	NO	NO	YES
@M@e	VT テン・キー Enter	NO	NO	YES
@M@f	VT 編集 検索	NO	NO	YES
@M@i	VT 編集 挿入	NO	NO	YES
@M@r	VT 編集 除去	NO	NO	YES
@M@s	VT 編集 選択	NO	NO	YES
@M@p	VT 編集 前の画面	NO	NO	YES
@M@n	VT 編集 次の画面	NO	NO	YES
@M@a	VT PF1	NO	NO	YES
@M@b	VT PF2	NO	NO	YES
@M@c	VT PF3	NO	NO	YES
@M@d	VT PF4	NO	NO	YES
@M@h	VT 保留画面	NO	NO	YES

表 14. @M、@Q、および @ 英小文字を使用した略号 (VT の場合のみ) (続き)

略号	意味	3270	5250	VT
@M@(スペース)	制御コード NUL	NO	NO	YES
@M@A	制御コード SOH	NO	NO	YES
@M@B	制御コード STX	NO	NO	YES
@M@C	制御コード ETX	NO	NO	YES
@M@D	制御コード EOT	NO	NO	YES
@M@E	制御コード ENQ	NO	NO	YES
@M@F	制御コード ACK	NO	NO	YES
@M@G	制御コード BEL	NO	NO	YES
@M@H	制御コード BS	NO	NO	YES
@M@I	制御コード HT	NO	NO	YES
@M@J	制御コード LF	NO	NO	YES
@M@K	制御コード VT	NO	NO	YES
@M@L	制御コード FF	NO	NO	YES
@M@M	制御コード CR	NO	NO	YES
@M@N	制御コード SO	NO	NO	YES
@M@O	制御コード SI	NO	NO	YES
@M@P	制御コード DLE	NO	NO	YES
@M@Q	制御コード DC1	NO	NO	YES
@M@R	制御コード DC2	NO	NO	YES
@M@S	制御コード DC3	NO	NO	YES
@M@T	制御コード DC4	NO	NO	YES
@M@U	制御コード NAK	NO	NO	YES
@M@V	制御コード SYN	NO	NO	YES
@M@W	制御コード ETB	NO	NO	YES
@M@X	制御コード CAN	NO	NO	YES
@M@Y	制御コード EM	NO	NO	YES
@M@Z	制御コード SUB	NO	NO	YES
@M@u	制御コード ESC	NO	NO	YES
@M@v	制御コード FS	NO	NO	YES
@M@w	制御コード GS	NO	NO	YES
@M@x	制御コード RS	NO	NO	YES
@M@y	制御コード US	NO	NO	YES
@M@z	制御コード DEL	NO	NO	YES
@Q@A	VT ユーザー定義キー 6	NO	NO	YES
@Q@B	VT ユーザー定義キー 7	NO	NO	YES
@Q@C	VT ユーザー定義キー 8	NO	NO	YES

表 14. @M、@Q、および @ 英小文字を使用した略号 (VT の場合のみ) (続き)

略号	意味	3270	5250	VT
@Q@D	VT ユーザー定義キー 9	NO	NO	YES
@Q@E	VT ユーザー定義キー 10	NO	NO	YES
@Q@F	VT ユーザー定義キー 11	NO	NO	YES
@Q@G	VT ユーザー定義キー 12	NO	NO	YES
@Q@H	VT ユーザー定義キー 13	NO	NO	YES
@Q@I	VT ユーザー定義キー 14	NO	NO	YES
@Q@J	VT ユーザー定義キー 15	NO	NO	YES
@Q@K	VT ユーザー定義キー 16	NO	NO	YES
@Q@L	VT ユーザー定義キー 17	NO	NO	YES
@Q@M	VT ユーザー定義キー 18	NO	NO	YES
@Q@N	VT ユーザー定義キー 19	NO	NO	YES
@Q@0	VT ユーザー定義キー 20	NO	NO	YES
@Q@a	VT 後退タブ	NO	NO	YES
@Q@r	VT ページ消去	NO	NO	YES
@Q@s	VT 編集	NO	NO	YES

次の表は特殊文字を使用する略号の一覧です。

表 15. 特殊文字キーによる略号

略号	意味	3270	5250	VT
@@	@	YES	YES	YES
@$	カーソル切り替え (プレゼンテーション・マネージャー・インターフェースのみ)	YES	YES	YES
@<	バックスペース	YES	YES	YES

次の表は BIDI キーの略号の一覧です。

表 16. BIDI キーの略号

略号	意味	3270	5250	VT
@:@s	画面強調反転	YES	YES	YES
@:@n	BIDI 層	YES	YES	YES
@:@l	ローマ字層	YES	YES	YES
@:@F	フィールド強調反転	YES	YES	NO
@:@p	プッシュ	YES	NO	NO
@:@e	プッシュ終了	YES	NO	NO
@:@a	自動プッシュ	YES	NO	NO
@:@r	自動強調反転	YES	NO	NO
@:@d	CSD	YES	NO	NO
@:@f	最終	YES	NO	NO
@:@i	分離	YES	NO	NO
@:@m	中央	YES	NO	NO
@:@t	初期	YES	NO	NO
@:@h	フィールドの形状	YES	NO	NO
@:@u	フィールドの基本	YES	NO	NO
@:@b	基本	NO	YES	NO
@:@o	クローズ	NO	YES	NO
@:@K	列見出し	NO	NO	YES
@:@B	カーソルの方向	NO	NO	YES
@:@D	エンコード・モード	NO	NO	YES
@:@M	VT 変更表示モード	NO	NO	Yes (ヘブライ語のみ)

次の文字キーは表示どおりに解釈されます。

a ～ z	!	’	’	<	}
A ～ Z	$	(	.	>	[
0 ～ 9	%	)	/	=	]
~	&	*	:	?	|
#	″	+	;	{	

1390/1399 コード・ページ・サポート

ユニコード機能は 3270 および 5250 セッションでのみサポートされます。

STREOT オプションはユニコード・セッションではサポートされていません。詳細については、 165 ページの『Set Session Parameters (9)』を参照してください。

セッション・オプション ESC はユニコード・セッションではサポートされていません。このオプションを使用してユニコード文字を ESC 文字として設定すること

はできません。デフォルトの ESC 文字 @ をユニコード・セッションで使用してください。詳細については、 165 ページの『Set Session Parameters (9)』を参照してください。

前提呼び出し: Connect Presentation Space (1)

呼び出しパラメーター:

	標準インターフェース	**拡張インターフェース**
関数番号	必ず 3 にしてください。	
データ・ストリング	キー・ストロークのユニコード・ストリング。最大 255 バイト。英大文字と英小文字の ASCII 文字は、そのまま表されます。ファンクション・キーおよびシフト・ファンクション・キーは、略号によって表されます。 155 ページの『キーボードの略号』を参照してください。	
長さ	ユニコード文字のユニコード・ストリングの長さ。	
PS 位置	NA	

戻りパラメーター:

戻りコード	**説明**
0	キー・ストロークは送信され、状況は正常です。
1	ユーザー・プログラムが現在ホスト・セッションに接続されていません。
2	無効なパラメーターが EHLLAPI に渡されました。
4	ホスト・セッションは使用中であったため、どのキー・ストロークも送信できませんでした。
5	送信先セッションへの入力が禁止または拒否されたため、キー・ストローク全部を送信することはできませんでした。
9	システム・エラーが発生しました。

使用上の注意: キー・ストロークを PCOMM セッションに送信する前に、そのセッションがユニコード・セッションであり、現行のプラットフォームが Windows 2000 であることを確認してください。。

ストリングの長さはユニコード文字の数を示すもので、送信される ANSI 文字の数ではありません。

1137 コード・ページ・サポート

ユニコード機能は 5250 セッションでのみサポートされます。

STREOT オプションはユニコード・セッションではサポートされていません。詳細については、 165 ページの『Set Session Parameters (9)』を参照してください。

セッション・オプション ESC はユニコード・セッションではサポートされていません。このオプションを使用してユニコード文字を ESC 文字として設定することはできません。デフォルトの ESC 文字 @ をユニコード・セッションで使用してください。詳細については、 165 ページの『Set Session Parameters (9)』を参照してください。

前提呼び出し: Connect Presentation Space (1)

呼び出しパラメーター:

	標準インターフェース	拡張インターフェース
関数番号	必ず 3 にしてください。	
データ・ストリング	キー・ストロークのユニコード・ストリング。最大 255 バイト。英大文字と英小文字の ASCII 文字は、そのまま表されます。ファンクション・キーおよびシフト・ファンクション・キーは、略号によって表されます。 155 ページの『キーボードの略号』を参照してください。	
長さ	バイト単位でのユニコード・データ・ストリングの長さ。長さが 2 の倍数でない場合は、2 のエラー・コードが戻されます。	
PS 位置	NA	

戻りパラメーター:

戻りコード	説明
0	キー・ストロークは送信され、状況は正常です。
1	ユーザー・プログラムが現在ホスト・セッションに接続されていません。
2	無効なパラメーターが EHLLAPI に渡されました。
4	ホスト・セッションは使用中であったため、どのキー・ストロークも送信できませんでした。
5	送信先セッションへの入力が禁止または拒否されたため、キー・ストローク全部を送信することはできませんでした。
9	システム・エラーが発生しました。

使用上の注意: キー・ストロークを PCOMM セッションに送信する前に、そのセッションがユニコード・セッションであることを確認してください。セッションが ANSI で、ユニコード・ストリングが送信されると、化けた文字が表示されます。

ストリングの長さはバイト数を示すもので、送信されるユニコード文字の数を示すものではありません。そのため、長さは 2 の倍数でなければなりません。そうでない場合は、関数によって、パラメーター・エラーが戻されます。

Set Cursor (40)

3270	*5250*	*VT*
YES	YES	YES

Set Cursor 関数は、ホスト表示スペース内のカーソルの位置を設定する際に使用します。 **Set Cursor** 関数を使用する前に、ワークステーションのアプリケーションがホストの表示スペースに接続されていることが必要です。

前提呼び出し

Connect Presentation Space (1)

呼び出しパラメーター

	標準インターフェース	拡張インターフェース
関数番号	必ず 40 にしてください。	
データ・ストリング	NA	
長さ	NA	
PS 位置	接続したホスト表示スペース内の、配置したいカーソルの位置	

戻りパラメーター

戻りコード	説明
0	カーソルは指定された位置に配置されました。
1	ユーザー・プログラムが現在ホスト・セッションに接続されていません。
4	セッションは使用中です。
7	1 より小さい、または接続されているホスト表示スペース・サイズを超えるカーソル位置が指定されました。
9	システム・エラーが発生しました。

使用上の注意

DBCS の場合のみ: 指定したカーソルが 2 バイト文字の第 2 バイトである場合、カーソルはその文字の第 1 バイトへ移動し、エラー・コードは戻されません。

1137 コード・ページ・サポート

ユニコード・セッションでの **Set Cursor** の使用法は、以下の点を除いて SBCS セッションと同じです。

- ユニコード機能は 5250 セッションでのみサポートされます。
- ユニコード・セッション内でのみ、指定されたカーソルがクラスター (例えば、ヒンディ語言語クラスター) の中央にあれば、それからそのカーソルは自動的にクラスターの先頭に位置付けられます。

Set Session Parameters (9)

3270	*5250*	*VT*
YES	YES	YES

Set Session Parameters 関数を使用して、 EHLLAPI のすべてのセッションに関して特定のデフォルト・セッション・オプションを変更することができます。EHLLAPI がロードされると、セッション・オプションのデフォルトの設定値は 167 ページの『セッション・オプション』に示す表で、下線を引いた項目のようになります。これらの値のいずれか、またはすべてを、以下で説明するように、呼び出しデータ・ストリング内に必要なオプションを入れて変更することができます。指定した設定値は次のいずれかが発生するまで有効です。

- 次の **Set Session Parameters** (9) 関数で新しい値を指定したために変更された。

- **Reset System** (21) 関数が実行される。
- EHLLAPI アプリケーション・プログラムが終了する。

以下にセッション・オプションの影響を受ける EHLLAPI 関数の一覧表を示します。この表に記載されていない関数は、どのセッション・オプションからも影響を受けません。それぞれの関数に影響を及ぼすセッション・オプションは『参照個所』の欄の対応する項目に示されています。これらの項目は『呼び出しパラメーター』以降の一覧表に示されています。

関数番号	関数名	参照個所
1	**Connect Presentation Space**	**11, 23, 24**
3	**Send Key**	**1, 2, 9, 10, 19**
4	**Wait**	**12**
5	**Copy Presentation Space**	**5, 13, 14, 15, 17, 20, 21, 22**
6	**Search Presentation Space**	**1, 2, 3, 4**
8	**Copy Presentation Space to String**	**5, 13, 14, 15, 17, 20, 21, 22**
10	**Query Sessions**	**16, 22**
15	**Copy String to Presentation Space**	**1, 2, 13, 14, 18, 20, 21, 22**
18	**Pause**	**6**
30	**Search Field**	**1, 2, 3, 4, 22**
33	**Copy String to Field**	**1, 2, 13, 14, 18, 20, 21, 22**
34	**Copy Field to String**	**5, 13, 14, 17, 20, 21, 22**
51	**Get Key**	**9, 12**
90	**Send File**	**1, 2, 7, 8**
91	**Receive File**	**1, 2, 7, 8**
101	**Connect Window Services**	**23, 24**
注: この表の項目 20 と 21 は DBCS の場合のみ		

前提呼び出し

この関数には前提呼び出しはありません。

呼び出しパラメーター

	標準インターフェース	拡張インターフェース
関数番号	必ず 9 にしてください。	
データ・ストリング	変更しようとするセッション・オプションの必要な値を含むストリング。データ・ストリングには、 167 ページの『セッション・オプション』に記載されているどの値でも入れることができます。値は、データ・ストリング行にコンマまたはブランクで区切って入れます。パラメーター・セットについては、それらが影響を及ぼす関数の項で説明しています。	
長さ	元のデータ・ストリングの明示の長さ (STREOT オプションは使用できません)。	
PS 位置	NA	

セッション・オプション

以下の表にセッション・オプションを示します。デフォルト値には下線が付いています。

1. 次の表の値は、**Send Key** (3)、 **Search Presentation Space** (6)、 **Copy String to Presentation Space** (15)、 **Search Field** (30)、 **Copy String to Field** (33)、 **Send File** (90)、および **Receive File** (91) の関数に対してどのようにデータ・ストリングの長さを定義するかを判別します。

値	説明
STRLEN	すべてのストリングに対して明示の長さが渡されています。
STREOT	ストリングの長さは明示的に指定されていません。呼び出し元のデータ・ストリングは EOT 文字で終了します。

2. 次の表の値は、EHLLAPI 関数である **Send Key** (3)、 **Search Presentation Space** (6)、 **Copy String to Presentation Space** (15)、 **Search Field** (30)、 **Copy String to Field** (33)、 **Send File** (90)、および **Receive File** (91) に対して呼び出し元のデータ・ストリングのテキスト終了 (EOT) 区切り文字として使用する文字を指定する際に使用します。

値	説明
EOT=c	この値で、ストリング終了文字に EOT 文字を (STREOT モードで) 指定できます。 2 進数の 0 がデフォルト値です。等号の後をブランクにしないでください。

   有効値にするためには、1 バイトのリテラル文字ストリングとして、前にブランクを付けずに c を入力します。このステートメントで指定した EOT 文字は、STREOT オプション (セッション・オプションの 1 を参照) が有効な場合に限り、呼び出しデータ・ストリングの長さの判別に使用されます。

3. 次の表の値は、検索関数の **Search Presentation Space** (6) および **Search Field** (30) に影響を及ぼします。

値	説明
SRCHALL	**Search Presentation Space** (6) 関数と **Search Field** (30) 関数は、ホスト表示スペース全体またはフィールド全体を走査します。
SRCHFROM	**Search Presentation Space** (6) 関数と **Search Field** (30) 関数は、指定した PS 位置から開始 (SRCHFRWD の時) するか、または指定した PS 位置で終了 (SRCHBKWD の時) します。

4. 次の表の値は、検索関数の **Search Presentation Space** (6) および **Search Field** (30) に影響を及ぼします。この値により、検索方向が判別されます。

値	説明
SRCHFRWD	**Search Presentation Space** (6) 関数と **Search Field** (30) 関数は、昇順方向に実行されます。
SRCHBKWD	**Search Presentation Space** (6) 関数と **Search Field** (30) 関数は、降順方向に実行されます。検索ストリングの先頭文字が指定した範囲内にあるとき、検索の対象となります。

5. 次の表の値は、**Copy Presentation Space** (5)、**Copy Presentation Space to String** (8)、**Copy Field to String** (34) の関数について、属性バイトをどのように取り扱うかを決定します。

値	説明
NOATTRB	不明な値をすべてブランクに変換します。
ATTRB	元の値として ASCII に相当しないコードをすべて戻します。
NULLATTRB	フィールド属性をすべて NULL 文字に変換します。

6. 次の表の値は、**Pause** (18) 関数に影響を及ぼします。

値	説明
FPAUSE	全期間休止。 **Pause** (18) 関数で指定した期間、休止します。
IPAUSE	割り込み可能な休止。 **Start Host Notification** (23) 関数の実行後、ホストのイベントが休止を終了させます。

7. 次の表の値は、ファイル転送関数の **Send File** (90) および **Receive File** (91) が生成するメッセージを表示するかどうかを判別します。

値	説明
NOQUIET	SEND および RECEIVE メッセージが表示されます。
QUIET	SEND および RECEIVE メッセージは表示されません。

8. 次の表のステートメントは、ファイル転送関数の **Send File** (90) および **Receive File** (91) の実行中にパーソナル・コミュニケーションズ EHLLAPI がどれくらい待機してから Cancel を自動的に発行するかを判別します。有効値にするために、c をアラビア数字の 0〜9 または英文字の J〜N にして、前にブランクを入れないでください。

値	説明
TIMEOUT=0	約 20 秒の遅延の後に Cancel が自動的に発行されます。

値	説明
TIMEOUT=c	指定した遅延の後に Cancel が自動的に発行されます。下の表にある 1 文字の標識は、Cancel を発行する前に、30 秒サイクルを何回受け入れる必要があるかを パーソナル・コミュニケーションズ に通知します。 **文字** **値 (分)** **1** 0.5 **2** 1.0 **3** 1.5 **4** 2.0 **5** 2.5 **6** 3.0 **7** 3.5 **8** 4.0 **9** 4.5 **J** 5.0 **K** 5.5 **L** 6.0 **M** 6.5 **N** 7.0

9. 次の表のステートメントは、キー・ストローク略号に対するエスケープ文字を定義するために使用します。このセッション・オプションは、**Send Key** (3) 関数と **Get Key** (51) 関数に影響を及ぼします。有効値にするためには、1 バイトのリテラル文字ストリングとして、前にブランクを付けずに c を入力します。

値	説明
ESC=c	この値により、キー・ストローク略号に対するエスケープ文字を指定できます (@ がデフォルト値)。等号の後をブランクにしないでください。ブランクはエスケープ文字に使用できません。

10. 次の表の値は、**Send Key** (3) 関数を使用して送信されたストリングを EHLLAPI が先に自動的にリセットするかどうかを判別します。

値	説明
AUTORESET	EHLLAPI は、リセット付きの **Send Key** (3) 関数を使用して送信されたキーの全ストリングに接頭部を付けることにより、すべての使用禁止条件のリセットを試行します。
NORESET	AUTORESET を行いません。

11. 次の表の値は、 **Connect Presentation Space** (1) コマンド関数に影響を及ぼします。

値	説明
CONLOG	ワークステーション・セッションとホスト・セッションの間の論理接続を確立します。接続中は、要求された表示スペースへジャンプしません。
CONPHYS	ワークステーション・セッションとホスト・セッションの間の物理接続を確立します。接続中に、要求された表示スペースへジャンプします。

12. 次の表の値は、 **Wait** (4) 関数と **Get Key** (51) 関数に影響を及ぼします。それぞれの値については、2 つの異なる影響があり、それぞれの関数に対応しています。

値	説明
TWAIT	**Wait** (4) 関数は、XCLOCK (X []) または XSYSTEM でタイムアウトになるまで最大 1 分間待機します。 **Get Key** (51) 関数は、EHLLAPI アプリケーション・プログラムがキーを代行受信してから、そのアプリケーションに制御を戻します (このキーは、通常のキーまたは **Start Keystroke Intercept** (50) 関数で指定されたオプションに基づく AID キーです)。
LWAIT	**Wait** (4) 関数は、 XCLOCK (X [])/XSYSTEM がクリアされるまで待機します。ホストが使用可能になるまでユーザーのアプリケーションに制御が戻らないため、このオプションはお勧めできません。 **Get Key** (51) 関数は、EHLLAPI アプリケーション・プログラムがキーを代行受信してから、そのアプリケーションに制御を戻します (このキーは、通常のキーまたは **Start Keystroke Intercept** (50) 関数で指定されたオプションに基づく AID キーです)。
NWAIT	**Wait** (4) 関数は、状況をチェックし、即座に状況を戻します (待機なし)。 **Get Key** (51) 関数は、 **Start Keystroke Intercept** (50) 関数で指定されたオプションに一致したオプションがキューにないと、4 番目のパラメーターに戻りコード 25 (キー・ストローク使用不可) を戻します。

注: NWAIT を使用するようお勧めします。

13. 次の表の値は、**Copy Presentation Space** (5)、 **Copy Presentation Space to String** (8)、 **Copy String to Presentation Space** (15)、**Copy String to Field** (33) および **Copy Field to String** (34) に影響を及ぼします。 拡張属性バイト (EAB) には、拡張文字属性および拡張フィールド属性が含まれます。

値	説明
NOEAB	データのみを渡します (EAB なし)。

値	説明
EAB	拡張属性バイトを指定して表示スペース・データを渡します。画面に表示されるそれぞれの文字については、2 バイトのデータが渡されます。このため、表示スペースの 2 倍のサイズのバッファーを事前割り振りしておくことが必要です。例えば、2 x 1920 = 3840 (24 行 80 桁の表示スペース用)。 文字ストリングの拡張属性は、フィールド内の個々の文字の属性としてではなく、フィールド・バイトの属性として報告される場合があります。このような場合、画面上の個々の文字または文字セットに下線が付けられるかどうかを知るには、フィールド属性バイト (画面に表示されるフィールドの前にあるバイト) の位置を指定して CopyPStoString を実行し、そのフィールド内の全文字に適用される EAB 情報を得てください。

注: EHLLAPI Copy PS to String を使用すると、オペレーターには見えないようにすべきテキストがコピーされます。隠しデータがあるかどうかを判別するには、EHLLAPI Set Session Parameters 関数を使用して、NODISPLAY オプションを設定してください。これにより、EHLLAPI は非表示フィールドを NULL として戻します。データを隠すための一般的な別の手順は、前景と背景の色を同一色 (例えば、黒色) に設定して、テキストは表示されるが人間のオペレーターには見えないようにすることです。アプリケーションにこれを検出させる唯一の方法は、 EAB と XLATE のセッション・パラメーターを使用して PS をコピーすることです。各バイト位置の前景/背景の色が戻され、見えない文字を判別することができます。

14. 次の表の値は、**Copy Presentation Space** (5)、 **Copy Presentation Space to String** (8)、 **Copy String to Presentation Space** (15)、**Copy String to Field** (33) および **Copy Field to String** (34) に影響を及ぼします。

値	説明
NOXLATE	EAB は変換されません。
XLATE	EAB は PC のカラー・グラフィックス・アダプター (CGA) 形式に変換されます。

15. 次の表の値は、NOATTRB と NOEAB が指定されている場合、**Copy Presentation Space** (5) および **Copy Presentation Space to String** (8) に影響を及ぼします。

値	説明
BLANK	不明な値をすべて X'20' に変換します。
NOBLANK	不明な値をすべて X'00' に変換します。

デフォルト値は BLANK です。デフォルト値を NOBLANK に変更したい場合は、パーソナル・コミュニケーションズのユーザー・クラス・アプリケーション・データのディレクトリーに配置されている PCSWIN.INI ファイルに、以下のステートメントを追加してください。

```
[API]
NullToBlank=NO
```

16. 次の表の値は、**Query Sessions** (10) により戻される表示スペースのサイズに影響を及ぼします。

値	説明
CFGSIZE	接続されている表示スペースの構成サイズを戻します。このオプションは、ホストによる構成サイズの一時変更を無視します。
NOCFGSIZE	接続されている表示スペースの現在のサイズを戻します。

17. 次の表の値は、**Copy Presentation Space** (5)、**Copy Presentation Space to String** (8)、および **Copy Field to String** (34) に影響を及ぼします。

値	説明
DISPLAY	表示スペース内の非表示フィールドを、コピー先のバッファー域にコピーします。現行のアプリケーションは、通常通りに機能します。
NODISPLAY	表示スペース内の非表示フィールドを、表示フィールドと同じ方法でコピー先のバッファー域にコピーしません。非表示フィールドは、コピー先のバッファーに NULL 文字のストリングとしてコピーされます。これにより、アプリケーションでは、パスワードなどの機密情報を表示せずに、コピーしたバッファーをプレゼンテーション・ウィンドウに表示することができます。

18. 次の表の値は、**Copy String to Presentation Space** (15) および **Copy String to Field** (33) に影響を及ぼします。

値	説明
NOPUTEAB	**Copy String to Presentation Space** または **Copy String to Field** のデータ・ストリングに EAB (DBCS の場合は EAD) を含めません。
PUTEAB	**Copy String to Presentation Space** または **Copy String to Field** のデータ・ストリングに EAB を含めます。

このオプションは、コミュニケーション・マネージャー/2 と互換性を持たせるためのものです。コミュニケーション・マネージャー/2 では、**Set Session Parameters** の EAB (または EAD) が有効なときは、**Copy String to Presentation Space** または **Copy String to Field** で指定するデータ・ストリングに、EAB (または EAD) が文字データと共に含まれていなければなりません。一方、従来のパーソナル・コミュニケーションズでは、EAB (または EAD) が有効であっても、これらの関数に指定するデータ・ストリングに含まれるのは文字データのみでした。しかし、パーソナル・コミュニケーションズでは、PUTEAB を設定して、データ・ストリングの中に EAB (または EAD) を含めることにより、コミュニケーション・マネージャー/2との互換性を持つことができます。

19. 次の表の値は、**Send Key** (3) 関数に影響を及ぼします。キーボードがブロックまたは使用されている場合、キー・ストロークは処理されません。 4 分のタイムアウトが発生するまで、関数がキー・ストロークの再送を試行するか、あるいは、キーボードがブロックまたは使用されていることが確認されて即座に関数が戻るかどうかを、このオプションにより決定します。

値	説明
RETRY	キー・ストロークが送信されるか、4 分のタイムアウトが発生するまでキー・ストロークの送信を試行し続けます。
NORETRY	キーボードがブロックまたは使用されていることが確認されたら、関数は即座に戻ります。

20. **DBCS の場合のみ:** 次の表は、 **Copy Presentation Space** (5)、 **Copy Presentation Space to String** (8)、 **Copy String to Presentation Space** (15)、**Copy String to Field** (33)、および **Copy Field to String** (34) に影響を及ぼします。

値	説明
NOEAD	DBCS に関する属性文字を渡しません。
EAD	表示スペースのデータと 2 バイト文字セット (DBCS) に関する 2 つの属性文字を渡します。 (ユーザーはデータの他にそれぞれの文字ごとに 2 バイトを受け取ります。したがって、表示スペース用に 2 倍のサイズのバッファーを事前に割り当てる必要があります。)

21. **DBCS の場合のみ:** 次の表は、 **Copy Presentation Space** (5)、 **Copy Presentation Space to String** (8)、 **Copy String to Presentation Space** (15)、**Copy String to Field** (33)、および **Copy Field to String** (34) に影響を及ぼします。

値	説明
NOSO	表示スペースから、シフトイン (SI) およびシフトアウト (SO) の各制御文字を除いたデータを渡します。
SO	X'0E' に変換された SI 制御文字と X'0F' に変換された SO 制御文字を含む表示スペース・データを渡します。バッファーに割り当てられるサイズは、そこに保管するデータの長さによって決まります。
SPACESO	X'20' (ブランク) に変換された SI 制御文字と SO 制御文字を含む表示スペース・データを渡します。バッファーに割り当てられるサイズは、そこに保管するデータの長さによって決まります。

22. 次の表の値は、**Copy Presentation Space** (5)、**Copy Presentation Space to String** (8)、**Copy String to Presentation Space** (15)、**Copy String to Field** (33)、**Copy Field to String** (34) **Search Field** (30) および **Query Sessions** (10) に影響を及ぼします。

値	説明
EXTEND_PS	5250 エミュレーションは、24 行 80 桁の表示スペースをサポートします。コミュニケーション・マネージャー 5250 エミュレーションが 25 番目の行を表示するインスタンスもあります。このようなことが起こるのは、ホストからのエラー・メッセージが表示される場合、またはオペレーターが SysReq キーを選択する場合です。パーソナル・コミュニケーションズは、25 行目の情報を 24 行目に表示しますが、EHLLAPI は、通常は実際の 24 行を参照します。**EXTEND_PS** オプションを指定すると、EHLLAPI アプリケーションでコミュニケーション・マネージャー EHLLAPI と同じインターフェースを使用することができ、このような状況が発生した時に有効な表示スペースが拡張されます。
NOEXTEND_PS	上記の状態が発生した場合に表示スペースを拡張しません。これはデフォルト値です。

23. 次の表の値は、**Connect Presentation Space** (1) と **Connect Window Services** (101) 関数に影響を及ぼします。これらのオプションは、アプリケーションで、それが接続されている表示スペースを他の表示スペースと共用できるか、また共用することになるかを指定します。各 **Set Session Parameter** 呼び出しについて、次の値のうち 1 つだけを指定することができます。

値	説明
SUPER_WRITE	アプリケーションは、共用が可能で書き込みアクセス許可を有する他のアプリケーションに同時に同じ表示スペースに接続することを許可します。この関数を呼び出したアプリケーションでは、監視タイプの関数が実行されますが、表示スペースを共用する他のアプリケーションに対してエラーは生成されません。
WRITE_SUPER	アプリケーションは、書き込みアクセスを要求し、監視アプリケーションだけにその表示スペースへ同時に接続することを許可します。これはデフォルト値です。
WRITE_WRITE	アプリケーションは、書き込みアクセスを要求し、その動作が予測可能なパートナーまたは他のアプリケーションに表示スペースを共用することを許可します。
WRITE_READ	アプリケーションは、書き込みアクセスを要求し、読み取り専用関数を実行する他のアプリケーションに表示スペースを共用することを許可します。また、表示スペースのコピーおよび他の読み取り専用操作を通常通り実行することもできます。
WRITE_NONE	アプリケーションは、表示スペースを排他使用します。監視アプリケーションも含めて、他のアプリケーションは、表示スペースを共用できません。また、表示スペースのコピーおよび他の読み取り専用操作を通常通り実行することもできます。
READ_WRITE	アプリケーションは、表示スペースをモニターするための読み取りアクセスのみを要求し、読み取りまたは書き込み (あるいはその両方の) 関数を実行する他のアプリケーションに表示スペースを共用することを許可します。また、表示スペースのコピーおよび他の読み取り専用操作を通常通り実行することもできます。

24. 次の表の値により、アプリケーションは、表示スペースの共用の要求を、パートナー・アプリケーション (一緒に動作するよう開発されたアプリケーション)

との間での共用に制限することができます。

値	説明
NOKEY	この値を指定すると、アプリケーションは、**KEY** パラメーターを指定していない既存のアプリケーションとの互換性を持つようになります。
KEY$*nnnnnnnn*	サポートする表示スペースの共用アクセスを制限するために、キーワードを使用します。キーワードは、正確に 8 バイトの長さでなければなりません。

戻りパラメーター

この関数はデータ・ストリング長および戻りコードを戻します。

データ・ストリング長:
設定された正しいセッション・パラメーターの数。

戻りコード:
以下のコードが定義されています。

戻りコード	説明
0	セッション・パラメーターが設定されました。
2	1 つ以上のパラメーターが無効でした。
9	システム・エラーが発生しました。

1390/1399 および 1137 コード・ページ・サポート

コード・ページ 1390/1399 ユニコード機能は 3270 および 5250 セッションでのみ選択可能です。コード・ページ 1137 ユニコード機能は 5250 セッションでのみ選択可能です。

ユニコード・セッションの 1390/1399 および 1137 コード・ページ・サポートについて、以下のセッション・オプションの違いに注意してください。

- セッション・オプション STREOT は、以下の理由により、ユニコード・ストリングに使用しないでください。
  - セッション・オプション STREOT では、ストリングの長さを明示的に与えないよう指定されています。 EOT 文字がストリングの終わりを示します。 EOT 文字をスキャンしてストリングの長さを検出できます。この EOT 文字は単一バイト値として保管されます。単一バイト EOT 文字は、ユニコード・ストリングに使用することはできません。
    - *シナリオ:* ユーザーが EOT 文字を、'A' (ASCII 値が 0X'41') として設定します。ユーザーが関数に渡すストリング・バッファーにユニコード文字が入っていると、このユニコード文字の低位バイトはストリング区切り文字として扱われます。そのため、単一バイト EOT 文字は、ストリング区切り文字として使用することはできません。
  - **Set Session Parameter** 関数は PCOMM セッションから独立しており、同じ設定をすべての PCOMM のセッションに適用するので、 EOT 文字をユニコード文字として保管することはできません。 EOT がユニコード文字として保管

されると、SBCS および DBCS のインプリメンテーションが EOT 文字を渡す方法によって影響されます。現時点では、 EOT 文字は単一バイト値であることが予期されます。

注: セッション・オプション STREOT を使用する場合は、起こりうる結果は予測できません。単一バイト区切り文字が、バッファーで渡すユニコード値の一部にならないことが確かな場合は、その単一バイト区切り文字をユニコード・ストリングで使用できます。

- 175 ページのSTREOT にリストされている理由と同じ理由で、セッション・オプション ESC はユニコード・セッションでサポートされません。
- セッション・オプション XLATE はユニコードでサポートされません。このオプションを設定しても、無視されます。

Start Close Intercept (41)

3270	*5250*	*VT*
YES	YES	YES

Start Close Intercept 関数を使用して、アプリケーションは、ユーザーがエミュレーター・セッション・ウィンドウから終了オプションを選択したときに生成される終了要求を代行受信できます。この関数は **Stop Close Intercept** (43) 関数が要求されるまで、クローズ要求を代行受信してそれを破棄します。

この関数を使用した後、アプリケーション・プログラムは **Query Close Intercept** (42) 関数を使用して、いつ終了要求が出されたかを確認することができます。

前提呼び出し

この関数には前提呼び出しはありません。

呼び出しパラメーター

バイト位置	**定義**	
	標準インターフェース	拡張インターフェース
関数番号	必ず 41 にしてください。	
データ・ストリング	次の表を参照してください。	
長さ	5 または 6 にしてください。	必ず 12 にしてください。
PS 位置	NA	

データ・ストリングには次のものがあります。

バイト位置		**定義**
標準	拡張	
1	1	表示スペースの 1 文字の短縮名 (PSID)。
	2 ～ 4	予約済み。

バイト位置		定義
4 ～ 5		この位置にあるデータは EHLLAPI で無視されます。ただし、移行プログラムでこの位置にデータを指定していても、エラーは発生しません。このデータはアプリケーションの移行時に互換性を保つために設けられています。
6	5	非同期メッセージ・モードを要求する場合は、M を指定します (Windows のみ)。
	6 ～ 8	予約済み。
2 ～ 3	9 ～ 12	バイト位置 5 (16 ビットの場合は 6) に M を指定した場合、メッセージを受け取るウィンドウのウィンドウ・ハンドルを設定してください。メッセージは、RegisterWindowMessage (PCSHLL) の戻り値 (0 以外) です。

戻りパラメーター

この関数はデータ・ストリングおよび戻りコードを戻します。

データ・ストリング:

非同期メッセージ・モードをバイト位置 5 (標準インターフェースの場合は 6) に指定しないで、関数が正常終了した場合、以下のデータ・ストリングが戻されます。

バイト位置		定義
標準	拡張	
1	1	表示スペースの 1 文字の短縮名 (PSID)。
	2 ～ 8	予約済み。
	9 ～ 12	EHLLAPIによってイベント・オブジェクト・アドレスが戻される 4 バイト値。アプリケーションは、このイベント・オブジェクトを待ちます。(32 ビットのみ。)

データ・ストリング:

M (非同期メッセージ・モード) をバイト位置 5 (標準インターフェースの場合は 6) に指定して、関数が正常終了した場合、以下のデータ・ストリングが戻されます。

バイト位置		定義
標準	拡張	
1	1	表示スペースの 1 文字の短縮名 (PSID)
	2 ～ 8	予約済み
2 ～ 3	9 ～ 10	非同期メッセージ・モードのタスク ID

注: ユーザーが終了オプションを選択すると、アプリケーション・ウィンドウはメッセージを受け取ります。メッセージは、RegisterWindowMessage(PCSHLL) の戻り値です。wParam パラメーターには、この関数呼び出しによって戻されたタスク ID が入ります。 lParam パラメーターの HIWORD には戻りコード 26 (代行受信の終了) が入り、lParam パラメーターの LOWORD には関数番号 41 が入ります。

戻りコード:

以下のコードが定義されています。

戻りコード	説明
0	**Start Close Intercept** 関数は正常終了しました。
1	ホストの無効な表示スペースが指定されました。
2	パラメーター・エラーが生じました。
9	システム・エラーが発生しました。
10	エミュレーション・プログラムはこの関数をサポートしていません。

使用上の注意

1. 開始要求関数が戻される場合は、戻されるイベント・オブジェクトまたはセマフォーは非通知状態になります。イベント・オブジェクトは、終了要求が出されるたびに通知状態になります。複数の終了要求イベントに対する通知を受信するためには、**SetEvent** または **Query Close Intercept** (42) 関数を使用して、毎回イベント・オブジェクトを通知状態にすることが必要です。
2. この関数を使用した後、アプリケーション・プログラムは **Query Close Intercept** (42) 関数を使用して、いつ終了要求が出されたかを確認することができます。アプリケーションは、そのイベントが発生したことを確認するために戻されるイベント・オブジェクトを待つこともできます。
3. これは排他呼び出しではありません。同じ短縮セッション ID に対して、複数のアプリケーションがこの関数を要求することができます。
4. アプリケーションがセッションに対する終了要求を代行受信しない場合、その後ユーザーがエミュレーター操作ダイアログから選択した終了要求がそのセッションに対する正常停止の要求となります。

Start Communication Notification (80)

3270	*5250*	*VT*
YES	YES	YES

Start Communication Notification 関数はプロセスを開始します。このプロセスによって、ユーザーの EHLLAPI アプリケーションは、指定されたセッションがホストに接続されているかどうかを判別できます。

この関数を使用した後、アプリケーションは、**Query Communication Event** (81) を使用して、セッションが接続されているか、または切断されているかを判別できます。

前提呼び出し

この関数には前提呼び出しはありません。

呼び出しパラメーター

	拡張インターフェース

関数番号	必ず 80 にしてください。
データ・ストリング	事前に割り振られた構造。次の表を参照してください。
長さ	16
PS 位置	NA

呼び出しデータ構造には、以下の要素が含まれます。

バイト位置	定義
1	表示スペースの 1 文字の短縮名 (PSID)。
2 ～ 4	予約済み
5	次のいずれかの値: • 文字 C はセッションがホストから切断、またはホストに接続されたときの通知を要求します。 • 文字 A は、非同期モードの通知を要求します。A を指定した場合は、バイト位置 9 ～ 12 にイベント・オブジェクト (Windows) のアドレスが戻されます。文字 C を、位置 13 に指定する必要があります。 • 文字 M は、非同期メッセージ・モードの通知を要求します。 M を指定する場合、イベント選択文字 C をバイト位置 13 に指定する必要があります。
6 ～ 8	予約済み
9 ～ 12	バイト位置 5 に M を指定した場合、メッセージを受け取るウィンドウのウィンドウ・ハンドルを指定してください。メッセージは、RegisterWindowMessage (PCSHLL) の戻り値 (0 以外) です。
13	バイト位置 5 が A または M の場合、このバイトは文字 C を含む必要があります。
14 ～ 16	予約済み

データ・ストリング

A (非同期モード) を呼び出しデータ構造のバイト位置 5 に指定して、関数が正常終了した場合、以下のデータ・ストリングが戻されます。

バイト位置	定義
1	1 文字の表示スペース短縮名 (PSID)
2 ～ 8	予約済み
9 ～ 12	EHLLAPI によってイベント・オブジェクト・ハンドルが戻される、4 バイトの 2 進値。アプリケーションは、このイベント・オブジェクトを待ちます。

M (非同期メッセージ・モード) を呼び出しデータ構造のバイト位置 5 に指定して、この関数が正常終了した場合、以下のデータ・ストリングが戻されます。

バイト位置	定義
1	1 文字の表示スペース短縮名 (PSID)
2 ～ 8	予約済み

9 ～ 10	非同期メッセージ・モードのタスク ID

セッションがアプリケーション・ウィンドウを接続または切断したとき、メッセージを受信します。このメッセージは、RegisterWindowMessage (PCSHLL) の戻り値です。 wParam には、関数呼び出しによって戻されたタスク ID が入ります。セッションがホストに接続された場合、lParam の HIWORD には 21 が入り、セッションがホストから切断された場合、lParam の HIWORD には 22 が入ります。lParam の LOWORD には関数番号 80 が入ります。

戻りパラメーター

戻りコード	定義
0	関数は正常終了しました。
1	誤った PSID が指定されました。
2	指定されたパラメーターに誤りがありました。
9	システム・エラーが発生しました。

使用上の注意

1. 1 つのアプリケーション・プログラムが複数のホスト・セッションでこの関数を発行することができます。セッションの通信状況を判別するために、 **Query Communication Event** (81) 関数を使用することができます。
2. アプリケーションが非同期オプションを選択した場合、そのアプリケーションは Windows SDK 呼び出し **WaitForSingleObject** を使用して、そのセッションの通信状況が変化するのを待つことができます。
3. このイベント・オブジェクトは、最初は非通知状態にあります。イベントが起こるたびに、イベント・オブジェクトは通知されます。複数のイベントに対する通知を受信するためには、アプリケーションは、 Windows SDK 呼び出し **ResetEvent**、または、関数 81 **Query Communications Event** を使用して、通知されるたびにイベント・オブジェクトを非通知状態にする必要があります。
4. 同じアプリケーションからの同じオプションを指定したこの関数への複数の呼び出しは、無視されます。
5. この関数は 1 つのアプリケーションに対して排他的ではありません。同じセッション ID に対して、複数のアプリケーションがこの関数を要求することができます。

Start Host Notification (23)

3270	*5250*	*VT*
YES	YES	YES

Start Host Notification 関数は、指定されたホスト表示スペースまたは OIA が更新されたかどうかを EHLLAPI アプリケーション・プログラムが判別するための処理を開始します。

この関数を使用した後、アプリケーション・プログラムは **Query Host Update** (24) 関数を使用して、いつホスト・イベントが発生したかを確認することができます。

前提呼び出し

この関数には前提呼び出しはありません。

呼び出しパラメーター

	標準インターフェース	拡張インターフェース
関数番号	必ず 23 にしてください。	
データ・ストリング	事前に割り振られたストリング。次の表を参照してください。	
長さ	6 または 7 が暗黙指定されます。	16
PS 位置	NA	

呼び出しデータ・ストリングには次のものがあります。

バイト位置		定義
標準	拡張	
1	1	次のいずれかの値: • 表示スペースの 1 文字の短縮名 (PSID) • ホスト接続のホスト表示スペースに対する要求を示すブランクまたは NULL 文字
	2 〜 4	予約済み。
2	5	次のいずれかの値: • B — ホスト表示スペースと OIA の両方の更新通知を要求します。 • O — OIA の更新通知のみを要求します。 • P — ホスト表示スペースの更新通知のみを要求します。 • A — 非同期モードの通知を要求します。A を指定した場合は、バイト位置 9 〜 12 にイベント・オブジェクトのアドレスが戻されます。イベント選択文字 B、O、P はバイト位置 13 に指定する必要があります。 • M — 非同期メッセージ・モードの通知を要求します。 M を指定する場合、イベント選択文字 B、O、P のいずれかをバイト位置 13 (16 ビットの場合は 7) に指定します。 • E は、プリンター・セッション中の完了通知を要求する文字です。
	6 〜 8	予約済み。

バイト位置		定義
3 ～ 4	9 ～ 12	バイト位置 5 (16 ビットの場合は 2) に M を指定した場合、メッセージを受け取るウィンドウのウィンドウ・ハンドルを指定してください。メッセージは、RegisterWindowMessage (PCSHLL) の戻り値 (0 以外) です。
7	13	バイト位置 5 (16 ビットの場合は 2) に A または M を指定した場合、値は次のいずれかです。 • B — ホスト表示スペースと OIA の両方の更新通知を要求します。 • O — OIA の更新通知のみを要求します。 • P — ホスト・プレゼンテーションの更新通知のみを要求します。
	14 ～ 16	予約済み。

戻りパラメーター

この関数はデータ・ストリングおよび戻りコードを戻します。

データ・ストリング:

A (非同期モードの通知) をバイト位置 5 に指定し、関数が正常終了した場合、以下のデータ・ストリングが戻されます。

バイト位置		定義
標準	拡張	
1	1	表示スペースの 1 文字の短縮名 (PSID)。
	2 ～ 8	予約済み。
	9 ～ 12	EHLLAPI によってイベント・オブジェクト・アドレスが戻される 4 バイト値。アプリケーションは、このイベント・オブジェクトを待ちます (32 ビットのみ)。

データ・ストリング:

M (非同期メッセージ・モード) をバイト位置 5 (標準インターフェースの場合は 2) に指定して、関数が正常終了した場合、以下のデータ・ストリングが戻されます。

バイト位置		定義
標準	拡張	
1	1	表示スペースの 1 文字の短縮名 (PSID)
	2 ～ 8	予約済み
3 ～ 4	9 ～ 10	非同期メッセージ・モードのタスク ID

注: OIA または表示スペースを更新すると、アプリケーション・ウィンドウはメッセージを受け取ります。メッセージは、RegisterWindowMessage(PCSHLL) の戻り値です。wParam パラメーターには、関数呼び出しによって戻されたタスク ID が入ります。 lParam の HIWORD には戻りコード 21 (OIA の更新)、22

(ホスト表示スペースの更新)、23 (OIA とホスト表示スペースの両方の更新) が入り、lParam パラメーターの LOWORD には関数番号 23 が入ります。

戻りコード:

以下のコードが定義されています。

戻りコード	定義
0	**Start Host Notification** 関数は正常終了しました。
1	ホストの無効な表示スペースが指定されました。
2	指定されたパラメーターに誤りがありました。
9	システム・エラーが発生しました。

使用上の注意

1. 1 つのアプリケーション・プログラムが複数のホスト・セッションでこの関数を発行することができます。 **Pause** (18) 関数により、1 つ以上のホスト・セッション (PS、OIA、またはその両方) が変更された時点でアプリケーションに通知することができます。PS、OIA、またはその両方が変更されたかどうかを判別するために **Query Host Update** (24) 関数を使用することができます。
2. アプリケーションで非同期オプションを選択した場合、アプリケーションは、そのイベントが発生した時期を判別するために戻されるイベント・オブジェクトまたはセマフォーを待つこともできます。
3. イベント・オブジェクトまたはセマフォーは、最初は非送信状態であり、該当のイベントが発生するたびに送信されます。複数のイベントに対する通知を受信するためには、アプリケーションは、 **ResetEvent** または **Query Host Update** (24) 関数を使用して、通知されるたびにイベント・オブジェクトを非通知状態にする必要があります。
4. アプリケーションは、同じオプションで Start Host Notification を複数回要求することはできません。
5. これは排他呼び出しではありません。同じ短縮セッション ID に対して、複数のアプリケーションがこの関数を要求することができます。

Start Keystroke Intercept (50)

3270	*5250*	*VT*
YES	YES	YES

Start Keystroke Intercept 関数により、ワークステーションのアプリケーションは、端末オペレーターがセッションに送信したキー・ストロークをフィルターにかけることができます。この関数を呼び出した後、キー・ストローク・キューがオーバーフローするまで、あるいは、**Stop Keystroke Intercept** (53) 関数または **Reset System** (21) 関数が呼び出されるまで、キー・ストロークは代行受信されて保管されます。代行受信したキー・ストロークは以下のように処理することができます。

- **Get Key** (51) 関数により受信し、 **Send Key** (3) 関数で同じセッションまたは他のセッションへ送信する。
- **Post Intercept Status** (52) 関数により、受け入れまたは拒否する。
- **Send Key** (3) 関数で別のキー・ストロークに置き換える。

- 別の処理のトリガーとして使用する。

前提呼び出し

この関数には前提呼び出しはありません。

呼び出しパラメーター

	標準インターフェース	拡張インターフェース
関数番号	必ず 50 にしてください。	
データ・ストリング	次の表を参照してください。	
長さ	キー・ストローク・バッファー・サイズ。 EHLLAPI は、このバッファーに最低 32 バイトを割り振ります。	
PS 位置	NA	

呼び出しデータ・ストリングには次のものが入っています。

バイト位置		定義
標準	拡張	
1	1	次のいずれかの値: • 特定のホスト表示スペースの短縮名 (PSID) • ホスト接続のホスト表示スペースに対する要求を示すブランクまたは NULL 文字
	2 〜 4	予約済み。
2	5	**オプション・コード**文字 • D — AID キー・ストロークのみ。 • L — すべてのキー・ストローク。 • M — 非同期メッセージ・モードの通知要求 (Windows のみ)。 M を指定する場合、コード文字 D または L をバイト位置 13 (16 ビットの場合は 7) に指定します。
	6 〜 8	予約済み。
3 〜 4	9 〜 12	バイト位置 5 (16 ビットの場合は 2) に M を指定した場合、メッセージを受け取るウィンドウのウィンドウ・ハンドルを指定してください。メッセージは、RegisterWindowMessage (PCSHLL) の戻り値 (0 以外) です。
7	13	バイト位置 5 (16 ビットの場合は 2) に M を指定した場合、値は次のいずれかです。 • D — AID キー・ストロークのみ。 • L — すべてのキー・ストローク。
	14 〜 16	予約済み。

データ・ストリング:

M (非同期メッセージ・モード) をバイト位置 5 (標準インターフェースの場合は 2) に指定して、関数が正常終了した場合、以下のデータ・ストリングが戻されます。

バイト位置		定義
標準	拡張	
1	1	表示スペースの 1 文字の短縮名 (PSID)
	2 ～ 8	予約済み
3 ～ 4	9 ～ 10	非同期メッセージ・モードのタスク ID

注: ユーザーがセッションへキー・ストロークを送信すると、アプリケーション・ウィンドウはメッセージを受け取ります。メッセージは、RegisterWindowMessge (PCSHLL) の戻り値です。wParam パラメーターには、関数呼び出しによって戻されたタスク ID が入ります。 lParam パラメーターの HIWORD には戻りコード 0 (関数の正常終了) が入り、LOWORD には関数番号 50 が入ります。

戻りパラメーター

戻りコード	説明
0	**Start Keystroke Intercept** 関数は正常終了しました。
1	無効な表示スペースが指定されました。
2	無効なオプションが指定されました。
4	該当する表示スペースが使用中のため、関数は実行できません。
9	システム・エラーが発生しました。 Release が使用中です。

使用上の注意

1. **Get Key** (51) 関数に対して戻りコードが 31 になる場合は、以下のどちらかを行ってください。
   - この関数の呼び出し元のデータ・ストリング長パラメーターの値を増やしてください。
   - **Get Key** (51) 関数をより頻繁に実行してください。

   代行受信されたキー・ストロークは、バッファー中の 3 バイトを使用します。次の代行受信されたキー・ストロークは、隣接する 3 バイトに置かれます。**Get Key** (51) 関数がキー・ストローク (先入れ、先出し、FIFO) を取り出す場合は、そのキー・ストロークが使用していた 3 バイトを他のキー・ストロークで使用できるようになります。バッファーのサイズを増やすか、バッファーからキー・ストロークを取り出す頻度を増やすことによって、バッファーのオーバーフローを避けることができます。

   PC/3270 において、戻りコード 31 を除去する他の方法は、再開モードで PC/3270 エミュレーターを操作することです。
2. オプション・コード D が指定されている場合、EHLLAPI は非 AID キーを、本来予定されている表示スペースへ書き込み、AID キーのみをアプリケーションに戻します。
3. **Stop Keystroke Intercept** (53) 関数を呼び出してから、EHLLAPI アプリケーションを終了してください。この関数を呼び出さない場合は、キー・ストロークの代行受信機能が引き続き使用可能となり、予測できない結果が発生します。

Start Playing Macro (110)

3270	*5250*	*VT*
YES	YES	YES

Start Playing Macro 関数を使用してマクロを呼び出します。マクロは接続しているセッションで実行されます。

注: このマクロは、パーソナル・コミュニケーションズのユーザー・クラス・アプリケーション・データのディレクトリーに存在する必要があります。さらに、このマクロ名の関数呼び出しには拡張子を指定してはなりません。

前提呼び出し

Connect Presentation Space (1)

呼び出しパラメーター

	標準インターフェース
関数番号	必ず 110 にしてください。
データ・ストリング	次の表を参照してください。
長さ	マクロ名の長さに 3 を加えたもの
PS 位置	NA

バイト位置		**定義**
標準	拡張	
1 〜 2		予約済み
3 〜 n		NULL で終了するマクロ名

戻りパラメーター

戻りコード	**説明**
0	**Start Playing Macro** 関数は正常終了しました。
1	プログラムが現在ホスト・セッションに接続されていません。
2	パラメーター指定でエラーが発生しました。
9	システム・エラーが発生しました。

Stop Close Intercept (43)

3270	*5250*	*VT*
YES	YES	YES

Stop Close Intercept 関数によりアプリケーションは **Start Close Intercept** (41) 関数をオフにすることができます。アプリケーションにより **Stop Close Intercept** 関数が発行されると、その後の終了要求は通常停止となって論理端末セッションへ送信されます。

前提呼び出し

Start Close Intercept (41)

呼び出しパラメーター

	標準インターフェース	拡張インターフェース
関数番号	必ず 43 にしてください。	
データ・ストリング	ホスト表示スペースの 1 文字の短縮セッション ID。	
長さ	1	必ず 4 にしてください。
PS 位置	NA	

呼び出しデータ・ストリングには次のものがあります。

バイト位置		定義
標準	拡張	
1	1	表示スペースの 1 文字の短縮名 (PSID)
	2 ～ 4	予約済み

戻りパラメーター

戻りコード	説明
0	**Stop Close Intercept** 関数は正常終了しました。
1	ホストの無効な表示スペースが指定されました。
2	パラメーター指定でエラーが発生しました。
8	**Start Close Intercept** (41) 関数が事前に発行されていませんでした。
9	システム・エラーが発生しました。
12	セッションが停止しました。

Stop Communication Notification (82)

3270	*5250*	*VT*
YES	YES	YES

Stop Communication Notification は、指定されたセッションでなんらかの通信イベントが発生したかどうかを判別する **Query Communication Event** (81) 関数を使用停止にします。

前提呼び出し

Start Communication Notification (80)

呼び出しパラメーター

	拡張インターフェース
関数番号	必ず 82 にしてください。

データ・ストリング	1 文字のホスト表示スペースの短縮名、またはホスト接続されている表示スペースへの更新要求を表すブランクあるいは NULL。
長さ	4 が暗黙指定されます。
PS 位置	NA

呼び出しデータ構造には、以下の要素が含まれます。

バイト位置	定義
1	表示スペースの 1 文字の短縮名 (PSID)
2 ～ 4	予約済み

戻りパラメーター

戻りコード	定義
0	関数は正常終了しました。
1	誤った PSID が指定されました。
8	この PSID に対して、先に **Start Communication Notification** (80) 関数が呼び出されていません。
9	システム・エラーが発生しました。

Stop Host Notification (25)

3270	*5250*	*VT*
YES	YES	YES

Stop Host Notification 関数は、ホスト表示スペースまたは OIA が更新されたかどうかを確認する **Query Host Update** (24) 関数を使用不可にします。この関数は、ホストのイベントが **Pause** (18) 関数に影響を及ぼすことも防ぎます。

前提呼び出し

Start Host Notification (23)

呼び出しパラメーター

	標準インターフェース	拡張インターフェース
関数番号	必ず 121 にしてください。	
データ・ストリング	以下の注を参照してください。	
長さ	1 が暗黙指定されます。	必ず 4 にしてください。
PS 位置	NA	

呼び出しデータ・ストリングには次のものがあります。

バイト位置		定義
標準	拡張	
1	1	表示スペースの 1 文字の短縮名 (PSID)
	2 ～ 4	予約済み

注: 該当する表示スペースの 1 文字の短縮 ID、またはホストで接続する表示スペースへの要求を表すブランクあるいは NULL。

戻りパラメーター

戻りコード	定義
0	**Stop Host Notification** 関数は正常終了しました。
1	ホストの無効な表示スペースが指定されました。
8	**Start Host Notification** (23) 関数が事前に発行されていませんでした。
9	システム・エラーが発生しました。

Stop Keystroke Intercept (53)

3270	*5250*	*VT*
YES	YES	YES

Stop Keystroke Intercept 関数は、アプリケーション・プログラムのキー・ストローク代行受信を終了します。

前提呼び出し

Start Keystroke Intercept (50)

呼び出しパラメーター

	標準インターフェース	拡張インターフェース
関数番号	必ず 53 にしてください。	
データ・ストリング	該当する表示スペースの短縮名 (PSID)	
長さ	1 が暗黙指定されます。	必ず 4 にしてください。
PS 位置	NA	

呼び出しデータ・ストリングには次のものがあります。

バイト位置		定義
標準	拡張	
1	1	表示スペースの 1 文字の短縮名 (PSID)
	2 ～ 4	予約済み

戻りパラメーター

戻りコード	説明
0	**Stop Keystroke Intercept** 関数は正常終了しました。
1	無効な表示スペースが指定されました。
8	この表示スペースに対して、先に **Start Keystroke Intercept** (50) 関数が呼び出されていません。
9	システム・エラーが発生しました。

Wait (4)

3270	*5250*	*VT*
YES	YES	YES

Wait 関数は、ホストで接続する表示スペースの状況をチェックします。セッションがホストの応答 (XCLOCK (X []) または XSYSTEM で示される) を待機している場合、**Wait** 関数は、待機状態が解除されたかどうかを確認するため、EHLLAPI を最大 1 分間待機させます。

前提呼び出し

Connect Presentation Space (1)

呼び出しパラメーター

	標準インターフェース	拡張インターフェース
関数番号	必ず 4 にしてください。	
データ・ストリング	NA	
長さ	NA	
PS 位置	NA	

戻りパラメーター

戻りコード	定義
0	キーボードはアンロックされており、入力できます。
1	アプリケーション・プログラムが正しいセッションに接続されていません。
4	XCLOCK (X []) または XSYSTEM の間にタイムアウトになりました。
5	キーボードがロックされています。
9	システム・エラーが発生しました。

使用上の注意

1. **Wait** 関数は、**Send Key** (3) などの関数によるホスト要求を完了するために必要な時間を提供します。 **Set Session Parameters** (9) 関数を使用して、TWAIT、LWAIT、または NWAIT の各オプションを要求することができます。12 (170 ページ) を参照してください。
2. この関数を使用して、ホストの OIA が使用禁止かどうかを調べることができます。
3. ホストがキーボードをアンロックすると、 **Wait** 関数が実行されます。したがって、戻りコード 0 は、トランザクションが完了したことを必ずしも意味しません。トランザクションの完了を確認するには、**Search Field** (30) 関数または **Search Presentation Space** (6) 関数を、**Wait** 関数と組み合わせて使用し、該当するキーワード・プロンプトを探してください。

Window Status (104)

3270	*5250*	*VT*
YES	YES	YES

Window Status 関数を使用して、アプリケーションはウィンドウの表示スペース・サイズ、位置、表示状態を照会したり変更することができます。

前提呼び出し

Connect Window Services (101)

呼び出しパラメーター

	標準インターフェース	**拡張インターフェース**
関数番号	必ず 104 にしてください。	
データ・ストリング	次の表を参照してください。	
長さ	16 または 20 にしてください。	24 または 28 にしてください。
PS 位置	NA	

呼び出しデータ・ストリングには次のものがあります。

バイト位置		**定義**
標準	拡張	
1	1	表示スペースの 1 文字の短縮名 (PSID)
	2 ～ 4	予約済み

バイト位置		定義
2	5	要求オプションの値。次のうちの 1 つを選択してください。 • 状況設定には X'01' **注:** セッションが複合 OLE 文書に組み込まれている場合は、この関数のセット形式 (バイト 5 = X'01') は常に 0 を戻しますが、何も行われません。 • 状況照会には X'02' • 拡張状況照会には X'03'
	6	予約済み

要求オプションの値が X'01' (状況設定) の場合:

バイト位置		定義
標準	拡張	
3 〜 4	7 〜 8	要求オプションが 1 (状況設定) の場合には、状況設定ビットを含む 16 ビット/32 ビットのワード。要求オプションが状況設定の場合、有効な戻り値は以下のコードです。 **X'0001'** ウィンドウ・サイズを変更します。 (最小化、最大化、復元、移動は無効) **X'0002'** ウィンドウを移動します。 (最小化、最大化、サイズ変更、復元は無効) **X'0004'** ZORDER ウィンドウの置き換え。 **X'0008'** ウィンドウを可視状態に設定します。 **X'0010'** ウィンドウを不可視状態に設定します。 **X'0080'** ウィンドウを活動状態にします。 (ZORDER が指定されていない場合には、ウィンドウ・フォーカスを設定し、前面に配置します。この場合、ZORDER 配置が使用されます。) **X'0100'** ウィンドウを非活動状態にします。 (ZORDER も共に指定されていない場合には、ウィンドウを非活動状態にし、一番下のウィンドウにします。この場合、ZORDER 配置が使用されます。) **X'0400'** ウィンドウを最小化します。 (最大化、復元、サイズ変更、移動は無効) **X'0800'** ウィンドウを最大化します。 (最小化、復元、サイズ変更、移動は無効) **X'1000'** ウィンドウを復元します。 (最小化、最大化、サイズ変更、移動は無効)
5 〜 6	9 〜 12	ウィンドウ位置の X 座標を含む 16 ビット/32 ビットのワード。 (移動オプションが設定されていない場合には無視されます。)
7 〜 8	13 〜 16	ウィンドウ位置の Y 座標を含む 16 ビット/32 ビットのワード。 (移動オプションが設定されていない場合には無視されます。)

バイト位置		定義
9 ～ 10	17 ～ 20	装置単位での X ウィンドウ・サイズを含む 16 ビット/32 ビットのワード。 (サイズ・オプションが設定されていない場合には無視されます。)
11 ～ 12	21 ～ 24	装置単位での Y ウィンドウ・サイズを含む 16 ビット/32 ビットのワード。 (サイズ・オプションが設定されていない場合には無視されます。)
13 ～ 16	25 ～ 28	ウィンドウの相対配置のためのウィンドウ・ハンドルを含む 16 ビット/32 ビットのワード。この 2 つのワードは設定オプション専用です。 (ZORDER オプションが設定されていない場合には無視されます。) 有効な値は次の通りです。 X'00000003' — すべての兄弟ウィンドウの前に配置。 X'00000004' — すべての兄弟ウィンドウの後ろに配置。

要求オプションの値が X'02' (状況照会) の場合:

バイト位置		定義
標準	拡張	
3 ～ 4	7 ～ 8	要求オプションが 2 (拡張状況照会) の場合には、X'0000' を含む 16 ビット/32 ビットのワード。要求オプションが状況照会の場合、可能な戻り値は以下のコードです。複数の状態も可能です。 **X'0008'** ウィンドウが可視状態です。 **X'0010'** ウィンドウは不可視状態です。 **X'0080'** ウィンドウは活動状態です。 **X'0100'** ウィンドウは非活動状態です。 **X'0400'** ウィンドウは最小化されています。 **X'0800'** ウィンドウは最大化されています。
5 ～ 6	9 ～ 12	ウィンドウ位置の X 座標を含む 16 ビット/32 ビットのワード。 (移動オプションが設定されていない場合には無視されます。)
7 ～ 8	13 ～ 16	ウィンドウ位置の Y 座標を含む 16 ビット/32 ビットのワード。 (移動オプションが設定されていない場合には無視されます。)
9 ～ 10	17 ～ 20	装置単位での X ウィンドウ・サイズを含む 16 ビット/32 ビットのワード。 (サイズ・オプションが設定されていない場合には無視されます。)
11 ～ 12	21 ～ 24	装置単位での Y ウィンドウ・サイズを含む 16 ビット/32 ビットのワード。 (サイズ・オプションが設定されていない場合には無視されます。)

バイト位置		定義
13 〜 16	25 〜 28	ウィンドウの相対配置のためのウィンドウ・ハンドルを含む 16 ビット/32 ビットのワード。この 2 つのワードは設定オプション専用です。 (ZORDER オプションが設定されていない場合には無視されます。) 有効な値は次の通りです。 X'00000003' — すべての兄弟ウィンドウの前に配置。 X'00000004' — すべての兄弟ウィンドウの後ろに配置。

要求オプション値が X'03' (拡張状況照会) の場合:

バイト位置		定義
標準	拡張	
3 〜 4	7 〜 8	要求オプションが 3 (拡張状況照会) の場合には、X'0000' を含む 16 ビット/32 ビットのワード。要求オプションが拡張状況照会の場合、可能な戻り値は以下のコードです。複数の状態も可能です。 **X'0008'** ウィンドウが可視状態です。 **X'0010'** ウィンドウは不可視状態です。 **X'0080'** ウィンドウは活動状態です。 **X'0100'** ウィンドウは非活動状態です。 **X'0400'** ウィンドウは最小化されています。 **X'0800'** ウィンドウは最大化されています。
5 〜 6	9 〜 10	X 方向の現行のフォント・サイズを含む 16 ビット/32 ビットのワード。値はスクリーン・ペル単位です。
7 〜 8	11 〜 12	Y 方向の現行のフォント・サイズを含む 16 ビット/32 ビットのワード。値はスクリーン・ペル単位です。
9 〜 12	13 〜 16	予約済み。この値は常に 0 です。
13 〜 14	17 〜 18	表示スペースの最初の可視文字の行番号を含む 16 ビット/32 ビットのワード。固定サイズ・フォント・オプションが無効で、表示スペースがいくらか隠れるようにウィンドウ・サイズが変更されていない場合、この値は通常 1 です。
15 〜 16	19 〜 20	表示スペースの最初の可視文字の列番号を含む 16 ビット/32 ビットのワード。
17 〜 20	21 〜 24	セッションの表示スペース・ウィンドウ・ハンドルを含む 16 ビット/32 ビットのワード。

戻りパラメーター

戻りコード	説明
0	**Window Status** 関数は正常終了しました。
1	表示スペースが正しくないか、または接続されていません。
2	無効なオプションが指定されました。

戻りコード	説明
9	システム・エラーが発生しました。
12	セッションが停止しました。

使用上の注意

論理端末 (LT) のウィンドウは文字セルを使用しています。 LT ウィンドウのサイズを再変更する場合、文字セルの切り捨てを防ぐために LT は数を切り上げます。必要なサイズや位置が要求されたものとわずかに異なることがあります。設定オプションと照会オプションに従い、最終的なプレゼンテーション・マネージャー・ウィンドウの位置とサイズを決定してください。すべての x 座標と y 座標の位置とサイズはペル単位です。

Write Structured Fields (127)

3270	*5250*	*VT*
YES	NO	NO

Write Structured Fields 関数により、アプリケーションは構造化フィールド・データをホスト・アプリケーションに書き込みます。呼び出しで S (同期) を指定した場合、 **Write Structured Fields** 関数が完了するまで、アプリケーションに制御が戻されません。呼び出しで A (非同期) を指定した場合、その呼び出しの直後に、アプリケーションに制御が戻されます。呼び出しで M を指定した場合、その呼び出しの直後に、アプリケーションに制御が戻されます。アプリケーションは、このメッセージを待機します。いずれの場合 (S、A、または M) でも、アプリケーションにより、ホストに対するデータが入るバッファー・アドレスが設定されます。

この関数が非同期に正常終了するためには、以下のステートメントを適用します。

パラメーター・リストの戻りコード・フィールドには、要求された入出力の結果が含まれていない場合もあります。戻りコードが 0 でない場合、要求は失敗しています。アプリケーションでは、戻りコードに応じて、適切な処置を行わなければなりません。

この要求に対する戻りコードが 0 である場合、アプリケーションで、この関数呼び出しと共に戻される要求 ID を使用して、 **Get Request Completion** 関数呼び出しを発行し、その要求 ID に関連する関数が終了したかどうかを判別します。 **Get Request Completion** 関数呼び出しによって、以下の情報が戻されます。

1. 関数要求 ID
2. 非同期要求からのデータ・ストリング・アドレス
3. データ・ストリングの長さ
4. 完了した関数の戻りコード

前提呼び出し

Connect for Structured Fields (120) **Allocate Communication Buffer** (123)

呼び出しパラメーター

	標準インターフェース	拡張インターフェース
関数番号	必ず 127 にしてください。	
データ・ストリング	次の表を参照してください。	
長さ	8、10、14 のいずれかにしてください。	必ず 20 にしてください。
PS 位置	NA	

呼び出しデータ・ストリングには次のものがあります。

バイト位置		定義
標準	拡張	
1	1	表示スペースの 1 文字の短縮名 (PSID)。
	2 ～ 4	予約済み。
2	5	S、A、M のいずれか **S =** 同期。制御は、読み取りが満たされるまでアプリケーションに戻りません。 **A =** 非同期。制御は、アプリケーション (イベント・オブジェクト待機可能) に即時に戻されます。 **M =** 非同期。制御は、アプリケーション (メッセージ待機可能) に即時に戻されます。
	6	予約済み。
3 ～ 4	7 ～ 8	2 バイトの Destination/Origin ID
5 ～ 8	9 ～ 12	データが書き込まれる 4 バイトのバッファー・アドレス。 **Allocate Communications Buffer** (123) 関数を使用してバッファーを取得する必要があります。
9 ～ 10	13 ～ 16	予約済み。
11 ～ 12	17 ～ 20	“M” をバイト位置 5 (16 ビットの場合は 2) に指定した場合、メッセージを受け取るウィンドウのウィンドウ・ハンドルを設定してください。メッセージは、RegisterWindowMessage (“PCSHLL”) の戻り値 (0 以外) です。
13 ～ 14		この位置にあるデータは EHLLAPI によって無視されます。ただし、移行プログラムがこの位置にデータを指定しても、エラーは発生しません。このデータはアプリケーションの移行時に互換性を保つために設けられています。

戻りパラメーター

この関数はデータ・ストリングおよび戻りコードを戻します。

データ・ストリング:

A (非同期) をバイト位置 5 (標準インターフェースの場合は 2) に指定して、関数が正常終了した場合、以下のデータ・ストリングが戻されます。

バイト位置		定義
9 ～ 10	13 ～ 14	2 バイトの関数要求 ID。この関数要求 ID は、この関数呼び出しの完了を判別するために **Get Request Completion** (125) 関数で使用します。
	15 ～ 16	予約済み。
	17 ～ 20	EHLLAPI によってイベント・オブジェクト・アドレスが戻される 4 バイト値。アプリケーションは、このイベント・オブジェクトを待ちます。イベント・オブジェクトがクリアされる場合、アプリケーションは **Get Request Completion** (125) 関数呼び出しを発行して、**Write Structured Fields** 要求の結果を取得しなければなりません(32 ビットのみ。)

注: イベント・オブジェクトは、非同期要求が正常終了するたびに戻されます。イベント・オブジェクトは再使用しないでください。要求ごとに新しいイベント・オブジェクトが戻され、そのイベント・オブジェクトはその要求の間だけ有効です。

データ・ストリング:

M (非同期メッセージ・モード) をバイト位置 5 (標準インターフェースの場合は 2) に指定して、関数が正常終了した場合、以下のデータ・ストリングが戻されます。

バイト位置		定義
9 ～ 10	13 ～ 14	2 バイトの関数要求 ID。この関数要求 ID は、この関数呼び出しの完了を判別するために **Get Request Completion** (125) 関数で使用します。
	15 ～ 16	予約済み。
11 ～ 12	17 ～ 18	非同期メッセージ・モードのタスク ID。
	19 ～ 20	予約済み。

注: 関数が正常終了した場合、アプリケーション・ウィンドウはメッセージを受け取ります。メッセージは、RegisterWindowMessage(PCSHLL) の戻り値です。wParam パラメーターには、関数呼び出しによって戻されたタスク ID が入ります。 lParam パラメーターの HIWORD には戻りコード 0 (関数の正常終了) が、LOWORD には関数番号 127 が入ります。

戻りコード:

以下のコードが定義されています。

戻りコード	説明
0	**Write Structured Fields** 関数は正常終了しました。
1	無効なホスト表示スペースの短縮セッション ID が指定されたか、またはホスト表示スペースが接続されていません。
2	パラメーター指定でエラーが発生しました。
9	システム・エラーが発生しました。
11	リソースが使用できません。(メモリーが使用できません。)

戻りコード	説明
34	インバウンドでホストに送信されたメッセージが取り消されました。
35	ホストからのアウトバウンド送信が取り消されました。
36	要求はリジェクトされました。ホストと接続できません。
37	関数は失敗しました。ホストのインバウンド送信は使用できません。

使用上の注意

1. 戻りコード 35。これは、ホストからのアウトバウンド送信が取り消された後に **Read Structured Fields** または **Write Structured Fields** が要求された場合に戻されます。アプリケーションで修正処置を行う必要があります。
2. 戻りコード 36。この場合、アプリケーションをエミュレーション・プログラムから切断してから再接続して、ホストとの通信を再確立する必要があります。アプリケーションで修正処置を行う必要があります。
3. 戻りコード 37。これはホスト・インバウンドが使用不可の場合に戻されます。
4. EHLLAPI では、アプリケーションごとに 20 までの非同期要求を未処理にすることができます。 20 を超える非同期要求が試行されると、リソースが使用不可であることを示す戻りコード (RC=11) が戻されます。
5. IBM グローバル・ネットワーク接続を使用する場合、非同期要求の最大数は 10 です。

構造化フィールド・データの形式は以下のとおりです。

オフセット	長さ	内容
0	1 ワード	X'0000'
2	1 ワード	m (メッセージの長さ。すなわち、メッセージ中のデータのバイト数。この数は 8 バイトを含むメッセージ・ヘッダーの接頭部を除いたものです。) この値はアプリケーションによって設定しなければなりません。
4	1 ワード	X'0000'
6	1 ワード	X'0000'
8	8 バイト	先頭の (または唯一の) 構造化フィールド・メッセージの長さ。
10	1 バイト	構造化フィールド・メッセージの長さ以外の最初のバイト。
		. . .
m+7	1 バイト	構造化フィールド・メッセージの末尾のバイト。

バイト 0 から 7 は、バッファー・ヘッダーです。最初のこれらの 8 バイトは、エミュレーション・プログラムで使用されます。バッファーのユーザー・セッションは、オフセット 8 からです。バイト 8 および 9 は、最初の構造化フィールドのバイト数が入っています。構造化フィールド・メッセージには複数の構造化フィール

ドを含むことができ、バイト 8 および 9 の 2 バイトを含みます。バイト 8 から *m* +7 は、ホストに送信される構造化フィールド・メッセージのために使用されます。

同期要求: **Write Structured Fields** が同期要求 (データ・ストリング中の S オプション) された際には、制御は要求が満たされた後に限りアプリケーションに戻されます。アプリケーションでは以下のことを想定できます。

- 戻りコードが正しい。
- 通信バッファー (読み取りバッファー) 内のデータが正しい。
- ホストが今後 **Write Structured Fields** 要求を処理しない。

非同期要求: **Write Structured Fields** が非同期要求 (データ・ストリング中の A オプション) された際には、アプリケーションでは以下のことを*想定できません*。

- 戻りコードが正しい。
- 通信バッファー (書き込みバッファー) 内のデータが正しい。
- ホストが今後 **Write Structured Fields** 要求を処理しない。

非同期要求がなされた際には、EHLLAPI は以下の値を戻します。

- 16 ビット要求 ID (データ・ストリングのバイト位置 13 ～ 14) (標準インターフェースの場合は 9 ～ 10)
- イベント・オブジェクトのアドレス (データ・ストリングのバイト位置 17 ～ 20)

これらは、非同期の **Write Structured Fields** 呼び出しを完了させるために使用します。

非同期の **Write Structured Fields** 関数呼び出しの結果を判別するためには、次のステップを行わなければなりません。

- EHLLAPI 戻りコードが 0 でない場合、要求は失敗しています。したがって、非同期要求は行われていません。アプリケーションでは、再び呼び出しを試みる前に、適切な処置を行わなければなりません。
- 戻りコードが 0 ならば、アプリケーションは、 **Get Request Completion** (125) 関数を使用してイベント・オブジェクトが送信状態になるまで待たなければなりません。イベント・オブジェクト **Get Request Completion** (125) 関数は再使用しないでください。イベント・オブジェクトは、**Write Structured Fields** 関数呼び出しから **Get Request Completion** (125) 関数呼び出し完了までの間で有効です。
- イベント・オブジェクトが送信状態になったら、**Get Request Completion** (125) 関数に対する呼び出しで要求 ID パラメーターとして戻された 16 ビットの要求 ID を使用してください。 **Get Request Completion** (125) 関数呼び出しから戻されたデータ・ストリングには、 **Write Structured Fields** 関数呼び出しの最終的な戻りコードが含まれています。

非同期要求: **Write Structured Fields** が非同期要求 (データ・ストリング中の M オプション) された際には、アプリケーションで以下のことを想定できません。

- 戻りコードが正しい。
- 通信バッファー (書き込みバッファー) 内のデータが正しい。
- ホストが今後 **Write Structured Fields** 要求を処理しない。

M オプションを指定して非同期要求がなされた際には、EHLLAPI は以下の値を戻します。

- 16 ビット要求 ID (データ・ストリングのバイト位置 13 〜 14) (標準インターフェースの場合は 9 〜 10)
- 非同期メッセージ・モードのタスク ID (データ・ストリングのバイト位置 17 〜 18) (標準インターフェースの場合は 11 〜 12)

これらは、非同期の **Write Structured Fields** 呼び出しを完了させるために使用します。

第 4 章 WinHLLAPI 拡張関数

この章では、WinHLLAPI プログラミング・サポートを使用した時に提供される拡張関数について説明します。

WinHLLAPI 関数の要約

3270、5250 および VT では、以下の WinHLLAPI 関数が使用可能です。

- 202 ページの『Wait (4)』
- 203 ページの『Start Host Notification (23)』
- 204 ページの『Start Close Intercept (41)』
- 205 ページの『Start Keystroke Intercept (50)』
- 206 ページの『Send File (90)』
- 207 ページの『Receive File (91)』

WinHLLAPI の非同期関数

以下のセクションでは、WinHLLAPI の非同期関数について説明します。

WinHLLAPIAsync

このエントリー・ポイントは、完了するのに長時間を要することの多い WinHLLAPI の 6 つの関数用に使用されます。 WinHLLAPIAsync を用いると、関数は非同期的に立ち上げられ、呼び出しアプリケーションの進行が続行するのを妨げることがありません。この 6 つの関数とは、**Wait** (04)、 **Start Host Notify** (23)、 **Start Close Intercept** (41)、 **Start Keystroke Intercept** (50)、 **Send File** (90)、および **Receive File** (91) であり、これらについて『第 4 章 WinHLLAPI 拡張関数』で説明します。

HANDLE WinHLLAPIAsync (HWIND hWnd, LPWORD *lpnFunction*, LPBYTE *lpData*, LPWORD *lpnLength*, LPWORD *lpnRetC*)*

パラメーター・リストは、関数番号の前にウィンドウ・ハンドルが必要であること以外は WinHLLAPI と同じです。関数は非同期で動作するため、関数の完了は登録済みメッセージによって通知されます。ウィンドウ・ハンドルは、メッセージのターゲットとして必須です。

WinHLLAPIAsync (90 と 91 以外のすべての関数の場合) と **WinHLLAPIAsyncFileTransfer** (関数 90 と 91 の場合) のストリングを用いた **RegisterWindowsMessage()** の呼び出しを通して、WinHLLAPI アプリケーションによって登録する必要のあるメッセージが 2 つあります。その標準形式は次のとおりです。

WPARAM
元の関数呼び出しによって戻されるタスク・ハンドルが入ります。

LPARAM

高位ワードにはエラー・コードが入り、低位ワードには元の関数番号が入ります。

Wait (4)

この関数は、ホスト・セッションが使用禁止状態にあるかどうかを判別します。何らかの理由によって関数が使用禁止状態になっている場合、この関数は、使用禁止状態の期限が切れた時点またはユーザーの待機期限が切れた時に、メッセージでアプリケーションに知らせます。待機する時間は、**Set Session Parameters (9)** 関数を用いて設定されます。

前提関数: Connect Presentation Space (1)

WinHLLAPIAsync (*hWnd, lpwFunction, lpbyString, lpwLength, lpwReturnCode*)

呼び出しパラメーター:

パラメーター	説明
データ・ストリング	NA
データ長	NA
PS 位置	NA

戻りコード:

コード	説明
WHLLOK	PS は使用禁止が解除されており、入力できます。
WHLLNOTCONNECTED	ユーザーの WinHLLAPI アプリケーションが、有効なホスト・セッションに接続されていません。
WHLLPSBUSY	使用禁止状態である間に関数がタイムアウトになりました。
WHLLNHIBITED	PS は使用禁止になっています。
SHLLSYSERROR	システム・エラーが原因で、関数が失敗しました。
WHLLCANCEL	非同期関数が取り消されました。

注釈: Asynchronous Wait は、PS の使用禁止状態の期限が切れる時点を呼び出しアプリケーションに通知するために使用されます。使用禁止状態の期限が切れると、このバージョンの **Wait** は、**WinHLLAPIAsync** メッセージを、*hWnd* で指定されたウィンドウに通知します。セッション・オプションの **TWAIT**、 **LWAIT**、および **NWAIT** は、この関数が待機する時間の長さに影響を及ぼします。これらのセッション・オプションの詳細については、 165 ページの『Set Session Parameters (9)』を参照してください。

注: セッション・パラメーターで **NWAIT** が指定され、アプリケーションが WinHLLAPI の改訂版 1.1 のインプリメンテーションを使用して登録する場合、**WINHLLAPIAsync** 呼び出しは、**WinHLLAPI** 呼び出しと同じ働きをし、メッセージは送信しません。改訂版 1.0 が使用されている場合、**Wait** は、PS の使用禁止状況とともに即時にメッセージを戻します。

Start Host Notification (23)

この関数を使用すると、WinHLLAPI アプリケーションに、ホスト・セッション表示スペース (PS) またはオペレーション情報域 (OIA) 内の変更を通知することができます。

前提関数: この関数には前提関数はありません。

WinHLLAPIAsync (*hWnd, lpwFunction, lpbyString, lpwLength, lpwReturnCode*)

呼び出しパラメーター:

パラメーター	説明
データ・ストリング	7 バイトのストリング。形式は次のとおりです。 **バイト 1** 希望するホスト・セッションの短縮名セッション ID、あるいは現行のホスト・セッションの場合はスペースまたは NULL。 **バイト 2** 通知モード。 ″P″ (表示スペースの更新のみの場合)、″O″ (OIA の 更新のみの場合)、″B″ (表示スペースと OIA の両方の更新の場合)。 WinHLLAPIAsync の呼び出し時、このバイト位置は ″A″ にすることができます。 **バイト 3 ～ 6** 使用されません。旧アプリケーションとの互換性を持たせるために用意されています。 **バイト 7** 予約済み、または WinHLLAPIAsync を使用していてバイト 2 が A である場合には、次のいずれかに置き換えられます。すなわち、表示スペースの更新のみの場合には P、OIA の更新のみの場合には O、表示スペースと OIA の両方の更新の場合には B のいずれか。
データ長	ホスト・イベント・バッファーの長さ (推奨値は 256)
PS 位置	NA

戻りパラメーター:

パラメーター	説明
データ・ストリング	呼び出し時のデータ・ストリング と同じです。

戻りコード:

コード	説明
WHLLOK	ホスト通知が使用可能になっています。
WHLLNOTCONNECTED	指定されたホスト・セッションが無効です。
WHLLPARAMETERERROR	1 つまたは複数のパラメーターが無効です。

コード	説明
WHLLSYSERROR	システム・エラーが原因で、関数が失敗しました。
WHLLCANCEL	非同期関数が取り消されました。

注釈: ホスト通知は、いったん使用可能になると、**Stop Host Notification (25)** または **WinHLLAPICancelAsyncRequest()** を呼び出すまで使用可能のままです。この関数は、ホスト通知を開始し、即時に制御をユーザーの Windows HLLAPI アプリケーションに戻します。これにより、ユーザーのアプリケーションは、ホストの更新を待機している間に他のタスクを実行することができます。更新が発生すると、この関数は、*hWnd* によって指定されたウィンドウに登録済みメッセージの **WinHLLAPIAsync** を表示して知らせます。

Start Close Intercept (41)

この関数は、ユーザー要求を代行受信して、パーソナル・コミュニケーションズをクローズします。

前提関数: この関数には前提関数はありません。

WinHLLAPIAsync (*hWnd, lpwFunction, lpbyString, lpwLength, lpwReturnCode*)

呼び出しパラメーター:

パラメーター	説明
データ・ストリング	戻されるセマフォー・アドレス用の 5 バイトのストリング。先頭バイトは、照会を行うセッションのセッション短縮名、もしくは現行セッションの場合はスペースまたは NULL。
データ長	必ず指定してください。
PS 位置	NA

戻りパラメーター:

パラメーター	説明
データ・ストリング	5 バイトのストリング。形式は次のとおりです。 **バイト 1** セッション短縮名、もしくは現行セッションの場合のスペースまたは NULL。 **バイト 2 〜 5** セマフォー・アドレス。

戻りコード:

コード	説明
WHLLOK	関数は正常終了しました。
WHLLNOTCONNECTED	無効な表示スペースが指定されました。
WHLLPARAMETERERROR	無効なオプションが指定されました。

コード	説明
WHLLSYSERROR	システム・エラーが原因で、関数が失敗しました。
WHLLCANCEL	非同期関数が取り消されました。

注釈: ホスト通知は、いったん使用可能になると、**Stop Close Intercept (43)** または **WinHLLAPICancelAsyncRequest ()** を呼び出すまで使用可能のままです。最初から、セマフォーは設定されています。この関数の使用後、ユーザーからのクローズ要求が破棄され、セマフォーは消去されます。

この関数は、クローズ代行受信を開始し、即時に制御をユーザーの Windows HLLAPI アプリケーションに戻します。これにより、ユーザーのアプリケーションは、クローズ要求を待機している間に他のタスクを実行することができます。クローズ要求が発生すると、この関数は、*hWnd* によって指定されたウィンドウに登録済みメッセージ **WinHLLAPIAsync** を表示して知らせます。

Start Keystroke Intercept (50)

この関数は、ユーザーによってセッションに送信されたキー・ストロークを代行受信します。

前提関数: この関数には前提関数はありません。

WinHLLAPIAsync (*hWnd, lpwFunction, lpbyString, lpwLength, lpwReturnCode***)**

呼び出しパラメーター:

パラメーター	説明
データ・ストリング	6 バイトのストリング。形式は次のとおりです。 **バイト 1** セッション短縮名、もしくは現行ホスト・セッションの場合のスペースまたは NULL。 **バイト 2** キー・ストローク代行受信コード。 ″D″ では、AID キー・ストロークだけが代行受信され、 ″L″ では、すべてのキー・ストロークが代行受信されます。 **バイト 3 〜 6** 予約済み
データ長	変数 (推奨値は 256)
PS 位置	NA

戻りコード:

コード	説明
WHLLOK	キー・ストローク代行受信は開始されています。

コード	説明
WHLLNOTCONNECTED	ホスト・セッションの表示スペースが無効です。
WHLLPARAMETERERROR	1 つ以上のパラメーターが無効です。
WHLLPSBUSY	セッションは使用中です。
WHLLSYSERROR	システム・エラーが原因で、関数が失敗しました。
WHLLCANCEL	非同期関数が取り消されました。

注釈: この関数は、キー・ストローク代行受信を開始し、即時に制御をユーザーの Windows HLLAPI アプリケーションに戻します。これにより、ユーザーのアプリケーションは、キー・ストロークを待機している間に他のタスクを実行することができます。この関数は開始されると、ユーザーが PS にキーを送信するたびに、*hWnd* によって指定されたウィンドウに **WinHLLAPIAsync** メッセージを表示して知らせます。通知後、代行受信されたキー・ストロークは、通常の EHLLAPI アプリケーションで使用できる任意の方法で処理することができます。キー・ストローク・バッファーのサイズは制限付きであるため、各キー・ストロークは処理してバッファーから除去するようにしてください。

Send File (90)

この関数は、PC からホストにファイルを転送します。

前提関数: この関数には前提関数はありません。

WinHLLAPIAsync (*hWnd, lpwFunction, lpbyString, lpwLength, lpwReturnCode*)

呼び出しパラメーター:

パラメーター	説明
データ・ストリング	SEND コマンド・パラメーター。
データ長	*Data String* の長さ。セッション・オプション EOT が指定されている場合は NA。
PS 位置	NA

戻りコード:

コード	説明
WHLLOK	ファイル転送は正常に開始されました。
WHLLPARAMETERERROR	パラメーター・エラー、もしくは データ長 がゼロかまたは 255 より大きくなっています。
WHLLFTXCOMPLETE	ファイル転送は完了しました。
WHLLFTXSEGMENTED	転送は完了し、レコードはセグメント化されました。
WHLLSYSERROR	システム・エラーが原因で、関数が失敗しました。

コード	説明
WHLLTRANSABORTED	ユーザーが「取り消し」ボタンをクリックしたか、あるいはタイムアウト期間が過ぎたことが原因で、ファイル転送が異常終了しました。
WHLLFILENOTFOUND	PC ファイルが見つかりません。
WHLLFTXCOMPLETECICS	ファイル転送 (CICS への転送) は正常終了しました。
WHLLACCESSDENIED	PC ファイルへのアクセスが拒否されました。
WHLLMEMORY	メモリーが不足しています。
WHLLINVALIDENVIRONMENT	環境が無効です。

注釈: 接続済みホスト・セッションごとにサポートされるファイル転送は 1 つだけです。

この関数は、ファイル転送を開始し、即時に制御をユーザーの Windows HLLAPI アプリケーションに戻します。これにより、ユーザーのアプリケーションは、ファイル転送が行われている間に他のタスクを実行することができます。この関数はいったん開始されると、**WinHLLAPIAsyncFileTransfer** メッセージを *hWnd* で指定されたウィンドウに定期的に表示します。これらのメッセージは、WinHLLAPI アプリケーションに転送の状況を知らせ、転送の完了時には最終メッセージを送信します。

wParm
状況標識です。高位バイトには セッション ID が入り、低位バイトには状況が入ります。低位バイトがゼロの場合、ファイル転送はまだ進行中です。低位バイトが 1 の場合、ファイル転送は完了しています。

lParm *wParm* の低位バイトがゼロ (進行中) の場合、*lParm* は、転送されたバイトの数です。*wParm* の低位バイトが 1 (完了済み) の場合、*lParm* は完了コードです。

Receive File (91)

この関数は、PC からホストにファイルを転送します。

前提関数: この関数には前提関数はありません。

WinHLLAPIAsync (*hWnd, lpwFunction, lpbyString, lpwLength, lpwReturnCode*)

呼び出しパラメーター:

パラメーター	説明
データ・ストリング	RECEIVE コマンド・パラメーター。
データ長	*Data String* の長さ。セッション・オプション EOT が指定されている場合は NA。
PS 位置	NA

戻りコード:

コード	説明
WHLLOK	ファイル転送は正常に開始されました。
WHLLPARAMETERERROR	パラメーター・エラー、もしくは データ長 がゼロかまたは 255 より大きくなっています。
WHLLFTXCOMPLETE	ファイル転送は完了しました。
WHLLFTXSEGMENTED	転送は完了し、レコードはセグメント化されました。
WHLLSYSERROR	システム・エラーが原因で、関数が失敗しました。
WHLLTRANSABORTED	ユーザーが「取り消し」ボタンをクリックしたか、あるいはタイムアウト期間が過ぎたことが原因で、ファイル転送が異常終了しました。
WHLLFILENOTFOUND	PC ファイルが見つかりません。
WHLLFTXCOMPLETECICS	ファイル転送 (CICS への転送) は正常終了しました。
WHLLACCESSDENIED	PC ファイルへのアクセスが拒否されました。
WHLLMEMORY	メモリーが不足しています。
WHLLINVALIDENVIRONMENT	環境が無効です。

注釈: 接続済みホスト・セッションごとにサポートされるファイル転送は 1 つだけです。

この関数は、ファイル転送を開始し、即時に制御をユーザーの Windows HLLAPI アプリケーションに戻します。これにより、ユーザーのアプリケーションは、ファイル転送が行われている間に他のタスクを実行することができます。この関数はいったん開始されると、**WinHLLAPIAsyncFileTransfer** メッセージを *hWnd* で指定されたウィンドウに定期的に表示します。これらのメッセージは、WinHLLAPI アプリケーションに転送の状況を知らせ、転送の完了時には最終メッセージを送信します。

wParm
状況標識です。高位バイトには セッション ID が入り、低位バイトには状況が入ります。低位バイトがゼロの場合、ファイル転送はまだ進行中です。低位バイトが 1 の場合、ファイル転送は完了しています。

lParm *wParm* の低位バイトがゼロ (進行中) の場合、*lParm* は、転送されたバイトの数です。*wParm* の低位バイトが 1 (完了済み) の場合、*lParm* は完了コードです。

WinHLLAPICancelAsyncRequest

この関数は、**WinHLLAPIAsync()** 呼び出しによって立ち上げられた未解決の非同期関数を取り消します。

構文

int WinHLLAPICancelAsyncRequest (HANDLE *hAsyncTask*, WORD *wFunction*)

パラメーター

hAsyncTask
関数が開始されたときに WinHLLAPIAsync() によって戻されるハンドル。

wFunction
取り消す非同期タスクの関数番号。このパラメーターは改訂版 1.1 では必須ですが 1.0 では必須ではなく、オプションです。

この関数を使用すると、それ以前に WinHLLAPIAsync() の呼び出しによって開始された非同期タスクはすべて、未解決の間に取り消されることがあります。

戻り

戻り値は、指定された関数が実際に取り消されたかどうかを示します。関数が取り消されている場合、戻り値は WHLLOK (0) となります。未解決の非同期関数が取り消されていない場合、以下のコードのいずれかが戻されます。

WHLLINVALID
hAsyncTask は、有効なタスク・ハンドルではありません。

WHLLALREADY
hAsyncTask によって指定された非同期タスクはすでに完了しています。

初期化関数と終了関数

以下の項では、WinHLLAPI プログラミング・サポートの初期化関数と終了関数について説明します。

WinHLLAPI Startup

この関数は、WinHLLAPI インプリメンテーションを用いてアプリケーションを登録する場合に使用し、他のどの WinHLLAPI インプリメンテーションの呼び出しよりも前に呼び出すことが必要です。このインプリメンテーションは、WinHLLAPI 仕様のバージョン 1.0 と 1.1 をサポートします。 WinHLLAPI アプリケーションは、この関数とのバージョンの互換性を持つようにする必要があります。

構文

int WinHLLAPIStartup(WORD *wVersionRequired,* LPWHLLAPIDATA *lpData*)

パラメーター

wVersionRequired
これは、WinHLLAPI アプリケーションが必要とするバージョンです。低位バイトにはメジャー・バージョン番号が入り、高位バイトにはマイナー・バージョン (改訂) 番号が入ります。

lpData これは、インプリメンテーションのバージョン番号と、WinHLLAPI インプリメンテーションのプロバイダーを記述しているストリングを受け取る WHLLAPIDATA 構造を指すポインターです。 WHLLAPIDATA 構造は次のように定義されます。

```
#define  WHLLDESCRIPTION_LEN   127
typedef  struct  tagWHLLAPIDATA
{
     WORD  wVersion;
     Char  szDescription[WHLLDESCRIPTION_LEN + 1];
}WHLLAPIDATA,  *  PWHLLAPIDATA,  FAR  *LPWHLLAPIDATA;
```

戻り

戻り値は、インプリメンテーションによる WinHLLAPI アプリケーションの登録が正常終了したか失敗したかを示します。登録が正常終了している場合、戻り値は WHLLOK (ゼロ) になります。正常終了していない場合は、次のいずれかになります。

WHLLSYSNOTREADY
基盤となるネットワーク・サブシステムが使用不可であることを示します。

WHLLVERNOTSUPPORTED
このインプリメンテーションが、要求されたバージョンを提供していないことを示します。このインプリメンテーションがサポートするのは、バージョン 1.0 と 1.1 だけです。

WinHLLAPI Cleanup

WinHLLAPI 仕様では、WinHLLAPI がこの関数を使用して WinHLLAPI インプリメンテーションによる登録を解除することをお勧めしています。

構文

BOOL WinHLLAPICleanup()

戻り

登録解除が正常終了した場合は、TRUE が戻されます。正常終了しなかった場合は FALSE が戻されます。

ブロッキング・ルーチン

以下のセクションでは、WinHLLAPI プログラミングによってサポートされるブロッキング・ルーチンについて説明します。

注: ブロッキング・ルーチンは WinHLLAPI 準拠のためにサポートされていますが、これらのルーチンの使用はお勧めしません。非同期処理のためには、WinHLLAPIAsync 関数の使用をお勧めしています。

WinHLLAPIIsBlocking

この関数は、呼び出し WinHLLAPI アプリケーションのスレッドに、ブロッキング呼び出しの実行が進行中であるかどうかを通知します。ブロッキング呼び出しとは、実行に長い時間がかかり、完了するまで戻らない同期関数のことです。この WinHLLAPI のインプリメンテーションには、5 つのブロッキング呼び出しがあります。そのブロッキング呼び出しは、**Get Key (51)**、 **Wait (4)**、**Pause (18)**、 **Send File (90)**、および **Receive File (91)** です。

構文

BOOL WinHLLAPIIsBlocking()

戻り

WinHLLAPI アプリケーションのスレッドがブロッキング呼び出しの中間にある場合、関数は TRUE を戻し、中間にない場合には FALSE を戻します。

注釈

デフォルトのブロッキング・フックを使用するとブロッキング呼び出し時にメッセージを処理することができるため、再度ブロッキング呼び出しを呼び出すことが可能です。

WinHLLAPISetBlockingHook

この関数はアプリケーションで定義済みのプロシージャーを、ブロッキング呼び出しの完了を待機している間に実行するよう設定します。ブロッキング呼び出しとは、実行に長い時間がかかり、完了するまで戻らない同期関数のことです。この WinHLLAPI のインプリメンテーションには、5 つのブロッキング呼び出しがあります。そのブロッキング呼び出しは、**Get Key (51)**、 **Wait (4)**、**Pause (18)**、 **Send File (90)**、および **Receive File (91)** です。

構文

FARPROC WinHLLAPISetBlockingHook(FARPROC *lpfnBlockingHook***)**

パラメーター

lpfnBlockingHook
これは、新規ブロッキング・プロシージャーを指すポインターです。

説明

WinHLLAPI インプリメンテーションには、メッセージ・ハンドラーそのもので構成されているデフォルトのブロッキング・プロシージャーがあります。このデフォルトの機構を以下の例に示します。

```
BOOL    DefaultBlockingHook
{
      MSG msg;

      if (PeekMessage (&msg, NULL, 0, 0, xfPM_NOREMOVE))
      {
            if(msg.message = = WM_QUIT)
            {
                return FALSE;
            }
            PeekMessage (&msg, NULL, 0, 0, PM_REMOVE);
            TranslateMessage (&msg);
            DispatchMessage (&msg);
      }
return TRUE;
}
```

ブロッキング・フックは、スレッドごとに実装されます。この関数によって設定されるブロッキング・フックは、別の **WinHLLAPISetBlockingHook()** 呼び出しによっ

て置き換えられるまで、もしくは **WinHLLAPIUnhookBlockingHook()** の呼び出しによってデフォルトが復元されるまで、そのスレッドに対して有効のままとなります。

ブロッキング関数は、**WM_QUIT** メッセージを受け取った場合は **FALSE** を戻し、WinHLLAPI がアプリケーションに制御を戻してメッセージを処理し、正常に終了できるようにする必要があります。上記以外の場合には、関数は **TRUE** を戻す必要があります。

戻り

この関数は、置き換えられるブロッキング関数を指すポインターを戻します。

WinHLLAPIUnhookBlockingHook

この関数は、呼び出しスレッド用にデフォルトのブロッキング・フックを復元します。

構文

BOOL WinHLLAPIUnhookBlockingHook()

戻り

この関数は、デフォルトのブロッキング機構が正常に復元された場合には TRUE を戻し、そうでない場合には FALSE を戻します。

WinHLLAPICancelBlockingCall

この関数は、*現行スレッド* 内のブロッキング呼び出しの実行を取り消します。ブロッキング呼び出しとは、実行に長い時間がかかり、完了するまで戻らない同期関数のことです。この WinHLLAPI のインプリメンテーションには、5 つのブロッキング呼び出しがあります。そのブロッキング呼び出しは、**Get Key** (51)、 **Wait** (4)、**Pause** (18)、 **Send File** (90)、および **Receive File** (91) です。これらのブロッキング呼び出しのいずれかが取り消されると、その取り消された関数は WHLLCANCEL を戻します。

構文

int WinHLLAPICancelBlockingCall()

戻り

戻り値は、指定された関数が実際に取り消されたかどうかを示します。関数が取り消されている場合、戻り値は WHLLOK (0) となります。未解決のブロッキング関数がない場合は、以下の戻りコードが戻されます。

WHLLINVALID
現在実行されているブロッキング呼び出しがないことを示します。

第 5 章 PCSAPI 関数

パーソナル・コミュニケーションズでは、API のセット (本書では *PCSAPI* として定義されています) が提供されています。セッションが確立された後、ワークステーションのアプリケーション・プログラムとホスト・システムとの間の対話を管理する場合には EHLLAPI が使用されますが、パーソナル・コミュニケーションズ・セッション自体を制御する場合は、PCSAPI を使用することができます。

PCSAPI の使用方法

C または C++ で PCSAPI を使用してアプリケーション・プログラムを作成することができます。 PCSAPI アプリケーションを開発するには、以下を行ってください。

1. ソース・コードを用意して、適切な PCSAPI 呼び出しを追加してください。
2. アプリケーションにヘッダー・ファイル PCSAPI.H を入れてください。
3. ソース・コードをコンパイルしてください。
4. コンパイルしてできた .OBJ ファイルを、適切なオブジェクト・ファイルまたはライブラリーにリンクしてください。

   また、PCSAPI インポート・ライブラリーの PCSCALLS.LIB (16 ビットの場合) および PCSCAL32.LIB (32 ビットの場合) も、このファイルにリンクしてください。

ページ・レイアウトについての規則

すべての PCSAPI 関数呼び出しは、必要な情報をす早く検索できるように同一の形式で表されています。形式は次のとおりです。

関数名

関数の形式

パラメーターの形式および説明

戻りコード

関数の形式

「関数の形式」では、関数の形式を次のように示します。

TYPE FunctionName(*TYPE Parameter1, ...***)**

パラメーターの形式および説明

「パラメーターの形式および説明」には、PCSAPI 関数呼び出しで指定する各パラメーターの形式と説明のリストが示されています。

戻りコード

「戻りコード」には、PCSAPI 関数を呼び出した後、呼び出し元プログラムが受け取るコードをリストします。

pcsConnectSession

3270	*5250*	*VT*
YES	YES	YES

pcsConnectSession 関数は、短縮セッション ID を指定してホスト・セッションとの通信を開始します。ホスト・セッションは、すでに開始している必要があります。この呼び出しは、エミュレーター・セッション・パネル上の「**通信**」→「**接続**」メニュー項目と同等です。

関数の形式

BOOL WINAPI pcsConnectSession(*char cShortSessionID***)**

パラメーターの形式および説明

char cShortSessionID
表示スペースの短縮セッション ID。

戻りコード

戻りコード	意味
TRUE	関数が正常終了しました。
FALSE	次のいずれかを示します。 • セッションが開始されていません。 • 無効なセッション ID が指定されました。 • 呼び出しが失敗しました。

pcsDisconnectSession

3270	*5250*	*VT*
YES	YES	YES

pcsDisconnectSession 関数は、短縮セッション ID を指定してホスト・セッションとの通信を切断します。この関数は、リンクを切断するだけで、セッションを停止するものではありません。この呼び出しは、エミュレーター・セッション・パネル上の「**通信**」→「**切断**」メニュー項目と同等です。

関数の形式

BOOL WINAPI pcsDisconnectSession(*char cShortSessionID***)**

パラメーターの形式および説明

char cShortSessionID
表示スペースの短縮セッション ID。

戻りコード

戻りコード	意味
TRUE	関数が正常終了しました。
FALSE	次のいずれかを示します。 • セッションが開始されていません。 • 無効なセッション ID が指定されました。 • 呼び出しが失敗しました。

pcsQueryConnectionInfo

3270	*5250*	*VT*
YES	NO	NO

pcsQueryConnectionInfo 関数は、指定されたホスト・セッションの Telnet 接続についての情報を戻します。戻り情報は、アプリケーションが提供するバッファーに戻されます。

関数の形式

BOOL WINAPI pcsQueryConnectionInfo(*char cShortSessionID, CONNECTIONINFO *ConnectionInfo***)**

パラメーターの形式および説明

char cShortSessionID
表示スペースの短縮セッション ID。

CONNECTIONINFO *ConnectionInfo
接続 info データが戻される CONNECTIONINFO 構造へのポインター。

戻りコード

戻りコード	意味
TRUE	関数が正常終了しました。
FALSE	次のいずれかを示します。 • セッションが開始されていません。 • 無効なセッション ID が指定されました。 • 指定されたセッションはこの API (Telnet ではない) 用にサポートされている接続タイプではありませんでした。

ConnectionInfo

CONNECTIONINFO 構造体には、下記の情報からなるホスト接続についての情報が入れられます。

構造体	情報
ホスト名	現在接続されている Telnet ホストの名前を示します。
LU 名	現在割り当てられている LU 名を示します。
ポート番号	接続で使用される予定のホスト・ポート番号を示します。
SSL 標識	セキュア接続を示します (1 = セキュア、0 = 非セキュア)。

注: この API は PCSAPI の 32 ビット・バージョンでのみ有効で、Telnet 接続用にのみ機能します。

例

```
typedef struct_CONNECTIONINFO
{ //Description of a connection @WD06A
   char hostName[63];    //telnet host name      @WD06A
   char reserved[1];     //reserved              @wD06A
   int portNumber;       //host port number      @WD06A
   char luName[17];      //LU name               @WD06A
   char reserved2[3];    //reserved              @WD06A
   BOOL sslIndicator;    //Secure Connection     @WD06A
                           indicator
   char reserved3[256];  //reserved              @WD06A
}CONNECTIONINFO;
```

pcsQueryEmulatorStatus

3270	*5250*	*VT*
YES	YES	YES

pcsQueryEmulatorStatus 関数は、短縮セッション ID を指定してホスト・セッションの状況を戻します。

関数の形式

ULONG WINAPI pcsQueryEmulatorStatus(*char cShortSessionID***)**

パラメーターの形式および説明

char cShortSessionID
表示スペースの短縮セッション ID。

戻りコード

戻りコードの値は、ビットごとに処理しなければなりません。つまり、以下の値のうち 1 つか、または以下の値の論理 OR 演算によって処理する必要があります。

戻りコード	値	意味
PCS_SESSION_STARTED	0x00000001	指定されたセッションが開始されました。このビットがオフの場合は、指定されたセッションが開始されていないか、または無効なセッション ID が指定されています。
PCS_SESSION_ONLINE	0x00000002	指定されたセッションがオンライン (接続されている) です。このビットがオフの場合は、指定されたセッションがオフライン (切断されている) です。
PCS_SESSION_API_ENABLED	0x00000004	API (EHLLAPI、DDE) は指定されたセッションで使用可能です。このビットがオフの場合は、API はこのセッションで使用不可です。

pcsQuerySessionList

3270	*5250*	*VT*
YES	YES	YES

pcsQuerySessionList 関数は、現在のホスト・セッションすべてのリストを戻します。アプリケーションは PCSAPI.H ファイルに定義されているとおりの SESSINFO 構造体の配列と、配列内のエレメント数を指定する必要があります。この関数はその構造体に、各セッションに関する情報を入れ、見付かったセッションの数を戻します。

ホスト・セッションのエレメント数よりも配列のエレメント数の方が少ない場合は、用意された配列のエレメントだけにデータが入れられます。配列が小さすぎる場合でも、この関数は常に実際のセッションの数を戻します。

配列エレメント数をゼロとしてアプリケーションがこの関数を呼び出して、存在するセッションの数だけを判別することもできます。それから改めて呼び出しを行って、セッション情報を取得することができます。

関数の形式

ULONG WINAPI pcsQuerySessionList(*ULONG Count, SESSINFO *SessionList***)**

パラメーターの形式および説明

ULONG Count
SessionList 配列内のエレメントの数。

SESSINFO *SessionList
PCSAPI.H に定義されている SESSINFO 構造体の配列へのポインター。

戻りパラメーター

戻りコード
パーソナル・コミュニケーションズ・セッションの合計数。この値は Count パラメーターよりも大きいことも小さいこともあります。

SessionList
SESSINFO 構造体の配列には、ホスト・セッションに関する情報が入れられます。セッションは任意の順序でリスト内に配置されます。各 SESSINFO 構造体には、以下のフィールドがあります (PCSAPI32.H に定義されています)。

Name char とセッション ID (A-Z) を含む ULONG の共用体。パーソナル・コミュニケーションズの現在のインプリメンテーションでは、下位バイト (char) のみが使用され、他のバイトはゼロとして戻されます。

Status セッションの現在の状況を示すビット・フラグの組み合わせ。フラグ (PCS_SESSION_*) の定義は次の表のとおりです。

状況コードの値は、ビットごとに処理しなければなりません。つまり、以下の値のうち 1 つか、または以下の値の論理 OR 演算によって処理する必要があります。

戻りコード	意味
PCS_SESSION_STARTED	セッションは実行中です。このフラグが立っていなければ、その他はすべて未定義です。
PCS_SESSION_ONLINE	セッションはホストへの通信を確立しました (つまり、セッションは接続されています)。
PCS_SESSION_API_ENABLED	セッションではプログラミング API が使用可能です。このフラグが立っていなければ、このセッションには EHLLAPI およびホスト・アクセス・クラス・ライブラリー API は使えません。

例

```
ULONG      NumSessions, i;  // Session counters
SESSINFO   *SessList;       // Array of session information structures
// Find out number of sessions that exist
NumSessions = pcsQuerySessionList (0,NULL);
if (NumSessions == 0) {
   printf("There are no sessions.");
   exit;
}

// Allocate array large enough for all sessions
SessList = (SESSINFO *)malloc(NumSessions * sizeof(SESSINFO));
```

```
memset(SessList, 0x00, NumSessions * sizeof(SESSINFO));

// Now read actual session info
pcsQuerySessionList(NumSessions, SessList);

for (i=0; i<NumSessions; i++) {
   if ((SessList[i].Status & PCS_SESSION_STARTED) &&
       (SessList[i].Status & PCS_SESSION_ONLINE))  {

      printf("Session %c is started and connected.",
        SessList[i].Name.ShortName);
   }
}

exit;
```

pcsQueryWorkstationProfile

3270	*5250*	*VT*
YES	YES	YES

pcsQueryWorkstationProfile 関数は、ホスト・セッションを呼び出すために使用されるワークステーション・プロファイル名を戻します。ホスト・セッションを指定するためには、短縮セッション ID を使用する必要があります。ワークステーション・プロファイル名は、アプリケーションが用意する作業バッファーにコピーされます。

関数の形式

BOOL WINAPI pcsQueryWorkstationProfile(*char cShortSessionID, PSZ lpBuffer***)**

パラメーターの形式および説明

char cShortSessionID

表示スペースの短縮セッション ID。

PSZ lpBuffer

NULL 文字で終了するワークステーション・プロファイル名をコピーするための作業バッファー。バッファーは、完全修飾ファイル名を含むことができるだけの大きさでなければなりません。

戻りコード

戻りコード	意味
TRUE	関数が正常終了しました。
FALSE	次のいずれかを示します。 • セッションが開始されていません。 • 無効なセッション ID が指定されました。

pcsSetLinkTimeout

3270	*5250*	*VT*
YES	YES	YES

pcsSetLinkTimeout 関数は、SSCP 所有の Telnet リンクのアイドル・タイムアウトを設定します。この関数は、非 TN 接続、または SSCP 所有状態にない接続に対しては影響を及ぼしません。タイムアウト値がゼロに設定されると、そのリンクはタイムアウトになりません。ゼロ以外に設定された場合、SSCP 所有状態でアイドル状態になってから、指定された分数が経過すると、タイムアウトになります (切断されます)。

関数の形式

ULONG WINAPI pcsSetLinkTimeout(*char cShortSessionID, USHORT Timeout***)**

パラメーターの形式および説明

char cShortSessionID
表示スペースの短縮セッション ID。

USHORT Timeout
タイムアウト値は分で指定します。ゼロという値は、タイムアウトの使用不可を意味します。

戻りコード

戻りコード	意味
PCS_SUCCESSFUL	関数が正常終了しました。
PCS_SYSTEM_ERROR	システム・エラーが発生しました。

pcsStartSession

3270	*5250*	*VT*
YES	YES	YES

pcsStartSession 関数は、指定されたワークステーション・プロファイルを使用してホスト・セッションを開始します。短縮セッション ID を指定することも可能です。

関数の形式

ULONG WINAPI pcsStartSession(*PSZ lpProfile, char cShortSessionID, USHORT fuCmdShow***)**

パラメーターの形式および説明

PSZ lpProfile
ロードするプロファイルのパスと完全なファイル名。パスは任意指定ですが、完全ファイル名の指定は必須です (.ws は想定されないので、拡張子も指定してください)。

char cShortSessionID
表示スペースの短縮セッション ID。スペースまたは NULL は次に使用可能なセッション ID を示します。

USHORT fuCmdShow
ウィンドウをどのように表示させるかの指定。 PCSAPI.H からの次のいずれかの値。

- PCS_HIDE
- PCS_SHOW
- PCS_MINIMIZE
- PCS_MAXIMIZE

戻りコード

戻りコード	値	意味
PCS_SUCCESSFUL	0	関数が正常終了しました。
PCS_INVALID_ID	1	無効なセッション ID が指定されました。
PCS_USED_ID	2	指定された短縮セッション ID はすでに使用されています。
PCS_INVALID_PROFILE	3	指定したワークステーション・プロファイルでエラーが発生したか、またはウィンドウ・パラメーターが正しくありません。
PCS_SYSTEM_ERROR	9	システム・エラーが発生しました。

pcsStopSession

3270	*5250*	*VT*
YES	YES	YES

pcsStopSession 関数は、短縮セッション ID によって指定されたホスト・セッションを停止します。

関数の形式

BOOL WINAPI pcsStopSession(*char cShortSessionID, USHORT fuSaveProfile***)**

パラメーターの形式および説明

char cShortSessionID
表示スペースの短縮セッション ID。

USHORT fuSaveProfile

このパラメーターは、次の値のいずれかです。

fuSaveProfile	値	意味
PCS_SAVE_AS_PROFILE	0	現行のプロファイルの指定に従って保管
PCS_SAVE_ON_EXIT	1	終了時にプロファイルを保管。
PCS_NOSAVE_ON_EXIT	2	終了時にプロファイルを保管しない。

戻りコード

戻りコード	意味
TRUE	関数が正常終了しました。
FALSE	次のいずれかを示します。 • セッションが開始されていません。 • 無効なセッション ID が指定されました。

ページ設定関数

このセクションで説明する各種 PCSAPI 関数を使用すると、パーソナル・コミュニケーションズのエミュレーター・セッションの「**ページ設定**」の設定値を制御および検索できます。

制約事項

以下の制約事項を順守していない場合は、API が失敗します。戻りコードに、障害の理由が示されます。

- 引き数 cShortSessionID に指定されるホスト・セッションが PDT モードになっていてはならない。
- API の呼び出し時に、ホスト・セッションが印刷中であってはならない。
- 「**ファイル**」→「**ページ設定**」ダイアログが使用中であってはならない。

PAGEINFO 構造体の一部のメンバーは、特定のセッション・タイプについてのみ有効、またはサポートされている場合があります。制約事項が指定されていない場合、そのメンバーは以下のセッション・タイプで有効またはサポートされています。

- 3270 表示装置
- 3270 印刷装置
- 5250 表示装置
- ASCII VT

5250 プリンター・セッションはサポートされていません。

注: これらの関数は、現在、DBCS セッションおよび双方向セッションについてはサポートされていません。

pcsGetPageSettings

3270	*5250*	*VT*
YES	YES	YES

pcsGetPageSettings 関数は、ホスト・セッションのページ設定を検索します (「**ファイル**」→ 「**ページ設定**」ダイアログでの設定と同様)。ダイアログの「**テキスト**」タブ内の設定のみサポートされます。

関数の形式

ULONG WINAPI pcsGetPageSettings(*char cShortSessionID, PAGEINFO * const pPageInfo, ULONG * const pErrorInfo***)**

パラメーターの形式および説明

char cShortSessionID
表示スペースの短縮セッション ID。

PAGEINFO * const pPageInfo
ページ設定が戻される PAGEINFO 構造体へのポインター。

nFlags 構造体の有効なメンバーを示す、ビット・フラグの組み合わせ。これらのフラグを単独で使用するか、論理和演算で処理することにより、プロパティー・ページ (PCSAPI32.H に定義されています) を復元することができます。各フラグと、対応する構造体内の有効なメンバーは以下のとおりです。

フラグ	構造体内の有効なメンバー
PCS_PAGE_CPI	nCPI
PCS_PAGE_LPI	nLPI
PCS_PAGE_FACE_NAME	szFaceName
PCS_PAGE_MPL	nMPL
PCS_PAGE_MPP	nMPP

nCPI 1 インチあたりの印刷文字数。

LOWORD は、実際の CPI 値です。

セッションで、フォント CPI が構成されている場合、HIWORD は 1 です。フォント CPI が構成されていない場合、HIWORD は 0 です。

nLPI 1 インチあたりの印刷行数。

LOWORD は、実際の LPI 値です。

セッションで、フォント LPI が構成されている場合、HIWORD は 1 です。フォント LPI が構成されていない場合、HIWORD は 0 です。

szFaceName
プリンター・フォントの書体名。これはヌル終了ストリングでなければなりません。

nFontSize
プリンター・フォントのサイズ。

注: これは、DBCS ホスト・セッション用にのみサポートされています。SBCS ホスト・セッションの場合には無視されます。

nMPL 1 ページに印刷可能な最大行数。

これは MPL (Maximum Print Lines) ともいいます。サポートされる範囲は、1 から 255 です。

nMPP 1 行に印刷可能な最大文字数。

これは MPP (Maximum Print Position) ともいいます。サポートされる範囲は、1 から 255 です。

ULONG * const pErrorInfo
使用されません。これは呼び出し側によって NULL に設定される必要があります。

戻りコード

戻りコード	値	意味
PCS_SUCCESSFUL	0	関数が正常終了しました。
PCS_INVALID_ID	1	無効なセッション ID が指定されました。
PCS_INVALID_SESS_TYPE	2	該当のホスト・セッション・タイプではサポートされていません。
PCS_DIALOG_IN_USE	3	ホスト・セッションの「ページ設定 (Page Setup)」ダイアログまたは「プリンター設定 (Printer Setup)」ダイアログが使用中であったため失敗しました。
PCS_PRINTING	4	ホスト・セッションが印刷中であったため、ページ設定を取得できません。
PCS_PDT_MODE	5	ホスト・セッションが PDT モードになっているため、ページ設定を取得できません。
PCS_SYSTEM_ERROR	9	システム・エラーが発生しました。

例

```
{
   ULONG Rc = 0;
   PAGEINFO *PageInfo;

   PageInfo = (PAGEINFO *) malloc(sizeof(PAGEINFO));
   memset(PageInfo, 0, sizeof(PAGEINFO));

   PageInfo->nFlags = PCS_PAGE_CPI | PCS_PAGE_LPI | PCS_PAGE_FACE_NAME|
                      PCS_PAGE_MPL | PCS_PAGE_MPP;

   Rc = pcsGetPageSettings('A', PageInfo, NULL);

   if (Rc == PCS_SUCCESSFUL) {
      printf("CPI = %d,
              LPI = %d,
              FaceName = %s,
              MPL = %d,
              MPP = %d¥n",
```

```
                LOWORD(PageInfo->nCPI),
                LOWORD(PageInfo->nLPI),
                PageInfo->szFaceName,
                PageInfo->nMPL,
                PageInfo->nMPP);

      if (HIWORD(PageInfo->nCPI))
         printf("FontCPI¥n");
         else
         printf("No FontCPI¥n");

      if (HIWORD(PageInfo->nLPI))
         printf("FontLPI¥n");
         else
         printf("No FontLPI¥n");

   } else
      printf("Failure. Return code = %d¥n", Rc);
   free(PageInfo);
}
```

pcsRestorePageDefaults

3270	*5250*	*VT*
YES	YES	YES

pcsRestorePageDefaults 関数は、nFlags フィールドで定義されている「ページ設定 (Page Setup)」プロパティー・ページのシステム・デフォルト値を復元します。これは、「**ファイル**」→ 「**ページ設定**」ダイアログのプロパティー・ページで「**デフォルト**」をクリックするのと同じです。「**テキスト**」タブ内の設定のみサポートされます。

関数の形式

ULONG WINAPI pcsRestorePageDefaults(*char cShortSessionID, ULONG nFlags*)

パラメーターの形式および説明

char cShortSessionID

表示スペースの短縮セッション ID。

ULONG nFlags

次のフラグは、指定されている「**ページ設定**」ダイアログ・プロパティー・ページの名前を記述します。このフラグをビット単位で論理和演算して、プロパティー・ページ (PCSAPI32.H で定義されています) を復元できます。

PCS_PAGE_TEXT

このフラグは「テキスト」プロパティー・ページを記述します。現在サポートされているのは、このプロパティー・ページのみです。

戻りコード

戻りコード	値	意味
PCS_SUCCESSFUL	0	関数が正常終了しました。
PCS_INVALID_ID	1	無効なセッション ID が指定されました。

戻りコード	値	意味
PCS_INVALID_SESS_TYPE	2	nFlags パラメーターに、ホスト・セッション・タイプとして無効なオプションが 1 つ以上含まれています。設定は復元されませんでした。
PCS_DIALOG_IN_USE	3	ホスト・セッションの「ページ設定 (Page Setup)」ダイアログまたは「プリンター設定 (Printer Setup)」ダイアログが使用中であったため失敗しました。
PCS_PRINTING	4	ホスト・セッションが印刷中であったため、ページ設定を変更できません。
PCS_PDT_MODE	5	ホスト・セッションが PDT モードになっているため、ページ設定を変更できません。
PCS_SYSTEM_ERROR	9	システム・エラーが発生しました。

例

```
{
   ULONG Rc = 0;

   Rc = pcsRestorePageDefaults('A', PCS_PAGE_TEXT);

   if (Rc != PCS_SUCCESSFUL)
      printf("Failure. Return code = %d¥n", Rc);
}
```

pcsSetPageSettings

3270	*5250*	*VT*
YES	YES	YES

pcsSetPageSettings 関数は、ホスト・セッション・ページ設定を設定します。これは、「**ファイル**」→「**ページ設定**」ダイアログで設定を構成するのと同様です。「**テキスト**」タブ内の設定のみサポートされます。

注:

1. CPI、LPI、および FontSize は、ホスト・セッションで構成される FaceName に依存します。この API を使用して、CPI、LPI、FontSize、および FaceName を一緒に設定する場合は、最初に FaceName を設定してから、依存プロパティーを設定してください。
2. この API を使用して、FaceName および依存プロパティーを別個に呼び出して設定する場合は、最初に FaceName を設定してから、CPI、LPI、および FontSize を設定してください。このようにしない場合は、 FaceName を設定するたびに、CPI、LPI、および FontSize を照会し、適切な値になっていることを確認する必要があります。
3. CPI、LPI、または FontSize を FaceName より前に設定すると、ホスト・セッション内で CPI、LPI、または FontSize に対して異なる値が構成される場合があります。これは、CPI、LPI、または FontSize の現行の値が、 FaceName の新しい設定値に対して有効でない場合に起こります。

関数の形式

ULONG WINAPI pcsSetPageSettings(*char cShortSessionID, const PAGEINFO * const pPageInfo, ULONG * const pErrorInfo*)

パラメーターの形式および説明

char cShortSessionID

表示スペースの短縮セッション ID。

const PAGEINFO * const pPageInfo

ページ設定値が通知される PAGEINFO 構造体へのポインター。

nFlags 構造体の有効なメンバーを示す、ビット・フラグの組み合わせ。これらのフラグを単独で使用するか、論理和演算で処理することにより、プロパティー・ページ (PCSAPI32.H に定義されています) を復元することができます。各フラグと、対応する構造体内の有効なメンバーは以下のとおりです。

フラグ	構造体内の有効なメンバー
PCS_PAGE_CPI	nCPI
PCS_PAGE_LPI	nLPI
PCS_PAGE_FACE_NAME	szFaceName
PCS_PAGE_MPL	nMPL
PCS_PAGE_MPP	nMPP

nCPI 1 インチあたりの印刷文字数。

フォント CPI を選択するには、nCPI の HIWORD を 1 に設定します。 nCPI の LOWORD は無視されます。

特定の CPI 値を選択するには、以下を行ってください。

1. nCPI の HIWORD を 0 に設定します。
2. nCPI の LOWORD を実際の CPI 値に設定します。

nLPI 1 インチあたりの印刷行数。

フォント LPI を選択するには、nLPI の HIWORD を 1 に設定します。nLPI の LOWORD は無視されます。

特定の LPI 値を選択するには、以下を行ってください。

1. nLPI の HIWORD を 0 に設定します。
2. nLPI の LOWORD を実際の LPI 値に設定します。

szFaceName

プリンター・フォントの書体名。これはヌル終了ストリングでなければなりません。

nFontSize

プリンター・フォントのサイズ。

注: これは、DBCS ホスト・セッション用にのみサポートされています。SBCS ホスト・セッションの場合には無視されます。

nMPL 1 ページに印刷可能な最大行数。

これは MPL (Maximum Print Lines) ともいいます。サポートされる範囲は、1 から 255 です。

nMPP 1 行に印刷可能な最大文字数。

これは MPP (Maximum Print Position) ともいいます。サポートされる範囲は、1 から 255 です。

ULONG * const pErrorInfo

API が戻りコード PCS_FAILURE で失敗した場合に、詳細なエラー情報が記載されます。詳細なエラー情報が必要ない場合、このフラグは呼び出し側によって NULL に設定される必要があります。

これはビット・フラグの組み合わせで、正常に設定できない PAGEINFO 構造体のメンバーを記述します。これらのフラグは、PCSAPI32.H で以下のように定義されています。

フラグ	構造体内の有効なメンバー
PCS_PAGE_CPI	nCPI のみが無効です。
PCS_PAGE_LPI	nLPI のみが無効です。
PCS_PAGE_FACE_NAME	szFaceName のみが無効です。
PCS_PAGE_MPL	nMPL のみが無効です。
PCS_PAGE_MPP	nMPP のみが無効です。

戻りコード

戻りコード	値	意味
PCS_SUCCESSFUL	0	関数が正常終了しました。
PCS_INVALID_ID	1	無効なセッション ID が指定されました。
PCS_INVALID_SESS_TYPE	2	該当のホスト・セッション・タイプではサポートされていません。
PCS_DIALOG_IN_USE	3	ホスト・セッションの「ページ設定 (Page Setup)」ダイアログまたは「プリンター設定 (Printer Setup)」ダイアログが使用中であったため失敗しました。
PCS_PRINTING	4	ホスト・セッションが印刷中であったため、ページ設定を変更できません。
PCS_PDT_MODE	5	ホスト・セッションが PDT モードになっているため、ページ設定を変更できません。
PCS_FAILURE	6	ホスト・セッションのページ設定が完全に適用されていません。PAGEINFO 構造体の一部またはすべてのフィールドに、無効なデータが指定されていることが原因として考えられます。 pErrorInfo を調べて、適用されていない設定についての詳細を確認してください。
PCS_SYSTEM_ERROR	9	システム・エラーが発生しました。

例

```
{
   ULONG Rc = 0, Error = 0;
   PAGEINFO *PageInfo;

   PageInfo = (PAGEINFO *) malloc(sizeof(PAGEINFO));
   memset(PageInfo, 0, sizeof(PAGEINFO));

   PageInfo->nFlags = PCS_PAGE_CPI | PCS_PAGE_LPI |
                      PCS_PAGE_FACE_NAME| PCS_PAGE_MPL |
                      PCS_PAGE_MPP;
   PageInfo->nCPI = MAKELONG(10, 0);
   PageInfo->nLPI = MAKELONG(8, 0);
   PageInfo->nMPL = 40;
   PageInfo->nMPP = 60;
   strcpy(PageInfo->szFaceName, "CourierPS");

   Rc = pcsSetPageSettings('A', PageInfo, &Error);

   if (Rc != PCS_SUCCESSFUL) {
      printf("Failure. Return code = %d¥n", Rc);
      printf("Following members could not be set : ");

      if (Rc == PCS_FAILURE) {
         if (Error & PCS_PAGE_CPI) printf(" nCPI");
         if (Error & PCS_PAGE_LPI) printf(" nLPI");
         if (Error & PCS_PAGE_FACE_NAME) printf(" szFaceName");
         if (Error & PCS_PAGE_MPL) printf(" nMPL");
         if (Error & PCS_PAGE_MPP) printf(" nMPP");
         printf("¥n");
      }
   }
   free(PageInfo);
}
```

プリンター設定関数

このセクションで説明する各種 PCSAPI 関数を使用すると、パーソナル・コミュニケーションズのエミュレーター・セッションの「**プリンター設定 (Printer Setup)**」の設定値を制御および検索できます。

制約事項

以下の制約事項を順守していない場合、API に障害が起こります。戻りコードに、障害の理由が示されます。

- API の呼び出し時に、ホスト・セッションが印刷中であってはならない。
- 「**ファイル**」→「**プリンター設定 (Printer Setup)**」ダイアログが使用中であってはならない。

注: これらの関数は、現在、DBCS セッションおよび双方向セッションについてはサポートされていません。

pcsGetPrinterSettings

3270	*5250*	*VT*
YES	YES	YES

pcsGetPrinterSettings 関数は、ホスト・セッションのプリンター設定値を検索します (「**ファイル**」→「**プリンター設定 (Printer Setup)**」ダイアログの設定と同様)。

関数の形式

ULONG WINAPI pcsGetPrinterSettings(*char cShortSessionID, PRINTINFO * const pPrintInfo, ULONG * const pErrorInfo***)**

パラメーターの形式および説明

char cShortSessionID
表示スペースの短縮セッション ID。

PRINTINFO * const pPrintInfo
プリンター設定値が指定される PRINTINFO 構造体へのポインター。

nFlags 0 を設定する必要があります。この値は無視されます。

nBufSize
以下のフィールドに割り振られるバッファーのサイズ。

- lpPDTFile
- lpPrtToDskAppFile
- lpPrtToDskSepFile
- lpPrinterName

単一の API 呼び出しで、複数のメンバーが検索される場合、呼び出し側は、すべてのバッファーに同じサイズを割り振り、このメンバーにそのサイズを渡す必要があります。

このメンバーが 0 に設定されている場合、フィールドは無視されます。フィールドのバッファーに必要な最大サイズは、nSizeNeeded に戻されます。

nSizeNeeded
このメンバーの値は、以下のフィールドに関連する条件によって決まります。

- lpPDTFile
- lpPrtToDskAppFile
- lpPrtToDskSepFile
- lpPrinterName

条件は、以下のとおりです。

- 呼び出し側によって割り振られたバッファーのサイズが、上記にリストしたフィールドを戻すのに十分でない場合、この値は、必要なバイト数になる。
- 上記にリストしたフィールドの複数が呼び出し側によって取得される場合、この値は、必要バッファーの最大サイズになる。
- nBufSize が呼び出し側によって 0 に設定されている場合、このメンバーには、上にリストしたフィールドのバッファーに必要な最大サイズが含まれる。

bPromptDialog
可能な値は以下のとおりです。

- TRUE の場合、印刷の前に「プリンター設定 (Printer Setup)」ダイアログが表示されます。
- FALSE の場合、印刷の前に「プリンター設定 (Printer Setup)」ダイアログは表示されません。

bPDTMode

可能な値は以下のとおりです。

- TRUE の場合、ホスト・セッションは PDT モードになります。
- FALSE の場合、ホスト・セッションは非 PDT モード (GDI モード) になります。

lpPDTFile

呼び出し側がこのメンバーを取得しない場合は、NULL に設定されている必要があります。ヌル・ポインターでない場合は、PDT ファイルが戻されます。このメンバ ーは、呼び出し側によって割り振られたサイズ nBufSize のバッファーを指している必要があります。

API が戻るとき、このメンバーには以下のいずれかが含まれています。

- セッション PDT ファイルの完全修飾パス名。
- セッションで PDT ファイルが構成されていない場合は、空ストリング (″″)。
- バッファー・サイズが十分でない場合は、切り捨てられたファイル名。メンバー nSizeNeeded に、必要なバッファー・サイズが含まれています。

nPrtMode

これは、接続の PrintMode を示す列挙値です。列挙データ型 PRINTMODE は、PCSAPI32.H に定義されています。 nPrtMode 設定値は、以下のいずれかでなければなりません。

- **PrtToDskAppend** (**ディスクへの印刷 - コピー追加**モード)

  これは、ホスト・セッションの「**プリンター設定 (Printer Setup)**」→「**プリンター (Printer)**」→「**ディスクへの印刷 (Print to Disk)**」ダイアログで「**コピー追加 (Append)**」オプションを選択するのと同じです。

- **PrtToDskSeparate** (**ディスクへの印刷 - 別個**モード)

  これは、ホスト・セッションの「**プリンター設定 (Printer Setup)**」→「**プリンター (Printer)**」→「**ディスクへの印刷 (Print to Disk)**」ダイアログで「**別個 (Separate)**」オプションを選択するのと同じです。

- **WinDefaultPrinter** (**Windows デフォルト・プリンター**・モード)

  これは、ホスト・セッションの「**プリンター設定 (Printer Setup)**」ダイアログで、「**Windows デフォルト・プリンターを使用 (Use Windows Default Printer)**」オプションを選択するのと同じです。

- **SpecificPrinter** (**特定プリンター**・モード)

これは、ホスト・セッションの「**プリンター設定 (Printer Setup)**」ダイアログで、「**Windows デフォルト・プリンターを使用 (Use Windows Default Printer)**」オプションのチェックを外した状態でプリンターを選択するのと同じです。

lpPrtToDskAppFile

呼び出し側がこのメンバーを取得しない場合は、NULL に設定されている必要があります。ヌル・ポインターでない場合は、**Print to Disk-Append** ファイルが戻されます。このメンバ ーは、呼び出し側によって割り振られたサイズ nBufSize のバッファーを指している必要があります。

API が戻るとき、このメンバーには以下のいずれかが含まれています。

- セッションの **Print to Disk-Append** ファイルの完全修飾パス名。
- セッションに **Print to Disk-Append** ファイルが構成されていない場合は、空ストリング (″″)。
- バッファー・サイズが十分でない場合は、切り捨てられたファイル名。 nSizeNeeded メンバーに、必要なバッファー・サイズが含まれています。

lpPrtToDskSepFile

呼び出し側がこのメンバーを取得しない場合は、NULL に設定されている必要があります。ヌル・ポインターでない場合は、 **Print to Disk-Separate** ファイルが戻されます。このメンバ ーは、呼び出し側によって割り振られたサイズ nBufSize のバッファーを指している必要があります。

API が戻るとき、このメンバーには以下のいずれかが含まれています。

- セッションの **Print to Disk-Separate** ファイルの完全修飾パス名。
- セッションで、**Print to Disk-Separate** ファイルが構成されていない場合は、空ストリング (″″)。
- バッファー・サイズが十分でない場合は、切り捨てられたファイル名。 nSizeNeeded メンバーに、必要なバッファー・サイズが含まれています。

lpPrinterName

呼び出し側がこのメンバーを取得しない場合は、NULL に設定されている必要があります。ヌル・ポインターでない場合は、プリンター名が戻されます。このメンバ ーは、呼び出し側によって割り振られたサイズ nBufSize のバッファーを指している必要があります。

API が戻るとき、このメンバーには以下のいずれかが含まれています。

- ホスト・セッション nPrtMode が SpecificPrinter である場合は、セッションで構成されている特定プリンターの名前。

- ホスト・セッション nPrtMode が WinDefaultPrinter である場合は、セッションで構成されている Windows デフォルト・プリンターの名前。
- ホスト・セッション nPrtMode が PrtToDskAppend または PrtToDskSeparate である場合は、空ストリング (″″)。
- バッファー・サイズが十分でない場合は、切り捨てられたプリンター名。nSizeNeeded メンバーに、必要なバッファー・サイズが含まれています。

プリンター名の形式は次のとおりです。

<Printer name> on *<Port Name>*

次に例を示します。

- IBM InfoPrint 40 PS on Network Port
- HP LaserJet 4050 Series PCL 6 on LPT1

ULONG * const pErrorInfo

API が戻りコード PCS_FAILURE で失敗した場合に、詳細なエラー情報が記載されます。エラーの詳細が必要ない場合、pErrorInfo は、呼び出し側によって NULL に設定される必要があります。

以下のセクションで、PCSAPI32.H で定義されているフラグを説明します。

PRINTINFO 構造体の pErrorInfo メンバー用のフラグ

PCS_PRINT_PRINTMODE_ERROR

ホスト・セッションで PrintMode が構成されていません。

PCS_PRINT_PDTFILE_SIZEERR

lpPDTFile のバッファー・サイズが十分でないので、ファイル名が切り捨てられます。nSizeNeeded メンバーに、PDT ファイルを戻すのに必要となる実際のバッファー・サイズが含まれています。

PCS_PRINT_DSKAPPFILE_SIZEERR

lpPrtToDskAppFile のバッファー・サイズが十分でないので、ファイル名が切り捨てられます。nSizeNeeded メンバーに、**Print to Disk-Append** ファイルを戻すのに必要となる実際のバッファー・サイズが含まれています。

PCS_PRINT_DSKSEPFILE_SIZEERR

lpPrtToDskSepFile のバッファー・サイズが十分でないので、ファイル名が切り捨てられます。nSizeNeeded メンバーに、**Print to Disk-Separate** ファイルを戻すのに必要となる実際のバッファー・サイズが含まれています。

PCS_PRINT_PRINTERNAME_SIZEERR

lpPrinterName のバッファー・サイズが十分でないので、プリンター名が切り捨てられます。 nSizeNeeded メンバーに、プリンター名を戻すのに必要となる実際のバッファー・サイズが含まれています。

戻りコード

戻りコード	値	意味
PCS_SUCCESSFUL	0	関数が正常終了しました。

戻りコード	値	意味
PCS_INVALID_ID	1	無効なセッション ID が指定されました。
PCS_DIALOG_IN_USE	3	ホスト・セッションの「ページ設定 (Page Setup)」ダイアログまたは「プリンター設定 (Printer Setup)」ダイアログが使用中であったため失敗しました。
PCS_PRINTING	4	ホスト・セッションが印刷中であったため、プリンター設定を変更できませんでした。後で、アプリケーションを再試行する必要があります。
PCS_FAILURE	6	一部のプリンター設定を正常に検索できませんでした。 pErrorInfo に、検索できなかった設定に関する詳細なエラー情報が含まれています。
PCS_SYSTEM_ERROR	9	システム・エラーが発生しました。

例

```
{
   ULONG Rc = 0, Error=0, Size;
   PRINTINFO *PrintInfo;

   PrintInfo = (PRINTINFO *) malloc(sizeof(PRINTINFO));
   memset(PrintInfo, 0, sizeof(PRINTINFO));

   PrintInfo->nBufSize = 0;

   Rc = pcsGetPrinterSettings('A', PrintInfo, &Error);
   if (Rc != PCS_SUCCESSFUL)
      printf("Failure. Return code = %d\n", Rc);
   else {
      Size = PrintInfo->nSizeNeeded;
      PrintInfo->nBufSize = Size;
      PrintInfo->lpPDTFile = (char *)malloc(sizeof(char) * Size);
      PrintInfo->lpPrtToDskAppFile = (char *)malloc(sizeof(char) * Size);
      PrintInfo->lpPrtToDskSepFile = (char *)malloc(sizeof(char) * Size);
      PrintInfo->lpPrinterName = (char *)malloc(sizeof(char) * Size);
      Rc = pcsGetPrinterSettings('A', PrintInfo, &Error);

      if (Rc != PCS_SUCCESSFUL)
         printf("Failure. Return code = %d, Extended Error = 0x%08x¥n", Rc, Error);
      else {
         if (PrintInfo->bPromptDialog)
            printf("PromptDialog¥n");
         else
            printf("No PromptDialog¥n");
         if (PrintInfo->bPDTMode)
            printf("PDT Mode¥n");
         else
            printf("Not PDT Mode¥n");

         switch(PrintInfo->nPrtMode) {

         case PrtToDskAppend:
            printf("Print to Disk-Append Mode¥n");
            break;
         case PrtToDskSeparate:
            printf("Print to Disk-Separate Mode¥n");
            break;
         case SpecificPrinter:
```

```
            printf("Specific Printer Mode¥n");
            break;
         case WinDefaultPrinter:
            printf("Windows Default Printer Mode¥n");
            break;
         }
         if (PrintInfo->lpPDTFile[0] == '¥0')
            printf("No PDT File configured¥n");
         else
            printf("PDT File = %s¥n", PrintInfo->lpPDTFile);
         if (PrintInfo->lpPrtToDskAppFile[0] == '¥0')
            printf("No Disk Append File configured¥n");
         else
            printf("DiskAppend File=%s¥n", PrintInfo->lpPrtToDskAppFile);
         if (PrintInfo->lpPrtToDskSepFile[0] == '¥0')
            printf("No Disk Separate File configured¥n");
         else
            printf("DiskSeparate File=%s¥n", PrintInfo->lpPrtToDskSepFile);
         if ((PrintInfo->nPrtMode == SpecificPrinter) ||
             (PrintInfo->nPrtMode == WinDefaultPrinter))
            printf("Printer = %s¥n", PrintInfo->lpPrinterName);
      }
      free(PrintInfo->lpPDTFile);
      free(PrintInfo->lpPrtToDskAppFile);
      free(PrintInfo->lpPrtToDskSepFile);
      free(PrintInfo->lpPrinterName);
   }
   free(PrintInfo);
}
```

pcsSetPrinterSettings

3270	*5250*	*VT*
YES	YES	YES

pcsSetPrinterSettings 関数は、ホスト・セッションのプリンター設定を制御します (「**ファイル**」→「**プリンター設定 (Printer Setup)**」ダイアログの設定と同様)。

関数の形式

ULONG WINAPI pcsSetPrinterSettings(*char cShortSessionID, const PRINTINFO * const pPrintInfo, ULONG * const pErrorInfo*)

パラメーターの形式および説明

char cShortSessionID

表示スペースの短縮セッション ID。

const PRINTINFO * const pPrintInfo

プリンター設定が通知される PRINTINFO 構造体へのポインター。

nFlags 構造体の有効なメンバーを示す、ビット・フラグの組み合わせ。これらのフラグを単独で使用するか、論理和演算で処理することにより、プロパティー・ページ (PCSAPI32.H に定義されています) を復元することができます。各フラグと、対応する構造体内の有効なメンバーは以下のとおりです。

フラグ	構造体内の有効なメンバー
PCS_PRINT_PDT	bPDTMode、lpPDTFile

PCS_PRINT_PRINTMODE nPrtMode、lpPrtToDskAppFile、lpPrtToDskSepFile、lpPrinterName

PCS_PRINT_PROMPT_DIALOG
bPromptDialog

nBufSize
0 を設定する必要があります。この値は無視されます。

nSizeNeeded
0 を設定する必要があります。この値は無視されます。

bPromptDialog
可能な値は以下のとおりです。

- TRUE の場合、印刷の前に「プリンター設定 (Printer Setup)」ダイアログが表示されます。
- FALSE の場合、印刷の前に「プリンター設定 (Printer Setup)」ダイアログは表示されません。

bPDTMode
可能な値は以下のとおりです。

- TRUE の場合、接続は PDT モードに設定されます。
- FALSE の場合、接続は非 PDT モード (GDI モード) に設定されます。

lpPDTFile
bPDTMode が TRUE に設定されている場合にのみ使用されます。bPDTMode が FALSE に設定されている場合は、無視されます。

これは、PDT ファイル名を含むヌル終了ストリングで、以下のいずれかである必要があります。

- NULL

  現在、接続で構成されている PDT ファイルが使用されます。接続で既に構成されている PDT ファイルがない場合、API は失敗して例外を出します。
- ファイル名 (パスなし)

  パーソナル・コミュニケーションズ・インストール・パスの PDFPDT サブフォルダー内の lpPDTFile が使用されます。
- ファイルの完全修飾パス名

  lpPDTFile が存在しない場合、API は失敗します。

nPrtMode
これは、接続の PrintMode を示す列挙値です。列挙データ型 PRINTMODE は、PCSAPI32.H に定義されています。 nPrtMode 設定値は、以下のいずれかでなければなりません。

- **PrtToDskAppend (ディスクへの印刷 - コピー追加**モード)

これは、ホスト・セッションの「**プリンター設定 (Printer Setup)**」→「**プリンター (Printer)**」→「**ディスクへの印刷 (Print to Disk)**」ダイアログで「**コピー追加 (Append)**」オプションを選択するのと同じです。

- **PrtToDskSeparate** (**ディスクへの印刷 - 別個**モード)

  これは、ホスト・セッションの「**プリンター設定 (Printer Setup)**」→「**プリンター (Printer)**」→「**ディスクへの印刷 (Print to Disk)**」ダイアログで「**別個 (Separate)**」オプションを選択するのと同じです。

- **WinDefaultPrinter** (**Windows デフォルト・プリンター**・モード)

  これは、ホスト・セッションの「**プリンター設定 (Printer Setup)**」ダイアログで、「**Windows デフォルト・プリンターを使用 (Use Windows Default Printer)**」オプションを選択するのと同じです。

- **SpecificPrinter** (**特定プリンター**・モード)

  これは、ホスト・セッションの「**プリンター設定 (Printer Setup)**」ダイアログで「**Windows デフォルト・プリンターを使用 (Use Windows Default Printer)**」オプションのチェックを外した状態でプリンターを選択するのと同じです。

lpPrtToDskAppFile

これは、nPrtMode が PrtToDskAppend に設定されている場合にのみ使用されます。

これは、**Print to Disk-Append** ファイル名を含むヌル終了ストリングで、以下のいずれかである必要があります。

- NULL

  現在、接続で PrtToDskAppend モード用に構成されているファイルが使用されます。接続で既に構成されている PDT ファイルがない場合、 API は失敗します。

- ファイル名 (パスなし)

  ユーザー・クラスのアプリケーション・データ・ディレクトリー・パスを使用してファイルの場所を検索します。ファイルが存在する場合は、それが使用されます。存在しない場合、印刷の完了時に作成されます。

- ファイルの完全修飾パス名

  パスにディレクトリーが存在している必要があります。存在していない場合、API は失敗します。パスにファイルが存在している必要はありません。

lpPrtToDskSepFile

可能な値は以下のとおりです。

- セッションの **Print to Disk-Separate** ファイルの完全修飾パス名。

- セッションで、**Print to Disk-Separate** ファイルが構成されていない場合は、空ストリング (″″)。
- バッファー・サイズが十分でない場合は、切り捨てられたファイル名。 nSizeNeeded メンバーに、必要なバッファー・サイズが含まれています。

lpPrinterName

これは、nPrtMode が SpecificPrinter に設定されている場合にのみ使用されます。そうでない場合は無視されます。これは、プリンター名を含むヌル終了ストリングです。プリンターが存在しない場合、このメンバーは失敗します。

プリンター名の形式は次のとおりです。

```
<Printer name> on <Port Name>
```

次に例を示します。

- `IBM InfoPrint 40 PS on Network Port`
- `HP LaserJet 4050 Series PCL 6 on LPT1`

ULONG * const pErrorInfo

API が戻りコード PCS_FAILURE で失敗した場合に、詳細なエラー情報が記載されます。エラーの詳細が必要ない場合、pErrorInfo は、呼び出し側によって NULL に設定される必要があります。

以下のセクションで、PCSAPI32.H で定義されているフラグを説明します。

PRINTINFO 構造体の pErrorInfo メンバー用のフラグ

PCS_PRINT_PDTMODE_ERROR

以下のいずれかの理由により発生します。

- bPDTMode が TRUE、lpPDTFile が NULL に設定されている場合に、ホスト・セッション用に構成済みの PDT ファイルがない。
- nPrtMode が PrtToDskAppend または PrtToDskSeparate に設定され、PCS_PRINT_PDT が nFlags 内で設定されていない場合に、ホスト・セッションが PDT モードになっていない。
- nPrtMode が PrtToDskAppend または PrtToDskSeparate に設定され、bPDTMode が FALSE に設定されている。

PCS_PRINT_PDTFILE_ERROR

lpPDTFile で指定されているファイルまたはパスが見つかりませんでした。

PCS_PRINT_PRTTODSK_FILE_ERROR

以下のいずれかの理由により発生します。

- lpPrtToDskAppFile フィールドまたは lpPrtToDskSepFile フィールドで指定されているフォルダーが存在していないか、そのフォルダーに書き込みアクセス権限がない。
- lpPrtToDskSepFile フィールドに拡張子が指定されている。

PCS_PRINT_PRINTMODE_ERROR

nPrtMode が正常に設定できません。これは、以下のいずれかの理由により起こります。

- nPrtMode の値が、PRINTMODE 列挙データ型の列挙型定数の 1 つでない。
- nPrtMode が PrtToDskAppend に、lpPrtToDskAppFile が NULL に設定されている場合に、ホスト・セッションに構成済みの **Print to Disk-Append** ファイルがない。
- nPrtMode が PrtToDskSeparate に、lpPrtToDskSepFile が NULL に設定されている場合に、ホスト・セッションに構成済みの **Print to Disk-Separate** ファイルがない。
- nPrtMode が SpecificPrinter に設定されている場合に、lpPrinterName フィールドで指定されているプリンターが見つからない。
- nPrtMode が WinDefaultPrinter に設定されている場合に、デフォルトの Windows プリンターがシステムに構成されていない。
- bPDTMode が FALSE に設定されており、PCS_PRINT_PRINTMODE が nFlags に設定されていないが、ホスト・セッション PrintMode が PrtToDskAppend または PrtToDskSeparate である。

戻りコード

戻りコード	値	意味
PCS_SUCCESSFUL	0	関数が正常終了しました。
PCS_INVALID_ID	1	無効なセッション ID が指定されました。
PCS_DIALOG_IN_USE	3	ホスト・セッションの「ページ設定 (Page Setup)」ダイアログまたは「プリンター設定 (Printer Setup)」ダイアログが使用中であったため失敗しました。
PCS_PRINTING	4	ホスト・セッションが印刷中であったため、プリンター設定を変更できませんでした。後で、アプリケーションを再試行する必要があります。
PCS_FAILURE	6	ホスト・セッションのプリンター設定が適用されていません。PRINTINFO 構造体の一部またはすべてのフィールドに、無効なデータが指定されていることが原因として考えられます。pErrorInfo にエラーの詳細が記述されています。
PCS_SYSTEM_ERROR	9	システム・エラーが発生しました。

例

```
{
   ULONG Rc = 0, Error=0;
   PRINTINFO *PrintInfo;
   char PDTFile[] = "epson.pdt";
   char SepFile[] = "DiskSep";

   PrintInfo = (PRINTINFO *) malloc(sizeof(PRINTINFO));
   memset(PrintInfo, 0, sizeof(PRINTINFO));

   PrintInfo->nFlags = PCS_PRINT_PDT | PCS_PRINT_PRINTMODE |
         PCS_PRINT_PROMPT_DIALOG;
   PrintInfo->nBufSize = 0;
```

```
   PrintInfo->nSizeNeeded = 0;
   PrintInfo->bPDTMode = TRUE;
   PrintInfo->lpPDTFile =
         (char *)malloc(sizeof(char) * (strlen(PDTFile)+1));
   strcpy(PrintInfo->lpPDTFile, PDTFile);
   PrintInfo->nPrtMode = PrtToDskSeparate;
   PrintInfo->lpPrtToDskSepFile =
         (char *)malloc(sizeof(char) * (strlen(SepFile)+1));
   strcpy(PrintInfo->lpPrtToDskSepFile, SepFile);
   PrintInfo->bPromptDialog = TRUE;
   Rc = pcsSetPrinterSettings('A', PrintInfo, &Error);
   if (Rc != PCS_SUCCESSFUL)
      printf("Failure. Return code = %d, Extended Error = 0x%08x\n", Rc, Error);
   free(PrintInfo->lpPDTFile);
   free(PrintInfo->lpPrtToDskSepFile);
   free(PrintInfo);
}
```

第 6 章 32 ビット環境での DDE 関数

この章では、Windows 32 ビット環境で使用される DDE 関数について説明します。

パーソナル・コミュニケーションズは 32 ビットの動的データ交換 (DDE) インターフェースを備えており、これによってアプリケーションでデータを交換することができます。 2 つの Windows アプリケーション間のデータの交換は、クライアントとサーバー間の会話として考えることができます。クライアント は、DDE 会話を開始します。これに対しサーバー は、クライアントへの対応を行います。パーソナル・コミュニケーションズは、パーソナル・コミュニケーションズが管理しているオープン・セッションのための DDE サーバーです。 DDE の詳細については「*Microsoft Windows Software Development Kit Guide to Programming*」を参照してください。

注: Visual Basic で DDE 関数を使用する場合は、323 ページの『第 7 章 DDE クライアント・アプリケーションを使った DDE 関数の使用方法』を参照してください。

パーソナル・コミュニケーションズは 16 ビットの DDE アプリケーションもサポートしています。395 ページの『付録 E. 16 ビット環境での DDE 関数』を参照してください。

パーソナル・コミュニケーションズの DDE データ・アイテム

Microsoft Windows DDE は、データ・アイテムを識別するために 3 つのレベルの命名形式を使用します。つまり、アプリケーション、トピック、アイテムです。 表 17 はこれらのレベルを説明したものです。

表 17. データ・アイテム用の命名形式

レベル	説明	例
アプリケーション	Windows タスク、またはアプリケーションの特定のタスク。本書ではアプリケーションとは、パーソナル・コミュニケーションズのことです。	IBM327032
トピック	アプリケーションの特定の部分。	SessionA
アイテム	データ交換で受け渡しできるデータ・オブジェクト。アイテムとは、アプリケーションに定義されたデータ・アイテムのことで、Windows クリップボード形式の 1 つか、またはアプリケーションが独自に定義したクリップボード形式に従います。 Windows のクリップボード形式の詳細については、「*Microsoft Windows Software Development Kit Guide to Programming*」を参照してください。	PS (表示スペース)

パーソナル・コミュニケーションズは IBM327032 および IBM525032 アプリケーションを Win32 DDE サーバーとしてサポートします。

次のトピックを使用できます。

- System
- SessionA、SessionB、...、SessionZ
- LUA_xxxx、LUB_xxxx、...、LUZ_xxxx

DDE では、アトム を使用してアプリケーション名、トピック名、およびデータ・アイテムを識別します。アトムは、固有の整数値に縮小された文字ストリングを表します。この文字ストリングは、アトム・テーブルに追加され、このテーブルを参照すれば、アトムに関連付けられたストリングの値を見つけることができます。アトムは GlobalAddAtom 関数呼び出しによって作成します。アトムの作成方法と使用方法の詳細については、「*Microsoft Windows Software Development Kit Guide to Programming*」を参照してください。

システム・トピック・データ・アイテムの使用

DDE インターフェースを提供するアプリケーションは、SYSTEM という名前の特殊なトピックも提供することになっています。このトピックが、アプリケーションに一般的な関連をもつ情報アイテムに関するコンテキストを提供します。パーソナル・コミュニケーションズの SYSTEM トピックには、次の関連データ・アイテムが含まれます。

アイテム	関数
Formats	パーソナル・コミュニケーションズでレンダリング可能なクリップボード形式 (番号) のリストを戻します。
Status	各パーソナル・コミュニケーションズのセッションの状況に関する情報を戻します。
SysCon	パーソナル・コミュニケーションズのサポートのレベルと、その他のシステム関連の値を戻します。
SysItems	パーソナル・コミュニケーションズのシステム・トピックに接続したときに使用できるデータ・アイテムのリストを戻します。
Topics	使用可能なパーソナル・コミュニケーションズのトピックのリストを戻します。

セッション・トピック・データ・アイテムの使用

各セッション・トピックに対して、次のデータ・アイテムがサポートされています。

アイテム	関数
CLOSE	ウィンドウ・クローズ要求を取り出します。
CONV	ASCII から EBCDIC、および EBCDIC から ASCII へのコード変換を要求します。
EPS	追加データをもつセッションの表示スペースを取り出します。
EPSCOND	表示スペース・サービス条件を取り出します。
FIELD	セッションの表示スペース内のフィールドを取り出します。
KEYS	キー・ストロークを取り出します。
MOUSE	マウス入力を取り出します。
OIA	オペレーター情報域の状況表示行を取り出します。

アイテム	関数
PS	セッションの表示スペースを取り出します。
PSCOND	セッションのアドバイス条件を取り出します。
SSTAT	セッションの状況を取り出します。
STRING	ASCII ストリング・データを取り出します。
TRIMRECT	現行のトリミング長方形内にあるセッションの表示スペースを取り出します。

LU トピック・データ・アイテムの使用 (3270 のみ)

各 LU トピックに対して、次のデータ・アイテムがサポートされています。

アイテム	関数
SF	Destination/Origin 構造化フィールド・データを取り出します。
SFCOND	Query Reply データを取り出します。

DDE 関数

表 18 は、パーソナル・コミュニケーションズ で使用できる DDE 関数を示したものです。

表 18. パーソナル・コミュニケーションズで使用可能な DDE 関数

関数	3270	5250	VT
244 ページの『Code Conversion』	YES	YES	YES
246 ページの『Find Field』	YES	YES	YES
249 ページの『Get Keystrokes』	YES	YES	YES
250 ページの『Get Mouse Input』	YES	YES	YES
253 ページの『Get Number of Close Requests』	YES	YES	YES
254 ページの『Get Operator Information Area』	YES	YES	YES
255 ページの『Get Partial Presentation Space』	YES	YES	YES
257 ページの『Get Presentation Space』	YES	YES	YES
259 ページの『Get Session Status』	YES	YES	YES
261 ページの『Get System Configuration』	YES	YES	YES
262 ページの『Get System Formats』	YES	YES	YES
263 ページの『Get System Status』	YES	YES	YES
264 ページの『Get System SysItems』	YES	YES	YES
265 ページの『Get System Topics』	YES	YES	YES
266 ページの『Get Trim Rectangle』	YES	YES	YES
267 ページの『Initiate Session Conversation』	YES	YES	YES
268 ページの『Initiate Structured Field Conversation』	YES	NO	NO
269 ページの『Initiate System Conversation』	YES	YES	YES
269 ページの『Put Data to Presentation Space』	YES	YES	YES
270 ページの『Search for String』	YES	YES	YES
271 ページの『Send Keystrokes』	YES	YES	YES
273 ページの『Session Execute Macro』	YES	YES	YES
281 ページの『Set Cursor Position』	YES	YES	YES
282 ページの『Set Mouse Intercept Condition』	YES	YES	YES
285 ページの『Set Presentation Space Service Condition』	YES	YES	YES
286 ページの『Set Session Advise Condition』	YES	YES	YES
287 ページの『Set Structured Field Service Condition』	YES	NO	NO
289 ページの『Start Close Intercept』	YES	YES	YES

表 18. パーソナル・コミュニケーションズで使用可能な DDE 関数 (続き)

関数	3270	5250	VT
290 ページの『Start Keystroke Intercept』	YES	YES	YES
291 ページの『Start Mouse Input Intercept』	YES	YES	YES
294 ページの『Start Read SF』	YES	NO	NO
296 ページの『Start Session Advise』	YES	YES	YES
298 ページの『Stop Close Intercept』	YES	YES	YES
298 ページの『Stop Keystroke Intercept』	YES	YES	YES
299 ページの『Stop Mouse Input Intercept』	YES	YES	YES
300 ページの『Stop Read SF』	YES	NO	NO
300 ページの『Stop Session Advise』	YES	YES	YES
301 ページの『Terminate Session Conversation』	YES	YES	YES
302 ページの『Terminate Structured Field Conversation』	YES	NO	NO
302 ページの『Terminate System Conversation』	YES	YES	YES
303 ページの『Write SF』	YES	NO	NO

DDE 関数の要約については、 317 ページの『Windows 32 ビット環境での DDE 関数の要約』を参照してください。

パラメーターの命名規則

ほとんどの DDE パラメーター名には、ローカル変数が付いています。これらのローカル変数には、パラメーターの一般的なタイプを示す接頭部が付いており、その後にパラメーターの内容を説明する 1 つ以上の単語が続きます。本書で提示する接頭部は次のとおりです。

a アトム

c 文字 (1 バイト値)

f 16 ビット整数にパックされたビット・フラグ

h 16 ビット・ハンドル

p Short 型 (16 ビット) ポインター

lp Long 型 (32 ビット) ポインター

w Short 型 (16 ビット) 符号なし整数

u 符号なし整数

sz Null 文字で終了する文字ストリング

Code Conversion

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

Code Conversion 関数を使用すると、クライアント・アプリケーションは、ASCII から EBCDIC へまたは EBCDIC から ASCII への変換を行うことができます。この関数は、32 ビットのアプリケーションでのみ使用可能です。

メッセージを次のように送信します。

```
PostMessage (hServerWnd,
            WM_DDE_POKE,
            hClientWnd,
            PackDDEIParam (WM_DDE_POKE, hData, aCONV));
```

ここで、

hDATA

```
typedef struct tagWCDDE_CONV
{
    BYTE         ddepoke[(sizeof(DDEPOKE)-1)];
    char         szSourceName[256];   // name of memory-mapped file
    char         szTargetName[256];   // name of memory-mapped file
    BYTE         ConvType;            // Conversion method
    WORD         uSourceLength;       // Length of source buffer
    WORD         uTargetLength;       // Length of target buffer
}WCDDE_CONV;

typedef union tagDDE_CONV
{
     DDEPOKE     DDEpoke;
     WCDDE_CONV     DDEConv;
}DDE_CONV;

typedef DDE_CONV FAR *LPDDE_CONV;
```

変換の形式

```
 ConvType = 0x01   ASCII to EBCDIC
 ConvType = 0x02   EBCDIC to ASCII
```

注: 変換するストリングは、複数の処理の間でアクセス可能なメモリー・ブロックに保管されていることが必要です。 Win32 では、メモリー・マップ・ファイルを使用することによって、これを行うことができます。クライアント・アプリケーションでグローバル・メモリーの作成と命名が行われ、その名前は DDE メッセージを通じてパーソナル・コミュニケーションズに送信されます。これを行うのに必要なステップを、以下の例に示します。

```
//Steps for a Source Buffer (done in client application)
HANDLE hMapFile;
LPVOID lpMapAddress;
ATOM   aCONV;

hMapFile = CreateFileMapping((HANDLE)0xFFFFFFFF,  // not a real file
    NULL,                                         // Default security.
    PAGE_READWRITE,                               // Read/write
    (DWORD)0,                                     // Ignored
    (DWORD)nStringLength,                         // Length of string
    (LPCTSTR)szSourceName);                       // Name of
                                                  // mapping object.

If (hMapFile == NULL)
{
    MessageBox ("Could not create file-mapping Source object.");
    return;
}
// Now treat buffer like local memory
strcpy((LPSTR)lpMapAddress, szConcersionString);

// Repeat steps for a Target Buffer
.....
.....
// Set up ATOM information
```

```
aCONV = GlobalAddAtom("CONV"); // MUST be this string

// Post DDE Message Now ....

// When done with memory blocks, clean up
if (!UnmapViewOfFile(lpMapAddress))
{
     MessageBox ("Could not unmap view of Target.");
}

CloseHandle(hMapSFile);

// CODE ENDS
```

パーソナル・コミュニケーションズの応答

この関数は、DDE_POKE に対する WM_DDE_ACK メッセージで応答します。結果の値は、fsStatus ワードの高位バイトに戻ります。以下の戻りコードが有効です。

戻りコード	説明
0x0000	正常終了。
0x0200	正しくない変換タイプまたはパラメーターが指定されました。
0x0600	正しくない形式が指定されました。
0x0900	システム・エラーが発生しました。
0x1000	宛先バッファーが超過しました。
0x1100	内部変換エラーが起きました。

Find Field

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

Find Field 関数は、指定されたフィールドの情報をクライアントに戻します。これは、2 つの方法で使用することができます。

メッセージを次のように送信します。

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
             MAKELPARAM(cfFormat, aFIELD) );
```

ここで、

cfFormat

フィールド情報の形式を識別します。この値は、CF_DSPTEXT または CF_TEXT とすることができます。

aFIELD

Find Field 関数を指定するアトムです。このアトムで識別されるストリングは、**cfFormat** の値に基づいて異なる値を持つことができます。

CF_DSPTEXT

cfFormat に **CF_DSPTEXT** を指定する場合は、**aFIELD** は、ストリング FIELD を表すアトムでなければなりません。PS 位置は、先行する **Set Presentation Space Service Condition** 関数の呼び出しによって指定されている必要があります。このバージョンでは、その位置を含んでいるフィールドについての情報のみが戻されます。この情報は、WM_DDE_DATA(hData, aFIELD) メッセージで戻されます。ここで、

hData は、次のものを表します。

```
typedef struct tagFINDFIELD
{
  unsigned char
  data[sizeof(DDEDATA)-1];
  unsigned short uFieldStart;   //Field start position
  unsigned short uFieldLength;  //Field Length
  unsigned char  cAttribute;    //Attribute character value
  unsigned char  ubReserved;    //reserved, no information for client
} FINDFIELD;

typedef union tagDDE_FINDFIELD
{
  DDEDATA   DDEdata;
  FINDFIELD DDEfield;
} DDE_FINDFIELD, *lpDDE_FINDFIELD;
```

CF_TEXT

cfFormat に **CF_TEXT** を指定する場合は、 **aFIELD** は、ストリング FIELD (pos, "XX") を表すアトムでなければなりません。ここで、

pos PS 位置です。

XX どのフィールドの情報を戻すのか、その対象フィールドを **pos** との相対で表すコードです。これらのコードを以下に示します。

タイプ	意味
␣␣ または T␣	**pos** を含んでいるフィールド。
P␣	**pos** に先行するフィールド。保護または無保護。
PP	**pos** に先行する保護フィールド。
PU	**pos** に先行する無保護フィールド。
N␣	**pos** の次のフィールド。保護または無保護。
NP	**pos** の次の保護フィールド。
NU	**pos** の次の無保護フィールド。

注: ␣ 記号は、必要なブランクを表します。

これらのコードは、上に示したように引用符で囲む必要があります。この情報は、WM_DDE_DATA(hData, aFIELD) メッセージで戻されます。ここで、

hData は、次のものを表します。

```
typedef struct tagFINDFIELD_CF_TEXT
{
  uchar  data[sizeof(DDEDATA)-1];
  uchar  Fielddata[80];
} FINDFIELD_CF_TEXT;
```

```
typedef FINDFIELD_CF_TEXT FAR *LPFINDFIELD_CF_TEXT;

typedef union tagDDE_FIELD
{
  DDEDATA   DDEdata;
  FINDFIELD DDEFindField;
  FINDFIELD_CF_TEXT DDEFindField_cftext;
} DDE_FIELD;

typedef DDE_FIELD FAR *LPDDE_FIELD;
```

パーソナル・コミュニケーションズの応答

関数が成功すると、上に示した情報とともに WM_DDE_DATA メッセージが戻ります。機能が失敗すると、WM_DDE_ACK(wStatus, aFIELD) が戻ります。結果の値は、wStatus ワードの低位バイトに戻ります。以下の戻りコードが有効です。

戻りコード	説明
0x0001	PS 位置が無効です。
0x0002	PS が定様式化されていません。
0x0006	無効な形式が指定されました。
0x0009	システム・エラーが発生しました。

フィールド情報の構造体

フィールド情報は、次の形式のストリングとして、 FINDFIELD_CF_TEXT 構造体の **Fielddata** メンバーに戻ります。

3270 の場合:

```
"Formatted¥t%01d¥t%01d¥t%01d¥t%01d¥t%04d¥t%04d"
```

FA ビット 2	無保護/保護	0 または 1
FA ビット 3	英数字/数字	0 または 1
FA ビット 4 ～ 5	輝度/高/通常	1、2 または 3
FA ビット 7	無変更/変更	0 または 1
開始位置	フィールドの開始位置 (FA を除く)	
長さ	フィールド長 (FA を除く)	

注: FA = フィールド属性

5250 の場合:

```
"Formatted¥t%01d¥t%01d¥t%01d¥t%01d¥t%01d¥t%01d¥t%04d¥t%04d"
```

FA ビット 0	フィールド属性フラグ	0 または 1
FA ビット 1	可視/不可視	0 または 1
FA ビット 2	無保護/保護	0 または 1
FA ビット 3	輝度 低/高	0 または 1

FA ビット 4 ～ 6	Field Type 0 = 英数字 1 = 英字 2 = 数字シフト 3 = 数字 4 = デフォルト値 5 = 数字のみ 6 = 磁気ストライプ読み取り装置データ 7 = 符号付き数値	0 ～ 7
FA ビット 7	無変更/変更	0 または 1
開始位置	フィールドの開始位置 (FA を除く)	
長さ	フィールド長 (FA を除く)	

注: FA = フィールド属性

Get Keystrokes

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

Get Keystrokes 関数は、**Start Keystroke Intercept** 関数によって代行受信したキー・ストロークをクライアントへ戻します。クライアントはキー・ストローク情報を受け取るために次のメッセージを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
             MAKELPARAM(cfFormat, aKEYS) );
```

ここで、

cfFormat
キー・ストローク情報の形式を識別します。これは、CF_DSPTEXT でなければなりません。

aKEYS
キー・ストローク・データ・アイテムを識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズは、キー・ストロークを DDE データ・メッセージ 内に戻すか、または状況情報が入った次の ACK メッセージの 1 つで応答します。

- WM_DDE_DATA(hData, aKEYS)
- WM_DDE_ACK(wStatus, aKEYS)

パーソナル・コミュニケーションズがキー・ストローク情報を戻せない場合は、次の状況コードのうちの 1 つが wStatus ワードの下位バイトに戻されます。

戻りコード	説明
2	キー・ストロークは代行受信されませんでした。
6	無効な形式が指定されました。
9	システム・エラーが発生しました。

キー・ストローク情報の構造体

パーソナル・コミュニケーションズは、キー・ストローク情報を次の構造体で戻します。

```
typedef struct tagKEYSTROKE
{
  unsigned char  data[(sizeof(DDEDATA)-1)];
  unsigned short uTextType;      /* Type of keystrokes
  unsigned char  szKeyData_1¨;   /* Keystrokes
} KEYSTROKE;

typedef union tagDDE_GETKEYSTROKE
{
  DDEDATA     DDEdata;
  KEYSTROKE   DDEkey;
} DDE_GETKEYSTROKE, *lpDDE_GETKEYSTROKE;
```

キー・ストローク・パラメーターの形式は、**Session Execute Macro** 関数の SENDKEY コマンドのパラメーターと同じです。

次のキー・テキスト・タイプがサポートされています。

```
 PCS_PURETEXT   0         /* Pure text, no HLLAPI commands
 PCS_HLLAPITEXT 1         /* Text, including HLLAPI tokens
```

Get Mouse Input

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

Get Mouse Input 関数は、 **Start Mouse Input Intercept** 関数によって代行受信した最新のマウス入力をクライアントへ戻します。

注: クライアントは、この関数を使用する前に **Start Mouse Input Intercept** 関数を呼び出す必要があります。

クライアントはマウス入力情報を受け取るために次のコマンドを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
             MAKELPARAM(cfFormat, aMOUSE) );
```

ここで、

cfFormat

表示スペースの形式を識別します。有効値は、CF_TEXT または CF_DSPTEXT です。これらの 2 つの形式でのマウス入力データの構造体については、後で示します。

aMOUSE
マウス入力をアイテムとして識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズは、マウス入力データを DDE データ・メッセージ内に戻すか、または、次の ACK メッセージの 1 つで応答します。

- WM_DDE_DATA(hData, aMOUSE)
- WM_DDE_ACK(wStatus, aMOUSE)

パーソナル・コミュニケーションズがマウス入力情報を戻せない場合は、次の状況コードのうち 1 つが wStatus ワードの下位バイト内に戻されます。

戻りコード	説明
2	マウス入力情報は、代行受信されませんでした。
6	無効な形式が指定されました。
9	システム・エラーが発生しました。

マウス入力情報の構造体

形式が CF_TEXT の場合は、パーソナル・コミュニケーションズは以下の形式でマウス入力情報を戻します。

```
typedef struct tagMOUSE_CF_TEXT
{
  unsigned char data[(sizeof(DDEDATA)-1)];
  unsigned char PSPos[4];       /* PS Offset - Mouse position
  unsigned char Tab1[1];        /* Tab character
  unsigned char PSRowPos[4];    /* ROW number of Mouse position
  unsigned char Tab2[1];        /* Tab character
  unsigned char PSColPos[4];    /* Col number of Mouse position
  unsigned char Tab3[1];        /* Tab character
  unsigned char PSSize[4];      /* Size of Presentation Space
  unsigned char Tab4[1];        /* Tab character
  unsigned char PSRows[4];      /* Row number of PS
  unsigned char Tab5[1];        /* Tab character
  unsigned char PSCols[4];      /* Column number of PS
  unsigned char Tab6[1];        /* Tab character
  unsigned char Button[1];      /* Type of clicked mouse button
  unsigned char Tab7[1];        /* Tab character
  unsigned char Click[1];       /* Type of clicking
  unsigned char Tab8[1];        /* Tab character
  unsigned char zClickString[1];/* Retrieved string
} MOUSE_CF_TEXT;

typedef union tagDDE_MOUSE_CF_TEXT
{
  DDEDATA  DDEdata;
  MOUSE_CF_TEXT  DDEmouse;
} DDE_MOUSE_CF_TEXT, *lpDDE_MOUSE_CF_TEXT;
```

次の表はパラメーターの値を示しています。

パラメーター名	意味	値
PSPos	マウスがクリックされた位置の PS オフセット	0 ... (PSSize - 1)

パラメーター名	意味	値
PSRowPos	マウスがクリックされた位置の行番号	0 ... (PSRows - 1)
PSColPos	マウスがクリックされた位置の桁番号	0 ... (PSCols - 1)
PSSize	表示スペースのサイズ	
PSRows	表示スペースの行数	
PSCols	表示スペースの桁数	
ButtonType	クリックされたマウス・ボタンのタイプ	**L** 左ボタン **M** 中央ボタン **R** 右ボタン
ClickType	クリックのタイプ	**S** シングルクリック **D** ダブルクリック
ClickString	マウスが指示した取り出しストリング	'\0' で終了する文字ストリング
Tab1〜8	区切り文字としてのタブ文字	'\t'

形式が CF_DSPTEXT の場合は、パーソナル・コミュニケーションズは以下の形式でマウス入力情報を戻します。

```
typedef struct tagMOUSE_CF_DSPTEXT
{
  unsigned char  data[(sizeof(DDEDATA)-1)];
  unsigned short uPSPos;                    /* PS Offset of the Mouse position
  unsigned short uPSRowPos;                 /* ROW number of Mouse position
  unsigned short uPSColPos;                 /* Column number of Mouse position
  unsigned short uPSSize;                   /* Size of Presentation Space
  unsigned short uPSRows;                   /* Row number of PS
  unsigned short uPSCols;                   /* Column number of PS
  unsigned short uButtonType;               /* Type of clicked mouse button
  unsigned short uClickType;                /* Type of clicking
  unsigned char  zClickString[1];             /* Retrieved string
} MOUSE_CF_DSPTEXT;

typedef union tagDDE_MOUSE_CF_DSPTEXT
{
  DDEDATA           DDEdata;
  MOUSE_CF_DSPTEXT  DDEmouse;
} DDE_MOUSE_CF_DSPTEXT, *lpDDE_MOUSE_CF_DSPTEXT;
```

次の表はパラメーターの値を示しています。

パラメーター名	意味	値
uPSPos	マウスがクリックされた位置の PS オフセット	0 ... (uPSSize - 1)
uPSRowPos	マウスがクリックされた位置の行番号	0 ... (uPSRows - 1)
uPSColPos	マウスがクリックされた位置の桁番号	0 ... (uPSCols - 1)
uPSSize	表示スペースのサイズ	

パラメーター名	意味	値
uPSRows	表示スペースの行数	
uPSCols	表示スペースの桁数	
uButtonType	クリックされたマウス・ボタンのタイプ	**0x0001** 左ボタン **0x0002** 中央ボタン **0x0003** 右ボタン
uClickType	クリックのタイプ	**0x0001** シングルクリック **0x0002** ダブルクリック
szClickString	マウスが指示したストリング	'\0' で終了する文字ストリング

Get Number of Close Requests

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

Get Number of Close Requests 関数は、 **Start Close Intercept** 関数によって代行したクローズ要求回数をクライアントへ戻します。クライアントはクローズ要求回数を受け取るために次のメッセージを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
             MAKELPARAM(cfFormat, aCLOSE) );
```

ここで、

cfFormat
クローズ・インターセプト情報の形式を識別します。これは、CF_DSPTEXT でなければなりません。

aCLOSE
クローズ・インターセプト・データ・アイテムを識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズはクローズ要求回数を DDE データ・メッセージ内に戻すか、または、状況情報が入った、次の ACK メッセージの 1 つで応答します。

- WM_DDE_DATA(hData, aCLOSE)
- WM_DDE_ACK(wStatus, aCLOSE)

パーソナル・コミュニケーションズがクローズ・インターセプト情報を戻せない場合は、次の状況コードのうち 1 つが wStatus ワードの下位バイト内に戻されます。

戻りコード	説明
6	無効な形式が指定されました。
9	システム・エラーが発生しました。

クローズ要求回数情報の構造体

パーソナル・コミュニケーションズは、クローズ・インターセプト情報を次の構造体で戻します。

```
typedef struct tagCLOSEREQ
{
  unsigned char  data[(sizeof(DDEDATA)-1)];
  unsigned short uCloseReqCount; /* Number of the close requests.
} CLOSEREQ;

typedef union tagDDE_CLOSEREQ
{
  DDEDATA   DDEdata;
  CLOSEREQ  DDEclose;
} DDE_CLOSEREQ, *lpDDE_CLOSEREQ;
```

Get Operator Information Area

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

Get Operator Information Area (OIA) 関数は、 OIA のコピーをクライアントへ戻します。クライアントは OIA を要求するために次のメッセージを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
             MAKELPARAM(cfFormat, aOIA) );
```

ここで、

cfFormat
OIA の形式を識別します。 OIA の場合、この形式は CF_DSPTEXT でなければなりません。

aOIA オペレーター情報域をアイテムとして識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズは OIA を DDE データ・メッセージ内に戻すか、または、次の ACK メッセージの 1 つで応答します。

- WM_DDE_DATA(hData, aOIA)
- WM_DDE_ACK(wStatus, aOIA)

パーソナル・コミュニケーションズが OIA を戻せない場合は、次の状況コードのうち 1 つ が wStatus ワードの下位バイト内に戻されます。

戻りコード	説明
6	無効な形式が指定されました。
9	システム・エラーが発生しました。

オペレーター情報域の構造体

パーソナル・コミュニケーションズは、オペレーター情報域を次の構造体で戻します。

```
typedef struct tagOIADATA
{
  unsigned char  data[(sizeof(DDEDATA)-1)];
  unsigned char  OIA[80];
} OIADATA;

typedef union tagDDE_OIADATA
{
  DDEDATA   DDEdata;
  OIADATA   DDEoia;
} DDE_OIADATA, *lpDDE_OIADATA;
```

Get Partial Presentation Space

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

Get Partial Presentation Space 関数は、セッション表示スペースの一部または全部をクライアントへ戻します。

注: クライアントはこの関数を使用する前に、**Set Presentation Space Service Condition** 関数を使用して PS 開始位置と PS 長またはフィールドの終わり (EOF) フラグを設定する必要があります。EOF フラグが PCS_EFFECTEOF に設定されていると、この関数は、PS 開始位置パラメーターの指定するフィールド全体を戻します。

クライアントは表示スペースを獲得するために次のコマンドを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
             MAKELPARAM(cfFormat, aEPS) );
```

ここで、

cfFormat
表示スペースの形式を識別します。有効値は、CF_TEXT または CF_DSPTEXT です。これら 2 つの形式での表示スペースの構造体については、後で示します。

aEPS セッション表示スペースをアイテムとして識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズは、表示スペース・データを戻すか、または、wStatus ワードの下位バイトにエラー・コードが入った、次の ACK メッセージの 1 つで応答します。

- WM_DDE_DATA(hData, aEPS)
- WM_DDE_ACK(wStatus, aEPS)

パーソナル・コミュニケーションズが表示スペースを戻せない場合は、次の状況コードのうち 1 つが wStatus ワードの下位バイト内に戻されます。

戻りコード	説明
1	事前に **Set Presentation Space Service Condition** 関数が呼び出されなかった、または無効なパラメーターが設定されました。
6	無効な形式が指定されました。
9	システム・エラーが発生しました。

表示スペースの構造体

パーソナル・コミュニケーションズは、**Get Partial Presentation Space** 要求に指定された形式で表示スペースの一部を戻します。

形式が CF_DSPTEXT の場合は、パーソナル・コミュニケーションズは次の形式で表示スペースを戻します。

```
typedef struct tagEPS_CF_DSPTEXT
{
  unsigned char  data[(sizeof(DDEDATA)-1)];
  unsigned short uPSPosition;       /* Position of the part of PS
  unsigned short uPSLength;         /* Length of the part of the PS
  unsigned short uPSRows;           /* PS number of rows
  unsigned short uPSCols;           /* PS number of columns
  unsigned short uPSOffset;         /* Offset to the presentation space
  unsigned short uFieldCount;       /* Number of fields
  unsigned short uFieldOffset;      /* Offset to the field array
  unsigned char  PSData[1];     /* PS + Field list Array(lpPSFIELDS)
} EPS_CF_DSPTEXT;

typedef union tagDDE_EPS_CF_DSPTEXT
{
  DDEDATA         DDEdata;
  EPS_CF_DSPTEXT  DDEeps;
} DDE_EPS_CF_DSPTEXT, *lpDDE_EPS_CF_DSPTEXT;

# The PSFIELDS structure is replaced with below structure.

typedef struct tagPSFIELDS
{
  unsigned short uFieldStart;    /* Field start offset
  unsigned short uFieldLength;   /* Field Length
  unsigned char  cAttribute;     /* Attribute character
  unsigned char  ubReserved;     /* *** Reserved ***
} PSFIELDS, *lpPSFIELDS;
```

注: 次の例は、PS および PSFIELDS 配列に対して long 型のポインターを取得する方法を示しています。

```
lpDDE = (lpDDE_EPS_CF_DSPTEXT)GlobalLock(hData);
lpps = lpDDE->DDEeps.PSData + lpDDE->DDEeps.uPSOffset;
lppsfields = lpDDE->DDEeps.PSData + lpDDE->DDEeps.uFieldOffset;
```

形式が CF_TEXT の場合、パーソナル・コミュニケーションズは次の形式で表示スペースの一部を戻します。

```
typedef struct tagEPS_CF_TEXT
{
  unsigned char  data[(sizeof(DDEDATA)-1)];
  unsigned char  PSPOSITION[4];/* Position of part of the PS
  unsigned char  Tab1[1];      /* Tab character
```

```
  unsigned char  PSLENGTH[4];  /* Length of the part of the PS
  unsigned char  Tab2[1];      /* Tab character
  unsigned char  PSROWS[4];    /* Number of rows in the PS
  unsigned char  Tab3[1];      /* Tab character
  unsigned char  PSCOLS[4];    /* Number of Cols in the PS
  unsigned char  Tab4[1];      /* Tab character
  unsigned char  PS[1];        /* PS
} EPS_CF_TEXT;

typedef union tagDDE_EPS_CF_TEXT
{
  DDEDATA      DDEdata;
  EPS_CF_TEXT  DDEeps;
} DDE_EPS_CF_TEXT, *lpDDE_EPS_CF_TEXT;
```

バッファー内の PS の後に、フィールド・リストを構成する、次のようなフィールドの追加構造体が続きます。

```
typedef struct tagFL_CF_TEXT
{
  unsigned char  Tab5[1];        /* Tab character
  unsigned char  PSFldCount[4];  /* Number of fields in the PS
  unsigned char  Tab6[1];        /* Tab character
  PS_FIELD       Field[1];       /* Field List Array
} FL_CF_TEXT, *lpFL_CF_TEXT;

typedef struct tagPS_FIELD
{
  unsigned char  FieldStart[4];
  unsigned char  TabF1[1];
  unsigned char  FieldLength[4];
  unsigned char  TabF2[1];
```

注: 次の例は、PS および PS_FIELD 配列に対して long 型のポインターを取得する方法を示しています。

```
lpDDE = (lpDDE_EPS_CF_TEXT)GlobalLock(hData);
lpps = lpDDE->DDEeps.PS;
lpps_field = lpDDE->DDEeps.PS
           + atoi(lpDDE->DDEeps.PSLENGTH)
           + ((atoi(lpDDE->DDEeps.PSROWS) -1) * 2) // CR/LF
           + 1 + 1 + 4 + 1;  // Tabs + size of field count
```

Get Presentation Space

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

Get Presentation Space 関数は、セッション表示スペースをクライアントへ戻します。クライアントは表示スペースを獲得するために次のコマンドを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
             MAKELPARAM(cfFormat, aPS) );
```

ここで、

cfFormat

表示スペースの形式を識別します。有効値は、CF_TEXT または CF_DSPTEXT です。これら 2 つの形式での表示スペースの構造体については、後で示します。

aPS セッション表示スペースをアイテムとして識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズは、表示スペースおよびその表示スペースを構成するフィールドのリストを戻すか、または、wStatus ワードの下位バイトにエラー・コードが入った、次の ACK メッセージの 1 つで応答します。

- WM_DDE_DATA(hData, aPS)
- WM_DDE_ACK(wStatus, aPS)

パーソナル・コミュニケーションズが表示スペースを戻せない場合は、次の状況コードのうち 1 つが wStatus ワードの下位バイト内に戻されます。

戻りコード	説明
6	無効な形式が指定されました。
9	システム・エラーが発生しました。

表示スペースの構造体

パーソナル・コミュニケーションズは、**Get Presentation Space** 要求に指定された形式で表示スペースを戻します。

形式が CF_DSPTEXT の場合は、パーソナル・コミュニケーションズは次の形式で表示スペースを戻します。

```
typedef struct tagPS_CF_DSPTEXT
{
  unsigned char  data[(sizeof(DDEDATA)-1)];
  unsigned short uPSSize;        /* Size of the presentation space
  unsigned short uPSRows;        /* PS number of rows
  unsigned short uPSCols;        /* PS number of columns
  unsigned short uPSOffset;      /* Offset to the presentation space
  unsigned short uFieldCount;    /* Number of fields
  unsigned short uFieldOffset;   /* Offset to the field array
  unsigned char  PSData_1¨;      /* PS and Field list Array(lpPSFIELDS)
} PS_CF_DSPTEXT;

typedef union tagDDE_PS_CF_DSPTEXT
{
  DDEDATA        DDEdata;
  PS_CF_DSPTEXT  DDEps;
} DDE_PS_CF_DSPTEXT, *lpDDE_PS_CF_DSPTEXT;

typedef struct tagPSFIELDS
{
  unsigned short uFieldStart;    /* Field start offset
  unsigned short uFieldLength;   /* Field Length
  unsigned char  cAttribute;     /* Attribute character
  unsigned char  ubReserved;     /* *** Reserved ***
} PSFIELDS, *lpPSFIELDS;
```

注: 次の例は、PS および PSFIELDS 配列に対して long 型のポインターを取得する方法を示しています。

```
lpDDE = (lpDDE_PS_CF_DSPTEXT)GlobalLock(hData);
lpps = lpDDE->DDEps.PSData + lpDDE->DDEps.uPSOffset;
lppsfields = lpDDE->DDEps.PSData + lpDDE->DDEps.uFieldOffset;
```

形式が CF_TEXT の場合は、パーソナル・コミュニケーションズは次の形式で表示スペースを戻します。

```
typedef struct tagPS_CF_TEXT
{
  unsigned char  data[(sizeof(DDEDATA)-1)];
  unsigned char  PSSIZE[4];      /* Size of the PS
  unsigned char  Tab1[1];        /* Tab character
  unsigned char  PSROWS[4];      /* Number of rows in the PS
  unsigned char  Tab2[1];        /* Tab character
  unsigned char  PSCOLS[4];      /* Number of Cols in the PS
  unsigned char  Tab3[1];        /* Tab character
  unsigned char  PS[1];          /* PS
} PS_CF_TEXT;

typedef union tagDDE_PS_CF_TEXT
{
  DDEDATA     DDEdata;
  PS_CF_TEXT  DDEps;
} DDE_PS_CF_TEXT, *lpDDE_PS_CF_TEXT;
```

バッファー内の PS の後に、フィールド・リストを構成する、次のようなフィールドの追加構造体が続きます。

```
typedef struct tagPS_FIELD
{
  unsigned char  FieldStart[4];
  unsigned char  TabF1[1];
  unsigned char  FieldLength[4];
  unsigned char  TabF2[1];
  unsigned char  Attribute;
  unsigned char  TabF3[1];
} PS_FIELD, *lpPS_FIELD;
```

注: 次の例は、PS および PS 配列に対して long 型のポインターを取得する方法を示しています。

```
lpDDE = (lpDDE_PS_CF_TEXT)GlobalLock(hData);
lpps = lpDDE->DDEps.PS;
lpps_field = lpDDE->DDEps.PS
           + atoi(lpDDE->DDEps.PSSIZE)
           + ((atoi(lpDDE->DDEps.PSROWS) -1) * 2) // CR/LF
           + 1 + 1 + 4 + 1;  // Tabs + size of field count
```

Get Session Status

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

Get Session Status 関数は、接続したセッションの状況を戻します。クライアントは、セッション状況を要求するために次のメッセージを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
             MAKELPARAM(cfFormat, aSSTAT) );
```

ここで、

cfFormat
状況情報の DDE 形式を識別します。使用する値は CF_TEXT です。

aSSTAT
要求するデータ・アイテムとしてセッション状況を識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズは、セッション状況を DDE データ・メッセージ内に戻すか、または状況情報が入った ACK メッセージの 1 つで応答します。

- WM_DDE_DATA(hData, aSSTAT)
- WM_DDE_ACK(wStatus, aSSTAT)

パーソナル・コミュニケーションズがセッション状況を戻せない場合は、次の状況コードのうち 1 つが wStatus ワードの下位バイト内に戻されます。

戻りコード	説明
6	無効な形式が指定されました。
9	システム・エラーが発生しました。

状況情報の形式

パーソナル・コミュニケーションズは、セッション状況を CF_TEXT 形式のテキストとして戻します。次の考えられる値と共に、次のフィールドが戻されます。

フィールド	戻される値	説明
状況	Closed、Invisible、Maximized、Minimized、Normal	ウィンドウはこれらの状態のいずれかです。
使用法	DDE、User	セッションは DDE セッションまたはユーザー・セッションで接続されています。
スクリーン X	NN	画面の水平方向のサイズを定義します。
スクリーン Y	NN	画面の垂直方向のサイズを定義します。
カーソル X	NN	カーソルの水平方向の位置を定義します。 (0 ... ScreenX - 1)
カーソル Y	NN	カーソルの垂直方向の位置を定義します。 (0 ... ScreenY - 1)
トリミング長方形状況	Closed、Moved、Sized	トリミング長方形の現在の設定。
トリミング長方形 X1	N	トリミング長方形の左上隅の X 位置 (文字座標)

フィールド	戻される値	説明
トリミング長方形 Y1	N	トリミング長方形の左上隅の Y 位置 (文字座標)
トリミング長方形 X2	N	トリミング長方形の右下隅の X 位置 (文字座標)
トリミング長方形 Y2	N	トリミング長方形の右下隅の Y 位置 (文字座標)
セッション表示スペース状況	N	表示スペースの現在の設定。次の値が考えられます。 **0:** 表示スペースはロックされていません。 **4:** 表示スペースは使用中です。 **5:** 表示スペースはロックされています。
セッション・ウィンドウ・ハンドル	XXXX	セッションのウィンドウ・ハンドル

注意:

- 各フィールド状況は、その状況が要求されるたびに更新されます。
- パーソナル・コミュニケーションズの将来のバージョンでは、新しいフィールドが追加されることもあります。

Get System Configuration

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

Get System Configuration 関数は、パーソナル・コミュニケーションズがサポートするレベルとその他のシステムに関連した値を戻します。この情報のほとんどは、ユーザーがシステム・エラーを受け取った後、IBM のサポート部門へ連絡したときにサービス・コーディネーターが使用するためのものです。

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
             MAKELPARAM(cfFormat, aSYSCON) );
```

ここで、

cfFormat

要求するデータ・アイテムの DDE 形式を識別します。使用する値は CF_TEXT です。

aSYSCON

要求するデータ・アイテムとしてシステム構成を識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズはシステム構成データ・アイテムを DDE DATA メッセージ内に戻すか、または状況情報が入った ACK メッセージの 1 つで応答します。

- WM_DDE_DATA(hData, aSYSCON)
- WM_DDE_ACK(wStatus, aSYSCON)

パーソナル・コミュニケーションズがシステム構成を戻せない場合は、wStatus ワードの下位バイトにエラー・コードが入った、次の DDE ACK メッセージが戻されます。

```
WM_DDE_ACK(wStatus, aSYSCON)
```

戻りコード	説明
9	システム・エラーが発生しました。

システム構成情報の形式

パーソナル・コミュニケーションズは、システム構成を CF_TEXT 形式のテキストとして戻します。次の考えられる値と共に、次のフィールドが戻されます。

フィールド	戻される値	説明
バージョン	N	パーソナル・コミュニケーションズのバージョン
レベル	NN	パーソナル・コミュニケーションズのレベル
予約済み	XXXXXX	予約済み
予約済み	XXXX	予約済み
モニター・タイプ	MONO、CGA、EGA、VGA、XGA	モニターのタイプ
国コード	NNNN	3270 または 5250 で使用される国コード

Get System Formats

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

Get System Formats 関数は、パーソナル・コミュニケーションズがサポートする Windows クリップボード形式のリストを戻します。クライアント・アプリケーションは、パーソナル・コミュニケーションズがサポートする形式リストを取り出すために次のメッセージを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
             MAKELPARAM(cfFormat, aFORMATS) );
```

ここで、

cfFormat
要求するデータ・アイテムの DDE 形式を識別します。使用する値は CF_TEXT です。

aFORMATS
要求するデータ・アイテムとして形式を識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズは、サポートしている Windows クリップボード形式のリストを CF_TEXT 形式で DDE DATA メッセージ内に戻します。

```
WM_DDE_DATA(hData, aFORMATS)
```

パーソナル・コミュニケーションズは、次の Windows クリップボード形式をサポートしています。

- CF_TEXT
- CF_DSPTEXT

パーソナル・コミュニケーションズが形式データ・アイテムを戻せない場合は、wStatus ワードの下位バイトにエラー・コードが入った、次の DDE ACK メッセージが戻されます。

```
WM_DDE_ACK(wStatus, aFORMATS)
```

戻りコード	説明
9	システム・エラーが発生しました。

Get System Status

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

Get System Status 関数は、現在のパーソナル・コミュニケーションズの構成で使用できる各 3270 または 5250 のセッションの状況を戻します。クライアント・アプリケーションは、状況データ・アイテムを取り出すために次のメッセージを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
             MAKELPARAM(cfFormat, aSTATUS) );
```

ここで、

cfFormat
要求するデータ・アイテムの DDE 形式を識別します。使用する値は CF_TEXT です。

aSTATUS
要求するデータ・アイテムとして状況を識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズは、状況データ・アイテムを CF_TEXT 形式で DDE DATA メッセージ内に戻します。

```
WM_DDE_DATA(hData, aSTATUS)
```

パーソナル・コミュニケーションズは、オープンされている各セッションごとに状況情報行を戻します。それぞれの状況情報行には、次の範囲の値を持つ一連のフィールドが入っています。

フィールド	値の範囲	説明
セッション ID	A、B、...、Z	セッションの短縮 ID
ホスト・タイプ	370、400、ASCII	パーソナル・コミュニケーションズが現在サポートしているホスト・システム
エミュレーション・タイプ	3270、5250、VT	パーソナル・コミュニケーションズがサポートしているエミュレーション・タイプ
セッション状況	Closed、Invisible、Normal、Minimized、Maximized	セッションのウィンドウの現在の設定

パーソナル・コミュニケーションズが状況データ・アイテムを戻せない場合は、wStatus ワードの下位バイトにエラー・コードが入った、次の DDE ACK メッセージが戻されます。

```
WM_DDE_ACK(wStatus, aSTATUS)
```

戻りコード	説明
9	システム・エラーが発生しました。

Get System SysItems

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

パーソナル・コミュニケーションズは、DDE システム・トピックをサポートしているので、クライアント・アプリケーション はシステム・トピックに接続し、パーソナル・コミュニケーションズに関する情報と、パーソナル・コミュニケーションズが管理しているセッションの状況についての情報を取得できます。

Get System SysItems 関数は、パーソナル・コミュニケーションズのシステム・トピック内で使用できるデータ・アイテムのリストを戻します。クライアント・アプリケーションは、システム・トピック・データ・アイテムを取得するために次のメッセージを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
             MAKELPARAM(cfFormat, aSYSITEMS) );
```

ここで、

cfFormat
要求するデータ・アイテムの DDE 形式を識別します。使用する値は CF_TEXT です。

aSYSITEMS
要求するデータ・アイテムとして SysItems を識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズは、システム・トピック・データ・アイテムのリストを CF_TEXT 形式で DDE DATA メッセージ内に戻します。

```
WM_DDE_DATA(hData, aSYSITEMS)
```

パーソナル・コミュニケーションズでサポートされるデータ・アイテムは次のとおりです。

- SysItems
- Topics
- Status
- Formats
- SysCon

パーソナル・コミュニケーションズがシステム・データ・アイテムを戻せない場合は、wStatus ワードの下位バイトにエラー・コードが入った、次の DDE ACK メッセージが戻されます。

```
WM_DDE_ACK(wStatus, aSYSITEMS)
```

戻りコード	説明
9	システム・エラーが発生しました。

Get System Topics

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

Get System Topics 関数は、パーソナル・コミュニケーションズが現在サポートしている活動状態の DDE トピックのリストを戻します。クライアント・アプリケーションは、現在活動状態にあるトピックのリストを取り出すために次のメッセージをシステム・トピックへ送信します。

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
             MAKELPARAM(cfFormat, aTOPICS) );
```

ここで、

cfFormat
要求するデータ・アイテムの DDE 形式を識別します。使用する値は CF_TEXT です。

aTOPICS
要求するデータ・アイテムとしてトピックを識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズは、DDE トピックのリストを CF_TEXT 形式で DDE DATA メッセージ内に戻します。

```
WM_DDE_DATA(hData, aTOPICS)
```

パーソナル・コミュニケーションズでは、次のトピックのリストをサポートしています。

- System - システム・トピック
- SessionA - セッション A トピック

  .
  .
  .
- SessionZ - セッション Z トピック

注: サポートされるセッション・トピックの実際の数は、現在オープンされているセッションの数によって決まります。クライアント・プログラムは、常にシステム・トピックのトピック・データ・アイテムを問い合わせて、現在オープンされているセッションのリストを取得する必要があります。

パーソナル・コミュニケーションズがトピックのリストを戻せない場合は、wStatus ワードの下位バイトにエラー・コードが入った、次の DDE ACK メッセージが戻されます。

```
WM_DDE_ACK(wStatus, aTOPICS)
```

戻りコード	説明
9	システム・エラーが発生しました。

Get Trim Rectangle

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

Get Trim Rectangle 関数は、表示スペースのうち、現在のトリミング長方形内にある区域をクライアントへ戻します。クライアントはトリミング長方形を受け取るために次のメッセージを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
             MAKELPARAM(cfFormat, aTRIMRECT) );
```

ここで、

cfFormat
トリミング長方形の形式を識別します。これは CF_TEXT です。

aTRIMRECT
要求するデータ・アイテムとしてトリミング長方形を識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズはトリミング長方形を DDE データ・メッセージ内に戻すか、または、次のうちどちらかの ACK メッセージで応答します。

- WM_DDE_DATA(hData, aTRIMRECT)
- WM_DDE_ACK(wStatus, aTRIMRECT)

パーソナル・コミュニケーションズがトリミング長方形を戻せない場合は、次の状況コードのうち 1 つが wStatus ワードの下位バイト内に戻されます。

戻りコード	説明
6	無効な形式が指定されました。
9	システム・エラーが発生しました。

Initiate Session Conversation

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

Initiate Session Conversation 関数はクライアント・アプリケーション を、パーソナル・コミュニケーションズの使用可能なセッションへ接続します。セッション会話が確立されると、その会話が終了するまで、そのセッションはそのクライアント専用のものとして予約されます。

クライアント・アプリケーションは、セッションと DDE 会話を開始するために次のメッセージを送信します。

```
SendMessage( -1,
             WM_DDE_INITIATE,
             hClientWnd,
             MAKELPARAM(aIBM327032, aSessionN) );
```

ここで、

aIBM327032
アプリケーション・アトムを識別します。アトム aIBM327032 を作成するために使用するストリングは IBM327032 です。PC400 の場合は、アプリケーション・アトムは aIBM525032 で、これを作成するために使用するストリングは IBM525032 です。

aSessionN
トピック・アトムを識別します。アトム aSessionN を作成するために使用するストリングは、NULL または Session にセッション ID の A、B、...、Z までのいずれかを付けたものです。

パーソナル・コミュニケーションズの応答

特定のトピックが選択され、しかもパーソナル・コミュニケーションズがクライアント・アプリケーションとの会話をサポートできる場合、パーソナル・コミュニケーションズは以下のように指定して INITIATE トランザクションを確認します。

```
WM_DDE_ACK(aIBM327032, aSessionN)
```

トピックが選択されなかった場合 (aSessionN = NULL)、パーソナル・コミュニケーションズは以下のように、現在使用可能なすべてのトピックを確認することによって応答します。

```
WM_DDE_ACK(aIBM327032, aSystem)
WM_DDE_ACK(aIBM327032, aSessionA)
⋮
WM_DDE_ACK(aIBM327032, aSessionZ)
```

クライアント・アプリケーションは、戻されたトピック・リストから、通信したい会話を選択し、その他の必要ない会話をすべて終了します。

Initiate Structured Field Conversation

3270	*5250*	*VT*
Yes	*No*	*No*

Initiate Structured Field Conversation 関数は、クライアント・アプリケーションとホスト・アプリケーションを接続します。これによって、クライアント・アプリケーションとホスト・アプリケーションの間でデータを送受信できます。

クライアントは構造化フィールド会話を受け取るために次のコマンドを送信します。

```
SendMessage( -1,
             WM_DDE_INITIATE,
             hClientWnd,
             MAKELPARAM(aIBM327032, aLUN_xxxx) );
```

ここで、

aIBM327032
アプリケーション・アトムを識別します。

aLUN_xxxx
トピック・アトムを識別します。アトム aLUN_xxxx を作成するために使用するストリングは、LU にセッション ID の A、B、...、Z までのいずれかを付け、次に “_” を付け、さらに任意の長さのユーザー定義ストリングを付けたものです。

PC/3270 の応答

PC/3270 クライアント・アプリケーションとの構造化フィールド会話をサポートできる場合には、次のパラメーターを指定して肯定応答メッセージを戻します。

```
WM_DDE_ACK(aIBM327032, aLUN_xxxx)
```

Initiate System Conversation

3270	5250	VT
Yes	Yes	Yes

Initiate System Conversation 関数は、クライアント・アプリケーションをシステム会話へ接続します。システム会話へ接続できるクライアントは一度に 1 つだけです。クライアントはシステム会話を開始するために次のコマンドを送信します。

```
SendMessage( -1,
  WM_DDE_INITIATE,
  hClientWnd,
             MAKELPARAM(aIBM327032, aSystem) );
```

ここで、

aIBM327032
アプリケーション・アトムを識別します。

aSystem
トピック・アトムを識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズは、クライアント・アプリケーションとのシステム・トピック会話をサポートできる場合には、次に示すパラメーターを指定して肯定応答メッセージを戻します。

```
WM_DDE_ACK(aIBM327032, aSystem)
```

Put Data to Presentation Space

3270	5250	VT
Yes	Yes	Yes

Put Data to Presentation Space 関数は、呼び出しパラメーターで指定した位置でホスト表示スペースに書き込む、ASCIIZ データ・ストリングを送信します。クライアントはストリングを送信するために次のメッセージをセッションへ送信します。

```
PostMessage( hServerWnd,
             WM_DDE_POKE,
             hClientWnd,
             PackDDElParam(WM_DDE_POKE,
             hdata, aEPS) );
```

ここで、

hData セッションへ送信するストリングが入っている Windows グローバル・メモリー・オブジェクトへのハンドルを識別します。このグローバル・メモリー・オブジェクトには次の構造体が入っています。

```
typedef struct tagPutString
{
  unsigned char  poke[(sizeof(DDEPOKE)-1)];
  unsigned short uPSStart;                      /* PS Position
  unsigned short uEOFflag;                      /* EOF effective switch
```

```
  unsigned char  szStringData[1];                /* String Data
} PUTSTRING;

typedef union tagDDE_PUTSTRING
{
  DDEPOKE    DDEpoke;
  PUTSTRING  DDEputstring;
} DDE_PUTSTRING, *lpDDE_PUTSTRING;
```

uEOFflag フィールドには、次の値が有効です。

```
 PCS_UNEFFECTEOF 0        /* The string is not truncated at EOF.
 PCS_EFFECTEOF   1        /* The string is truncated at EOF.
```

aEPS 表示スペース・アトムをアイテムとして識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズは、ストリング・データを受け取り、それを表示スペースへ送信し、肯定の ACK メッセージを戻します。

表示スペースがストリング・データを受け入れない場合、パーソナル・コミュニケーションズは wStatus ワードの下位バイトに、次の状況コードのうち 1 つが入った否定の ACK メッセージを戻します。

```
WM_DDE_ACK(wStatus, aEPS)
```

戻りコード	説明
1	PS 位置が無効です。
2	長さが無効です。
3	EOF フラグの値が無効です。
5	宛先 PS への入力が禁止されていました。
6	無効な形式が指定されました。
7	ストリングは切り捨てられました (配置は成功)。
9	システム・エラーが発生しました。

Search for String

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

この関数を使用すると、指定したストリングが指定した区域内にあるかどうか、クライアント・アプリケーションで表示スペースを調べることができます。

注: クライアントはこの関数を使用する前に、 **Set Presentation Space Service Condition** 関数を使用して PS 開始位置と検索するストリング、および PS 長またはフィールドの終わり (EOF) フラグを設定する必要があります。 EOF フラグが PCS_EFFECTEOF に設定されていると、この関数は、PS 開始位置パラメーターの指定するフィールド全体を検索します。

クライアントはストリングを検索するために次のメッセージを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
             MAKELPARAM(cfFormat, aSTRING) );
```

ここで、

cfFormat
検索情報の形式を識別します。これは、CF_DSPTEXT でなければなりません。

aSTRING
検索データ・アイテムを識別します。

パーソナル・コミュニケーションズの応答

指定された区域内でストリングが見つかった場合、パーソナル・コミュニケーションズはそのストリングの開始位置を DDE データ・メッセージ内に戻します。

- WM_DDE_DATA(hData, aSTRING)
- WM_DDE_ACK(wStatus, aSTRING)

パーソナル・コミュニケーションズがストリングの開始位置を戻せない場合は、次の状況コードのうち 1 つが wStatus ワードの下位バイト内に戻されます。

戻りコード	説明
1	PS 位置が無効であるか、またはストリングが長すぎます。
2	ストリングが見つかりません。
6	無効な形式が指定されました。
9	システム・エラーが発生しました。

検索情報の構造体

パーソナル・コミュニケーションズは、検索情報を次の構造で戻します。

```
typedef struct tagSEARCH
{
  unsigned char  data[(sizeof(DDEDATA)-1)];
  unsigned short uFieldStart;    /* String start offset
} SEARCH;

typedef union tagSEARCH
{
  DDEDATA  DDEdata;
  SEARCH   DDEsearch;
} DDE_SEARCH, *lpDDE_SEARCH;
```

Send Keystrokes

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

Send Keystrokes 関数は、接続したセッションへキー・ストロークを送信します。クライアントはキー・ストロークを送信するために次のメッセージをセッションへ送信します。

```
PostMessage( hServerWnd,
             WM_DDE_POKE,
             hClientWnd,
             PackDDElParam(WM_DDE_POKE,
             hData, aKEYS) );
```

ここで、

hData セッションへ送信するキー・ストロークが入っている Windows グローバル・メモリー・オブジェクトへのハンドルを識別します。このグローバル・メモリー・オブジェクトには次の構造体が入っています。

```
typedef struct tagKeystrokes
{
  unsigned char  poke[(sizeof(DDEPOKE)-1)];
  unsigned short uTextType;                /* Type of keystrokes
  unsigned short uRetryCount;              /* Retry count 1 .. 16
  unsigned char  szKeyData[1];      /* Keystrokes
} KEYSTROKES;

typedef union tagDDE_SENDKEYSTROKES
{
  DDEPOKE     DDEpoke;
  KEYSTROKES  DDEkeys;
} DDE_SENDKEYSTROKES, *lpDDE_SENDKEYSTROKES;
```

次のキー・テキスト・タイプがサポートされています。

```
 PCS_PURETEXT   0         /* Pure text, no HLLAPI commands
 PCS_HLLAPITEXT 1         /* Text, including HLLAPI tokens
```

注: キー・ストロークが純粋テキストである場合、PCS_PURETEXT を指定すると、考えられる最も速い方法でキー・ストロークがホストへ転送されます。PCS_HLLAPITEXT を指定した場合、キー・ストローク・データにはテキストが含まれている HLLAPI コマンドが入っている場合があります。

aKEYS

キー・ストロークをアイテムとして識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズは、キー・ストロークを受け取り、それを表示スペースへ送信します。表示スペースがキー・ストロークを受け入れない場合、表示スペースへリセットが送信され、そのキー・ストロークが再度送信されます。この手順は、表示スペースがキー・ストロークを受け入れるか、または再試行カウントに達するまで続けられます。パーソナル・コミュニケーションズがキー・ストロークをホストへ送信できない場合、パーソナル・コミュニケーションズは wStatus ワードの下位バイトに、次の状況コードのうち 1 つが入った否定の ACK メッセージを戻します。それ以外の場合、パーソナル・コミュニケーションズは肯定の ACK メッセージを戻し、キー・ストロークの送信が完了したことを知らせます。

```
WM_DDE_ACK(wStatus, aKEYS)
```

戻りコード	説明
1	再試行カウントが無効でした。
2	キー・ストロークのタイプが無効でした。
6	無効な形式が指定されました。
9	システム・エラーが発生しました。

Session Execute Macro

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

ユーザーは、DDE_EXECUTE 関数と共にコマンドとマクロを発行できます。DDE_EXECUTE 関数は、コマンド・ストリングをパーソナル・コミュニケーションズへ渡します。コマンド・ストリングは DDE 仕様に準拠しなければなりません。DDE コマンド構文の詳細については「*Microsoft Windows Software Development Kit Guide to Programming*」を参照してください。

クライアントは、DDE_EXECUTE 関数を発行するために次のコマンドを送信します。

```
PostMessage ( hServerWnd,
              WM_DDE_EXECUTE,
              hClientWnd,
              (LPARAM)hCommands) );
```

ここで、

hCommands
パーソナル・コミュニケーションズのコマンドを含む Windows グローバル・メモリー・オブジェクトへのハンドルを識別します。発行できるコマンドのリストについては、『Session Execute Macro 関数のコマンド発行』を参照してください。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズがコマンド・ストリングを処理できる場合、パーソナル・コミュニケーションズは肯定の状況情報が入った ACK メッセージをクライアントへ戻します。パーソナル・コミュニケーションズがコマンド・ストリングを実行できない場合、パーソナル・コミュニケーションズは wStatus ワードの下位ワードに次のエラー・コードが含まれる ACK メッセージを戻します。

戻りコード	説明
9	システム・エラーが発生しました。

Session Execute Macro 関数のコマンド発行

ユーザーは **Session Execute Macro** 関数で次のコマンドを発行できます。

- KEYBOARD
- RECEIVE

- SEND
- SENDKEY
- WAIT
- WINDOW

各コマンドごとに別々の DDE_EXECUTE メッセージを使用してください。

注意:

- 非英数字またはブランクを含んでいる値は二重引用符で囲んでください (例えば、*"value value"*)。
- ストリング内に二重引用符を含めたい場合は、二重引用符を 2 回タイプします (例えば、 "This is a double quotation mark:""")。
- コマンドの最大長は 255 文字です。

WINDOW コマンド

```
[WINDOW(action[, "name"])]
```

ウィンドウ・アクションを実行します。ここで、

```
action = HIDE|RESTORE|MAXIMIZE|MINIMIZE|
         SHOW|CNGNAME
name = LT name or Switch List Entry name
```

注: *action* に CNGNAME が指定されている場合は *name* を指定してください。*name* が NULL ストリングである場合、デフォルトのタイトルが表示されます。

KEYBOARD コマンド

```
[KEYBOARD(action)]
```

マウスを含め、セッション・キーボードの使用を可能にしたり禁止したりします。

```
action= LOCK|UNLOCK
```

SEND コマンド

```
[SEND("pcfilename","hostfilename","options")]
```

ホストにファイルを送信します。ここで、

```
pcfilename = [path]filename[.ext]
hostfilename =
  For VM system:
    filename filetype[filemode]
  For MVS system:
     ['][filename[(membername)]['']
  For CICS system:
  For OS/400 system:
    library name filename member name
```

以下のファイル転送オプション MVS、VM、CICS、QUIET、OS/400、 およびエミュレーター固有の転送オプションは任意に組み合わせて、スペースで区切って *options* に組み込むことができます。

転送オプションの詳細については、「*管理者ガイドおよび解説書*」を参照してください。

RECEIVE コマンド

```
[RECEIVE("pcfilename","hostfilename","options")]
```

ホストからファイルを受信します。ここで、

```
       pcfilename = [path]filename[.ext]
   hostfilename =
     For VM system:
       filename filetype[filemode]
     For MVS system:
        ['']filename[(membername)][']
     For CICS system:
     For OS/400 system:
       library name filename member name
```

以下のファイル転送オプション MVS、VM、CICS、QUIET、OS/400、 およびエミュレーター固有の転送オプションは任意に組み合わせて、スペースで区切って *options* に組み込むことができます。

転送オプションの詳細については、「*管理者ガイドおよび解説書*」を参照してください。

SENDKEY コマンド

```
[SENDKEY(token,token)]
```

パーソナル・コミュニケーションズにキー・ストロークを送信します。ここで、

```
token = text string|command|macro macroname
```

注:

1. テキスト・ストリングは二重引用符で囲みます。
2. マクロは接頭符に “macro” を付けます。
3. SENDKEY の引き数ストリングは、255 文字以下でなければなりません。
4. 次のコマンドがサポートされています。

表 19. SENDKEY コマンド・リスト

コマンド名	トークン	PC/3270	PC400	VT
カーソル切り替え	alt cursor	YES	YES	NO
文字セット切り替え	alt view	YES	YES	NO
アテンション	sys attn	YES	YES	NO
バックスペース	backspace	YES	YES	YES
後退タブ	backtab	YES	YES	NO
後退タブ・ワード	backtab word	YES	YES	NO
前進	character advance	NO	YES	NO
文字バックスペース	backspace valid	NO	YES	NO
画面クリア	clear	YES	YES	NO

表 19. SENDKEY コマンド・リスト (続き)

コマンド名	トークン	PC/3270	PC400	VT
入力音 (クリッカー)	click	YES	YES	NO
カラー選択 青	blue	YES	NO	NO
カラー選択フィールド継承	field color	YES	NO	NO
カラー選択 緑	green	YES	NO	NO
カラー選択 ピンク	pink	YES	NO	NO
カラー選択 赤	red	YES	NO	NO
カラー選択 青緑	turquoise	YES	NO	NO
カラー選択 白	white	YES	NO	NO
カラー選択 黄色	yellow	YES	NO	NO
カーソル明滅	cursor blink	YES	YES	NO
カーソル下移動	down	YES	YES	YES
カーソル左移動	left	YES	YES	YES
カーソル右移動	right	YES	YES	YES
カーソル選択	cursor select	YES	YES	NO
カーソル上移動	up	YES	YES	YES
削除	delete char	YES	YES	NO
ワード削除	delete word	YES	YES	NO
装置取り消し (DvCnl)	device cancel	YES	YES	NO
複写 (DUP)	dup	YES	YES	NO
編集 消去	edit-clear	YES	YES	YES
編集 コピー	edit-copy	YES	YES	YES
編集 切り抜き	edit-cut	YES	YES	YES
編集 貼り付け	edit-paste	YES	YES	YES
編集 取り消し	edit-undo	YES	YES	YES
フィールドの終わり	end field	YES	YES	NO
実行 (Enter)	enter	YES	YES	NO
EOF 消去	erase eof	YES	YES	NO
フィールド消去	erase field	YES	NO	NO
入力消去	erase input	YES	YES	NO
高速カーソル下移動	fast down	YES	YES	NO
高速カーソル左移動	fast left	YES	YES	NO
高速カーソル右移動	fast right	YES	YES	NO
高速カーソル上移動	fast up	YES	YES	NO
フィールド終了	field exit	NO	YES	NO
フィールド・マーク	field mark	YES	YES	NO
フィールド +	field +	NO	YES	NO
フィールド -	field -	NO	YES	NO
グラフィック・カーソル	+cr	YES	NO	NO
ヘルプ	help	YES	YES	NO

表 19. SENDKEY コマンド・リスト (続き)

コマンド名	トークン	PC/3270	PC400	VT
拡張強調表示 フィールドの継承	field hilight	YES	NO	NO
拡張強調表示 反転	reverse	YES	NO	NO
拡張強調表示 下線	underscore	YES	NO	NO
カーソル・ホーム	home	YES	YES	NO
ホスト印刷	host print	YES	NO	NO
入力	input	YES	YES	NO
入力 (非表示)	input nd	YES	YES	NO
挿入切り替え	insert	YES	YES	NO
小文字化	to lower	YES	NO	NO
マーク 水平線下移動	mark down	YES	YES	YES
マーク 垂直線左移動	mark left	YES	YES	YES
マーク 垂直線右移動	mark right	YES	YES	YES
マーク 水平線上移動	mark up	YES	YES	YES
マーク 下移動	move down	YES	YES	YES
マーク 左移動	move left	YES	YES	YES
マーク 右移動	move right	YES	YES	YES
マーク 上移動	move up	YES	YES	YES
改行	newline	YES	YES	YES
次ページ	page down	NO	YES	NO
1 秒休止	pause	YES	YES	NO
前ページ	page up	NO	YES	NO
画面印刷	local copy	YES	YES	YES
プログラム・アテンション (PA) キー 1	pa1	YES	NO	NO
プログラム・アテンション (PA) キー 2	pa2	YES	NO	NO
プログラム・アテンション (PA) キー 3	pa3	YES	NO	NO
プログラム・ファンクション・キー 1 . . . プログラム・ファンクション・キー 5	pf1 . . . pf5	YES . . . YES	YES . . . YES	NO . . . NO
プログラム・ファンクション・キー 6 . . . プログラム・ファンクション・キー 20	pf6 . . . pf20	YES . . . YES	YES . . . YES	YES . . . YES

表 19. SENDKEY コマンド・リスト (続き)

コマンド名	トークン	PC/3270	PC400	VT
プログラム・ファンクション・キー 21 . . . プログラム・ファンクション・キー 24	pf21 . . . pf24	YES . . . YES	YES . . . YES	No . . . No
中止	quit	YES	YES	NO
リセット	reset	YES	YES	NO
応答時間モニター	rtm	YES	NO	NO
ロールダウン	roll down	NO	YES	NO
ロールアップ	roll up	NO	YES	NO
後退	rubout	YES	YES	YES
ルーラー	rule	YES	YES	YES
SO/SI 表示	so si	YES	YES	NO
SO/SI 生成	so si generate	NO	YES	NO
システム要求	sys req	YES	YES	NO
タブ	tab field	YES	YES	YES
タブ・ワード	tab word	YES	YES	NO
テスト	test request	NO	YES	NO
マーク解除	unmark	YES	YES	YES
大文字化	to upper	YES	NO	NO
大文字/小文字の切り替え	to other	YES	NO	NO
バインド待ち	wait app	YES	YES	NO
システムとの接続待ち	wait sys	YES	YES	NO
状態変化待ち	wait trn	YES	YES	NO
入力可能待ち	wait inp inh	YES	YES	NO
ウィンドウの再配置 1 . . . ウィンドウの再配置 8	view 1 . . . view 8	Yes . . . X	Yes . . . X	Yes . . . X
VT 構成	vt compose	NO	NO	YES
VT 検索	vt find	NO	NO	YES
VT 保留画面	vt hold	NO	NO	YES
VT ここに挿入	vt insert	NO	NO	YES
VT 次画面	vt next	NO	NO	YES
VT テン・キー 0	vt numpad 0	NO	NO	YES
VT テン・キー 1	vt numpad 1	NO	NO	YES
VT テン・キー 2	vt numpad 2	NO	NO	YES
VT テン・キー 3	vt numpad 3	NO	NO	YES
VT テン・キー 4	vt numpad 4	NO	NO	YES
VT テン・キー 5	vt numpad 5	NO	NO	YES
VT テン・キー 6	vt numpad 6	NO	NO	YES

表 19. SENDKEY コマンド・リスト (続き)

コマンド名	トークン	PC/3270	PC400	VT
VT テン・キー 7	vt numpad 7	NO	NO	YES
VT テン・キー 8	vt numpad 8	NO	NO	YES
VT テン・キー 9	vt numpad 9	NO	NO	YES
VT テン・キー - (負記号)	vt numpad minus	NO	NO	YES
VT テン・キー , (コンマ)	vt numpad comma	NO	NO	YES
VT テン・キー . (ピリオド)	vt numpad period	NO	NO	YES
VT テン・キー Enter	vt numpad enter	NO	NO	YES
VT PF1	vt pf1	NO	NO	YES
VT PF2	vt pf2	NO	NO	YES
VT PF3	vt pf3	NO	NO	YES
VT PF4	vt pf4	NO	NO	YES
VT 前画面	vt prev	NO	NO	YES
VT 除去	vt remove	NO	NO	YES
VT 選択	vt select	NO	NO	YES
VT ユーザー定義関数 6	vt user f6	NO	NO	YES
VT ユーザー定義関数 7	vt user f7	NO	NO	YES
VT ユーザー定義関数 8	vt user f8	NO	NO	YES
VT ユーザー定義関数 9	vt user f9	NO	NO	YES
VT ユーザー定義関数 10	vt user f10	NO	NO	YES
VT ユーザー定義関数 11	vt user f11	NO	NO	YES
VT ユーザー定義関数 12	vt user f12	NO	NO	YES
VT ユーザー定義関数 13	vt user f13	NO	NO	YES
VT ユーザー定義関数 14	vt user f14	NO	NO	YES
VT ユーザー定義関数 15	vt user f15	NO	NO	YES
VT ユーザー定義関数 16	vt user f16	NO	NO	YES
VT ユーザー定義関数 17	vt user f17	NO	NO	YES
VT ユーザー定義関数 18	vt user f18	NO	NO	YES
VT ユーザー定義関数 19	vt user f19	NO	NO	YES
VT ユーザー定義関数 20	vt user f20	NO	NO	YES

例:

1. ログオンするには次のように指定します。

   ```
   [SENDKEY("Logon")]
   ```

2. 読み取り装置リストを得るには次のように指定します。

   ```
   [SENDKEY("RDRL", enter)]
   ```

WAIT コマンド

```
[WAIT("[time out][wait condition]")]
```

タイムアウトが満了となるか、またはクライアントが指定した待ち条件が発生するまで待ちます。このコマンドの場合、クライアントは最低 1 つのオプションを設定しなければなりません。

time out (任意指定)
クライアントがコマンド・ステートメント内にタイムアウト値を設定する場合、 WAIT ステートメント内では次の単位が使用可能です。

- msec
- millisecond
- milliseconds
- sec
- second
- seconds
- minute
- minutes
- hour
- hours

wait condition (任意指定)
待ち条件オプションについては、クライアントは次のものを選択できます。

while cursor at (cursor row, cursor column)
カーソルが、(カーソル行、カーソル列) の位置にある間、待ち続けます。

while "string"
“string” が、画面上のいずれかの場所にある間、待ち続けます。

while "string" at (cursor row, cursor column)
“string” が、画面上の (カーソル行、カーソル列) の位置にある間、待ち続けます。

until cursor at (cursor row, cursor column)
カーソルが、(カーソル行、カーソル列) へ移動するまで、待ち続けます。

until "string"
“string” が、画面上のいずれかの場所に表示されるまで、待ち続けます。

until "string" at (cursor row, cursor column)
“string” が、(カーソル行、カーソル列) に表示されるまで、待ち続けます。

例:

1. 10 秒待つには、次のように指定します。

```
[WAIT("10 seconds")]
```

2. 画面の (2,9) に "ABCDEF" が表示されている間、待つには次のように指定します。

   ```
   [WAIT("while ""ABCDEF"" at (2,9)")]
   ```

3. 画面の (2,9) に "ABCDEF" が表示されるまで、または 8 秒後に表示されるように待つためには、次のように指定します。

   ```
   [WAIT("8 seconds until ""ABCDEF"" at (2,9)")]
   ```

Set Cursor Position

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

Set Cursor Position 関数を使用すると、クライアント・アプリケーションでセッション・ウィンドウ内のカーソル位置を設定できます。

```
PostMessage( hServerWnd,
             WM_DDE_POKE,
             hClientWnd,
             PackDDELParam(WW_DDE_POKE,
             hData, aSETCURSOR) );
```

ここで、

hData

次のような構造体でカーソル位置決め情報が入っている Windows グローバル・メモリー・オブジェクトに対するハンドルを識別します。

```
typedef struct tagSETCURSOR
{
  unsigned char  poke[(sizeof(DDEPOKE)-1)];
  unsigned short uSetCursorType; /* Cursor Set Type
  unsigned short uSetCursor1;    /* Cursor Row or PS Offset
  unsigned short uSetCursor2;    /* Cursor Col
} SETCURSOR;

typedef union tagDDE_SETCURSOR
{
  DDEPOKE    DDEpoke;
  SETCURSOR  DDEsetcursor;
} DDE_SETCURSOR, *lpDDE_SETCURSOR;
```

パーソナル・コミュニケーションズは、カーソル位置を設定する次の 2 つの方法をサポートします。

- PS オフセット (uSetCursorType = 0)
- 行/桁の番号 (uSetCursorType = 1)

アプリケーションでどちらかの方法を指定するには、まず uSetCursorType フィールドを該当する値に設定し、その後、次のように他の 2 つのフィールド uSetCursor1 と uSetCursor2 を該当する値に設定します。

- uSetCursorType = 0 オフセット
  - uSetCursor1: 0 ... (PSsize - 1)
- uSetCursorType = 1 行/列
  - uSetCursor1: 0 ... (PSrows - 1)

– uSetCursor2: 0 ... (PScols - 1)

aSETCURSOR
カーソル位置をアイテムとして識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズはカーソル情報を受け取り、PS 内の指定された位置へカーソルを移動させます。カーソルの位置設定が成功した場合、パーソナル・コミュニケーションズはクライアント・アプリケーションへ肯定の ACK メッセージを戻します。それ以外の場合は、wStatus ワードの下位バイトに次のエラー・コードのうち 1 つが入っている否定の ACK メッセージが戻されます。

```
WM_DDE_ACK(wStatus, aSETCURSOR)
```

戻りコード	説明
1	カーソル設定タイプが無効です。 0 か 1 でなければなりません。
2	カーソル PS オフセットが無効です。 0 ... (PSsize - 1) でなければなりません。
3	カーソル行の値が無効です。 0 ... (PSrows - 1) でなければなりません。
4	カーソル列の位置が無効です。 0 ... (PScols - 1) でなければなりません。
6	無効な形式が指定されました。
9	システム・エラーが発生しました。

Set Mouse Intercept Condition

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

この関数は、代行受信するマウス入力を指定します。クライアントは、代行受信するマウス・イベントを設定するために次のコマンドを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_POKE,
             hClientWnd,
             PackDDElParam(WM_DDE_POKE,
             hData, aMOUSE) );
```

ここで、

hData マウス入力の代行受信の条件を指定している Windows グローバル・メモリー・オブジェクトへのハンドルを識別します。

形式が CF_TEXT の場合は、クライアント・プログラムは次の構造体でこの条件を送信します。

```
typedef struct tagSETMOUSE_CF_TEXT
{
  unsigned char poke[(sizeof(DDEPOKE)-1)];
  unsigned char zMouseCondition[1];
} SETMOUSE_CF_TEXT;
```

```
typedef union tagDDE_SETMOUSE_CF_TEXT
{
  DDEPOKE           DDEpoke;
  SETMOUSE_CF_TEXT  DDEcond;
} DDE_SETMOUSE_CF_TEXT, *lpDDE_SETMOUSE_CF_TEXT;
```

次の表はパラメーターの値を示しています。

パラメーター名	意味	値
条件	マウス代行受信の条件	'\0' で終わるストリングは、次のように定義した定数で構成されます (任意の順序)。 **L** 左ボタンの代行受信が可能 **l** 左ボタンの代行受信が不可 **R** 右ボタンの代行受信が可能 **r** 右ボタンの代行受信が不可 **M** 中央ボタンの代行受信が可能 **m** 中央ボタンの代行受信が不可 **S** シングルクリックの代行受信が可能 **s** シングルクリックの代行受信が不可 **D** ダブルクリックの代行受信が可能 **d** ダブルクリックの代行受信が不可 **T** 指示されたストリングを取り出す **t** 指示されたストリングを取り出さない

形式が CF_DSPTEXT の場合は、クライアント・プログラムは次の構造体でこの条件を送信します。

```
typedef struct tagSETMOUSE_CF_DSPTEXT
{
  unsigned char poke[(sizeof(DDEPOKE)-1)];
  BOOL          bLeftButton;     /* Enable intercepting left button
  BOOL          bRightButton;    /* Enable intercepting right button
  BOOL          bMiddleButton;   /* Enable intercepting middle button
  BOOL          bSingleClick;    /* Enable intercepting single click
  BOOL          bDoubleClick;    /* Enable intercepting double click
  BOOL          bRetrieveString; /* Enable intercepting retrieve string
} SETMOUSE_CF_DSPTEXT;

typedef union tagDDE_SETMOUSE_CF_DSPTEXT
{
```

```
  DDEPOKE              DDEpoke;
  SETMOUSE_CF_DSPTEXT  DDEcond;
} DDE_SETMOUSE_CF_DSPTEXT, *lpDDE_SETMOUSE_CF_DSPTEXT;
```

次の表はパラメーターの値を示しています。

パラメーター名	意味	値	
bLeftButton	マウスの左ボタンの代行受信を行うかどうか	**True**	左ボタンの代行受信が可能
		False	左ボタンの代行受信が不可
bRightButton	マウスの右ボタンの代行受信を行うかどうか	**True**	右ボタンの代行受信が可能
		False	右ボタンの代行受信が不可
bMiddleButton	マウスの中央ボタンの代行受信を行うかどうか	**True**	中央ボタンの代行受信が可能
		False	中央ボタンの代行受信が不可
bSingleClick	シングルクリックの代行受信を行うかどうか	**True**	シングルクリックの代行受信が可能
		False	シングルクリックの代行受信が不可
bDoubleClick	ダブルクリックの代行受信を行うかどうか	**True**	ダブルクリックの代行受信が可能
		False	ダブルクリックの代行受信が不可
bRetrieveString	指示されているストリングを取り出すかどうか	**True**	指示されたストリングを取り出す
		False	指示されたストリングを取り出さない

aMOUSE
マウスをアイテムとして識別します。

パーソナル・コミュニケーションズの応答

Set Mouse Intercept Condition 要求を受信するときに、パーソナル・コミュニケーションズが指定された状況に代行受信条件を設定できる場合、ACK メッセージを戻します。それ以外の場合、wStatus フィールドの下位バイトに次の戻りコードの 1 つが入っている否定の ACK メッセージを、クライアントに戻します。

```
WM_DDE_ACK(wStatus, aMOUSE)
```

戻りコード	説明
2	Condition パラメーター内の 1 文字が無効です。
6	無効な形式が指定されました。
9	システム・エラーが発生しました。

Set Presentation Space Service Condition

3270	5250	VT
Yes	Yes	Yes

Set Presentation Space Service Condition 関数は、次の関数を使用するための条件を設定します。

- **Get Partial Presentation Space**
- **Find Field**
- **Search for String**

クライアント・アプリケーションは関数を呼び出すことによって次のような条件を設定します。

- **PS 開始位置**
- **PS の長さ**
- **EOF フラグ**
- **検索方向**
- **ASCIIZ ストリング**

クライアントは、上記の関数を呼び出す前に、**Set Presentation Space Service Condition** 関数を指定しなければなりません。この関数によって設定した条件は、次に **Set Presentation Space Service Condition** 関数を呼び出すまでその効力を持ち続けます。クライアントは、条件を設定するために次のメッセージを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_POKE,
             hClientWnd,
             PackDDELParam(WM_DDE_POKE,
             (hData, aEPSCOND) );
```

ここで、

hData

次の内容を含む Windows グローバル・メモリー・オブジェクトへのハンドルを識別します。

```
typedef struct tagPSSERVCOND
{
  unsigned char  poke[(sizeof(DDEPOKE)-1)];
  unsigned short uPSStart;                    /* PS Position
  unsigned short uPSLength;                   /* Length of String or PS
  unsigned short uSearchDir;                  /* Direction for search
  unsigned short uEOFflag;                    /* EOF effective switch
  unsigned char  szTargetString[1];           /* Target String
} PSSERVCOND;

typedef union tagDDE_PSSERVCOND
{
  DDEPOKE     DDEpoke;
  PSSERVCOND  DDEcond;
} DDE_PSSERVCOND, *lpDDE_PSSERVCOND;
```

uSearchDir フィールドには次の値が有効です。

```
PCS_SRCHFRWD    0       /* Search forward.
PCS_SRCHBKWD    1       /* Search backward.
```

uEOFflag フィールドには次の値が有効です。

```
PCS_UNEFFECTEOF 0   /* The PS Area is not truncated at End of Field (EOF).
PCS_EFFECTEOF   1   /* The PS Area is truncated at End of Field (EOF).
```

uEOFflag の値が PCS_EFFECTEOF の場合は、 PS 長と検索方向の指定は使用されません。

aEPSCOND
Set Presentation Space Service Condition 関数用のアイテムを識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズが **Set Presentation Space Service Condition** 関数を実行できる場合、パーソナル・コミュニケーションズは次に ACK メッセージを戻します。

```
WM_DDE_ACK(wStatus, aEPSCOND)
```

パーソナル・コミュニケーションズが Set Presentation Space Service Condition 関数を実行できない場合、パーソナル・コミュニケーションズは wStatus の下位バイトに次の戻りコードが入っている否定の ACK メッセージを戻します。

戻りコード	説明
1	PS 位置が無効です。
2	長さが無効です。
3	EOF フラグの値が無効です。
4	検索方向の値が無効です。
6	無効な形式が指定されました。
9	システム・エラーが発生しました。

Set Session Advise Condition

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

この関数は、**Start Session Advise** 関数の DDE_ADVISE のための条件を設定します。クライアントは検索ストリングと画面の領域を指定できます。アドバイス条件が満たされると、サーバーは **Start Session Advise** 関数によって指定されたオプションに従ってその条件をクライアントへ通知します。

注: クライアントは **Start Session Advise** 関数を呼び出す前に、**Set Session Advise Condition** 関数を指定しなければなりません。 **Start Session Advise** 関数を始動した後にアドバイス条件を設定した場合、そのアドバイス条件は無視され、クライアントは否定の ACK メッセージを受け取ります。アドバイスの始動の詳細については、 296 ページの『Start Session Advise』を参照してください。

クライアントは、アドバイス条件を設定するために次のメッセージを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_POKE,
             hClientWnd,
             PackDDElParam(WM_DDE_POKE,
             (hData, aPSCOND) );
```

ここで、

hData 次の内容を含む Windows グローバル・メモリー・オブジェクトへのハンドルを識別します。

```
typedef struct tagSEARCHDATA
{
  unsigned char  poke[(sizeof(DDEPOKE)-1)];
  unsigned short uPSStart;                  /* PS Position of string
  unsigned short uPSLength;                 /* Length of String
  BOOL           bCaseSensitive;            /* Case Sensitive TRUE=YES
  unsigned char  SearchString[1]; /* Search String
} SEARCHDATA;

typedef union tagDDE_SEARCHDATA
{
  DDEPOKE     DDEpoke;
  SEARCHDATA  DDEcond;
} DDE_SEARCHDATA, *lpDDE_SEARCHDATA;
```

aPSCOND

Set Session Advise Condition 関数用のアイテムを識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズが **Set Session Advise Condition** 関数を実行できる場合、パーソナル・コミュニケーションズは次の ACK メッセージを戻します。

```
WM_DDE_ACK(wStatus, aPSCOND)
```

パーソナル・コミュニケーションズが **Set Session Advise Condition** 関数を実行できない場合、パーソナル・コミュニケーションズは wStatus の下位バイトに次の戻りコードのうち 1 つが入っている否定の ACK メッセージを戻します。

戻りコード	説明
1	アドバイスはすでに活動状態です。
2	アドバイス条件はすでに活動状態です。
3	PS 位置が無効です。
4	ストリングの長さが無効です。
6	無効な形式が指定されました。
9	システム・エラーが発生しました。

Set Structured Field Service Condition

3270	*5250*	*VT*
Yes	*No*	*No*

Set Structured Field Service Condition 関数は、クライアント・アプリケーションが提供する Query Reply データを渡します。

注: クライアントは **Start Read SF** 関数または **Write SF** 関数を呼び出す前に、**Set Structured Field Service Condition** 関数を呼び出さなければなりません。

クライアントは、条件を設定するために次のメッセージを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_POKE,
             hClientWnd,
             PackDDElParam(WM_DDE_POKE,
             (hData, aSFCOND) );
```

ここで、

hData 次の内容を含む Windows グローバル・メモリー・オブジェクトへのハンドルを識別します。

```
typedef struct tagSFSERVCOND
{
  unsigned char  poke[(sizeof(DDEPOKE)-1)];
  unsigned short uBufferLength;             /* Buffer size of Read_SF
  unsigned short uQRLength;                 /* Length of Query Reply dat
  unsigned char  szQueryReply[1];            /* Query Reply data
} SFSERVCOND;

typedef union tagDDE_SFSERVCOND
{
  DDEPOKE     DDEpoke;
  SFSERVCOND  DDEcond;
} DDE_SFSERVCOND, *lpDDE_SFSERVCOND;
```

aSFCOND

Set Structured Field Service Condition 関数用のアイテムを識別します。

PC/3270 の応答

PC/3270 は、Query Reply ID、Type (DOID ではない) およびその長さを検査します。これらが有効であれば、PC/3270 は次に ACK メッセージを戻します。

```
WM_DDE_ACK(wStatus, aSFCOND)
```

PC/3270 が **Set Structured Field Service Condition** 関数を実行できない場合、PC/3270 は wStatus の下位バイトに次の戻りコードのうち 1 つが入っている否定の ACK メッセージを戻します。

戻りコード	説明
1	PS SF ID が無効です。
2	長さが無効です。
3	このセッションにはすでに 1 つの DDM 基本タイプが接続されています。
4	Structured Field Service Condition がすでに設定されています。
6	無効な形式が指定されました。
9	システム・エラーが発生しました。

Start Close Intercept

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

Start Close Intercept 関数を使用すると、ユーザーがエミュレーター・セッション・ウィンドウからクローズ・オプションを選択したときに生成されるクローズ要求を、クライアント・アプリケーションによって代行受信できます。この関数は **Stop Close Intercept** 関数が要求されるまで、クローズ要求を代行受信してそれを破棄します。この関数を使用した後、クライアントはクローズ要求が発生したこと (CLOSE) を通知する DATA メッセージを受け取ります。

クライアントはセッション・アドバイスを開始するために次のコマンドを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_ADVISE,
             hClientWnd,
             Pack DDElParam(WM_DDE_ADVISE,
             (hOptions, aCLOSE) );
```

ここで、

hOptions

DDEADVISE 構造体をもつ Windows グローバル・メモリー・オブジェクトへのハンドルです。

fDeferUpd の値が 1 である場合、NULL に設定された hData とともに DDE データ・メッセージがクライアント・アプリケーションへ送信されます。次に、クライアントはデータ・アイテムを要求するために DDE REQUEST を発行しなければなりません。

fAckReq の値が 1 である場合、サーバーはそれ以降のクローズ要求をクライアントへ通知しません。この状態は、サーバーが前の通知に対する応答で、クライアントから ACK メッセージを受け取るまで続きます。

cfFormat フィールドはクローズ要求を送信する形式を指定します。(CF_DSPTEXT 形式でなければなりません。)

aCLOSE

クローズ・インターセプトをアイテムとして識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズは、代行受信を開始できる場合には **Start Close Intercept** を受け取って ACK メッセージを戻します。それ以外の場合、wStatus フィールドの下位バイトに次の戻りコードの 1 つが入っている否定の ACK メッセージを、クライアントに戻します。

```
WM_DDE_ACK(wStatus, aCLOSE)
```

戻りコード	説明
1	Close Intercept はすでに始動しています。
6	無効な形式が指定されました。

戻りコード	説明
9	システム・エラーが発生しました。

代行受信が始動されると、クライアントはクローズ要求が代行受信されたことを知らせる次の DATA メッセージを受け取ります。

```
WM_DDE_DATA(hData, aCLOSE)
```

ここで、

hData 次の内容を含む Windows グローバル・メモリー・オブジェクトへのハンドルを識別します。

```
typedef struct tagCLOSEREQ
{
  unsigned char  data[(sizeof(DDEDATA)-1)];
  unsigned short uCloseReqCount; /* Number of the close requests.
} CLOSEREQ;

typedef union tagDDE_CLOSEREQ
{
  DDEDATA   DDEdata;
  CLOSEREQ  DDEclose;
} DDE_CLOSEREQ, *lpDDE_CLOSEREQ;
```

この DATA メッセージは、Stop Close Intercept メッセージがパーソナル・コミュニケーションズへ送信されるまで続きます。

Start Keystroke Intercept

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

Start Keystroke Intercept 関数を使用すると、端末オペレーターによってセッションへ送信された任意のキー・ストロークをクライアント・アプリケーションでフィルターにかけることができます。この関数を呼び出した後では、キー・ストロークは代行受信され、クライアントがそれ (KEYS) を受け取ります。

クライアントは代行受信を開始するために次のコマンドを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_ADVISE,
             hClientWnd,
             PackDDElParam(WM_DDE_ADVISE,
             (hOptions, aKEYS) );
```

ここで、

hOptions

DDEADVISE 構造体をもつ Windows グローバル・メモリー・オブジェクトへのハンドルです。

fDeferUpd の値が 1 である場合、NULL に設定された hData とともに DDE データ・メッセージがクライアント・アプリケーションへ送信されます。次にクライアントはデータ・アイテムを要求するために DDE REQUEST を発行しなければなりません。

fAckReq の値が 1 である場合、サーバーはそれ以降のキー・ストロークをクライアントへ通知しません。この状態は、サーバーが前のキー・ストロークの通知に対する応答でクライアントから ACK メッセージを受け取るまで続きます。

cfFormat フィールドは、端末オペレーターによってキー・ストロークが送信されるときにキー・ストロークを送信する形式を指定します。
(CF_DSPTEXT 形式でなければなりません。)

aKEYS
キー・ストロークをアイテムとして識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズは、代行受信を開始できる場合には **Start Keystroke Intercept** を受け取って ACK メッセージを戻します。それ以外の場合、wStatus フィールドの下位バイトに次の戻りコードの 1 つが入っている否定の ACK メッセージを、クライアントに戻します。

```
WM_DDE_ACK(wStatus, aKEYS)
```

戻りコード	説明
1	Keystroke Intercept はすでに始動しています。
6	無効な形式が指定されました。
9	システム・エラーが発生しました。

代行受信が始動されると、クライアントはキー・ストロークが代行受信されたことを知らせる次の DATA メッセージを受け取ります。

```
WM_DDE_DATA(hData, aKEYS)
```

この DATA メッセージは、**Stop Keystroke Intercept** メッセージがパーソナル・コミュニケーションズへ送信されるまで続きます。データ・アイテムの形式は、クライアントが DDE_REQUEST を経由してそのデータ・アイテムを要求した場合と同じ形式になります。

Start Mouse Input Intercept

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

Start Mouse Input Intercept 関数を使用すると、端末オペレーターがエミュレーター・セッション・ウィンドウ上でマウス・ボタンを押したとき、クライアント・アプリケーションがマウス入力を代行受信できます。この関数を呼び出した後、クライアントは、マウス入力が発生した PS 位置を含む DATA メッセージを受信します。

クライアントはマウス入力の代行受信を開始するために、次のコマンドを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_ADVISE,
             hClientWnd,
             PackDDElParam(WM_DDE_ADVISE,
             (hOptions, aMOUSE) );
```

ここで、

hOptions

DDEADVISE 構造体をもつ Windows グローバル・メモリー・オブジェクトへのハンドルです。

fDeferUpd の値が 1 である場合、NULL に設定された hData とともに DDE データ・メッセージがクライアント・アプリケーションへ送信されます。次に、クライアントはデータ・アイテムを要求するために DDE REQUEST を発行しなければなりません。

fAckReq の値が 1 である場合、サーバーはそれ以降の構造化フィールド・データをクライアントへ通知しません。この状態は、サーバーが前の通知に対する応答で、クライアントから ACK メッセージを受け取るまで続きます。

cfFormat フィールドは、更新されたデータ・アイテムを送信するための形式を指定します。

aMOUSE

マウスをアイテムとして識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズは、代行受信を開始できる場合には **Start Mouse Input Intercept** を受け取って ACK メッセージを戻します。それ以外の場合、wStatus フィールドの下位バイトに次の戻りコードの 1 つが入っている否定の ACK メッセージを、クライアントに戻します。

```
WM_DDE_ACK(wStatus, aMOUSE)
```

戻りコード	説明
1	Mouse Input Intercept はすでに始動しています。
6	無効な形式が指定されました。
9	システム・エラーが発生しました。

Mouse Input Intercept が始動すると、クライアントは構造化フィールドの DATA メッセージを受信します。

```
WM_DDE_DATA(hData, aMOUSE)
```

ここで、

hData

形式が CF_TEXT の場合は、パーソナル・コミュニケーションズは以下の形式でマウス入力情報を戻します。

```
typedef struct tagMOUSE_CF_TEXT
{
  unsigned char data[(sizeof(DDEDATA)-1)];
```

```
  unsigned char PSPos[4];       /* PS Offset - Mouse position
  unsigned char Tab1[1];        /* Tab character
  unsigned char PSRowPos[4];    /* ROW number of Mouse position
  unsigned char Tab2[1];        /* Tab character
  unsigned char PSColPos[4];    /* Col number of Mouse position
  unsigned char Tab3[1];        /* Tab character
  unsigned char PSSize[4];      /* Size of Presentation Space
  unsigned char Tab4[1];        /* Tab character
  unsigned char PSRows[4[;      /* Row number of PS
  unsigned char Tab5[1];        /* Tab character
  unsigned char PSCols[4];      /* Column number of PS
  unsigned char Tab6[1];        /* Tab character
  unsigned char Button[1];      /* Type of clicked mouse butt n
  unsigned char Tab7[1];        /* Tab character
  unsigned char Click[1];       /* Type of clicking
  unsigned char Tab8[1];        /* Tab character
  unsigned char zClickString[1];/* Retrieved string
} MOUSE_CF_TEXT;

typedef union tagDDE_MOUSE_CF_TEXT
{
  DDEDATA        DDEdata;
  MOUSE_CF_TEXT  DDEmouse;
} DDE_MOUSE_CF_TEXT, *lpDDE_MOUSE_CF_TEXT;
```

次の表はパラメーターの値を示しています。

パラメーター名	意味	値
PSPos	マウスがクリックされた位置の PS オフセット	0 ... (PSSize - 1)
PSRowPos	マウスがクリックされた位置の行番号	0 ... (PSRows - 1)
PSColPos	マウスがクリックされた位置の桁番号	0 ... (PSCols - 1)
PSSize	表示スペースのサイズ	
PSRows	表示スペースの行数	
PSCols	表示スペースの桁数	
ButtonType	クリックされたマウス・ボタンのタイプ	**L** 左ボタン **M** 中央ボタン **R** 右ボタン
ClickType	クリックのタイプ	**S** シングルクリック **D** ダブルクリック
ClickString	マウスが指示した取り出しストリング	'\0' で終了する文字ストリング
Tab1～8	区切り文字としてのタブ文字	'\t'

形式が CF_DSPTEXT の場合は、パーソナル・コミュニケーションズは以下の形式でマウス入力情報を戻します。

```
typedef struct tagMOUSE_CF_DSPTEXT
{
  unsigned char  data[(sizeof(DDEDATA)-1)];
  unsigned short uPSPos;                  /* PS Offset  - Mouse position
  unsigned short uPSRowPos;               /* ROW number - Mouse position
```

```
  unsigned short uPSColPos;              /* Col number - Mouse position
  unsigned short uPSSize;                /* Size of Presentation Space
  unsigned short uPSRows;                /* Row number of PS
  unsigned short uPSCols;                /* Column number of PS
  unsigned short uButtonType;            /* Type of clicked mouse button
  unsigned short uClickType;             /* Type of clicking
  unsigned char  zClickString[1];         /* Retrieved string
} MOUSE_CF_DSPTEXT;

typedef union tagDDE_MOUSE_CF_DSPTEXT
{
  DDEDATA           DDEdata;
  MOUSE_CF_DSPTEXT  DDEmouse;
} DDE_MOUSE_CF_DSPTEXT, *lpDDE_MOUSE_CF_DSPTEXT;
```

次の表はパラメーターの値を示しています。

パラメーター名	意味	値
uPSPos	マウスがクリックされた位置の PS オフセット	0 ... (uPSSize - 1)
uPSRowPos	マウスがクリックされた位置の行番号	0 ... (uPSRows - 1)
uPSColPos	マウスがクリックされた位置の桁番号	0 ... (uPSCols - 1)
uPSSize	表示スペースのサイズ	
uPSRows	表示スペースの行数	
uPSCols	表示スペースの桁数	
uButtonType	クリックされたマウス・ボタンのタイプ	**0x0001** 左ボタン **0x0002** 中央ボタン **0x0003** 右ボタン
uClickType	クリックのタイプ	**0x0001** シングルクリック **0x0002** ダブルクリック
szClickString	マウスが指示した取り出しストリング	'\0' で終了する文字ストリング

この DATA メッセージは、**Stop Mouse Input Intercept** メッセージがパーソナル・コミュニケーションズへ送信されるまで続きます。

Start Read SF

3270	*5250*	*VT*
Yes	*No*	*No*

Start Read SF 関数を使用すると、ホスト・アプリケーションからの構造化フィールド・データをクライアント・アプリケーションによって読み取れます。この関数を使用した後、クライアントはクローズ要求が発生したことを通知する DATA メッセージを受け取ります。

注: この関数を使用する前に、クライアントは Query Reply データをサーバーへ渡すために **Set Structured Field Service Condition** 関数を呼び出さなければなりません。

クライアントは Read SF を開始するために次のコマンドを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_ADVISE,
             hClientWnd,
             PackDDElParam(WM_DDE_ADVISE,
             (hOptions, aSF) );
```

ここで、

hOptions

DDEADVISE 構造体をもつ Windows グローバル・メモリー・オブジェクトへのハンドルです。

fDeferUpd の値が 1 である場合、NULL に設定された hData とともに DDE データ・メッセージがクライアント・アプリケーションへ送信されます。次に、クライアントはデータ・アイテムを要求するために DDE REQUEST を発行しなければなりません。

fAckReq の値が 1 である場合、サーバーはそれ以降の構造化フィールド・データをクライアントへ通知しません。この状態は、サーバーが前の通知に対する応答で、クライアントから ACK メッセージを受け取るまで続きます。

cfFormat フィールドは、構造化フィールド・データを送信するための形式を指定します。(CF_DSPTEXT 形式でなければなりません。)

aSF 構造化フィールドをアイテムとして識別します。

PC/3270 の応答

PC/3270 は、**Start Read SF** を始動できる場合には、Start Read SF を受け取って ACK メッセージを渡します。それ以外の場合、wStatus フィールドの下位バイトに次の戻りコードの 1 つが入っている否定の ACK メッセージを、クライアントに戻します。

```
WM_DDE_ACK(wStatus, aSF)
```

戻りコード	説明
1	Read SF はすでに始動しています。
3	前に **Set Structured Field Service Condition** 関数が呼び出されていません。
6	無効な形式が指定されました。
9	システム・エラーが発生しました。

Read SF を始動すると、クライアントは次の構造化フィールドの DATA メッセージを受け取ります。

```
WM_DDE_DATA(hData, aSF)
```

ここで、

hData 次の内容を含む Windows グローバル・メモリー・オブジェクトへのハンドルを識別します。

```
typedef struct tagMOUSE_CF_DSPTEXT
{
  unsigned char  data[(sizeof(DDEDATA)-1)];
  unsigned short uPSPos;                  /* PS Offset  - Mouse position
  unsigned short uPSRowPos;               /* ROW number - Mouse position
  unsigned short uPSColPos;               /* Col number - Mouse position
  unsigned short uPSSize;                 /* Size of Presentation Space
  unsigned short uPSRows;                 /* Row number of PS
  unsigned short uPSCols;                 /* Column number of PS
  unsigned short uButtonType;             /* Type of clicked mouse button
  unsigned short uClickType;              /* Type of clicking
  unsigned char  zClickString[1];          /* Retrieved string
} MOUSE_CF_DSPTEXT;

typedef union tagDDE_MOUSE_CF_DSPTEXT
{
  DDEDATA           DDEdata;
  MOUSE_CF_DSPTEXT  DDEmouse;
} DDE_MOUSE_CF_DSPTEXT, *lpDDE_MOUSE_CF_DSPTEXT;
typedef struct tagSFDATA
{
  unsigned char  data[(sizeof(DDEDATA)-1)];
  unsigned short uSFLength;                /* Length of SF data
  unsigned char  szSFData[1];               /* SF data
} SFDATA;

typedef union tagDDE_SFDATA
{
  DDEDATA  DDEdata;
  SFDATA   DDEsfdata;
} DDE_SFDATA, *lpDDE_SFDATA;
```

この DATA メッセージは、Stop Read SF メッセージが PC/3270 へ送信されるまで続きます。

Start Session Advise

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

Start Session Advise 関数は、パーソナル・コミュニケーションズのセッションとクライアントの間にリンクを確立します。これにより、クライアントはデータ・アイテムが更新されるときに表示スペース (PS)、オペレーター情報域 (OIA)、またはトリミング長方形 (TRIMRECT) の更新データを受け取ります。

注: 表示スペースの更新時、クライアント・アプリケーションに条件付き通知が必要な場合は、表示スペース用のアドバイス関数を呼び出す前にアドバイス条件を設定してください。 286 ページの『Set Session Advise Condition』を参照してください。

クライアントはセッション・アドバイスを開始するために次のコマンドを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_ADVISE,
             hClientWnd,
             PackDDElParam(WM_DDE_ADVISE,
             hOptions, aItem) );
```

ここで、

hOptions

DDEADVISE 構造体をもつ Windows グローバル・メモリー・オブジェクトへのハンドルです。構造体は次のとおりです。

```
typedef struct tagDDEADVISE
{
 unsigned reserved:14;        // Reserved
 unsigned fDeferUpd:1;        // Send notification only
 unsigned fAckReq:1;          // Client will ACK all notices
 WORD     cfFormat;           // Clipboard format to use
}  DDEADVISE, *lpDDEADVISE;
```

fDeferUpd の値が 1 である場合、NULL に設定された hData とともに DDE データ・メッセージがクライアント・アプリケーションへ送信されます。次に、クライアントはデータ・アイテムを要求するために DDE REQUEST を発行しなければなりません。

fAckReq の値が 1 である場合、サーバーはそれ以降のデータ・アイテムの変更をクライアントへ通知しません。この状態は、サーバーが前の更新通知に対する応答で、クライアントから ACK メッセージを受信するまで続きます。

アイテムが更新されると、cfFormat フィールドは、そのデータ・アイテムを送信するための形式を指定します。

aItem 要求されている情報のアイテムを指定します。この場合、値は PS、OIA、または TRIMRECT です。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズは、アドバイスを始動できる場合には、Start Session Advise を受け取って ACK メッセージを戻します。それ以外の場合、wStatus フィールドの下位バイトに次の戻りコードの 1 つが入っている否定の ACK メッセージを、クライアントに戻します。

```
WM_DDE_ACK(wStatus, aItem)
```

戻りコード	説明
1	アドバイスはデータ・アイテムに対してすでに活動状態です。
6	アドバイス・パラメーターが無効です。
9	システム・エラーが発生しました。

アドバイスが始動すると、クライアントはデータ・アイテム (PS、OIA、または TRIMRECT) が更新されたことを知らせる次の DATA メッセージを受け取ります。

```
WM_DDE_DATA(hData, aItem)
```

この DATA メッセージは、**Stop Session Advise** メッセージがパーソナル・コミュニケーションズへ送信されるまで続きます。データ・アイテムの形式は、クライアントが DDE_REQUEST を経由してそのデータ・アイテムを要求した場合と同じになります。

Stop Close Intercept

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

Stop Close Intercept 関数は、クローズ要求を代行受信するクライアント・アプリケーションの機能を終了させます。クライアントは、**Stop Close Intercept** 関数を実行するために次のコマンドを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_UNADVISE,
             hClientWnd,
             MAKELPARAM(NULL, aCLOSE) );
```

ここで、

aCLOSE
クローズ・インターセプトをアイテムとして識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズが DDE_UNADVISE を実行できる場合、パーソナル・コミュニケーションズは肯定の状況情報が入った、次の ACK メッセージをクライアントへ戻します。

```
WM_DDE_ACK(wStatus, aCLOSE)
```

パーソナル・コミュニケーションズが DDE_UNADVISE を実行できない場合、パーソナル・コミュニケーションズは、否定の状況情報、および wStatus ワードの下位バイトに次の戻りコードのうち 1 つが入っている ACK メッセージを戻します。

戻りコード	説明
1	アドバイスはまだ始動していません。
9	システム・エラーが発生しました。

Stop Keystroke Intercept

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

Stop Keystroke Intercept 関数は、キー・ストロークを代行受信するクライアント・アプリケーションの機能を終了させます。クライアントは、**Stop Keystroke Intercept** 関数を実行するために次のコマンドを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_UNADVISE,
             hClientWnd,
             MAKELPARAM(NULL, aKEYS) );
```

ここで、

aKEYS
キー・ストロークをアイテムとして識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズが DDE_UNADVISE を実行できる場合、パーソナル・コミュニケーションズは肯定の状況情報が入った、次の ACK メッセージをクライアントへ戻します。

```
WM_DDE_ACK(wStatus, aKEYS)
```

パーソナル・コミュニケーションズが DDE_UNADVISE を実行できない場合、パーソナル・コミュニケーションズは、否定の状況情報、および wStatus ワードの下位バイトに次の戻りコードのうち 1 つが入っている ACK メッセージを戻します。

戻りコード	説明
1	アドバイスはまだ始動していません。
9	システム・エラーが発生しました。

Stop Mouse Input Intercept

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

Stop Mouse Input Intercept 関数は、マウス入力を代行受信するクライアント・アプリケーションの機能を終了させます。

クライアントは、**Stop Mouse Input Intercept** 関数を実行するために次のコマンドを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_UNADVISE,
             hClientWnd,
             MAKELPARAM(NULL, aMOUSE) );
```

ここで、

aMOUSE
マウスをアイテムとして識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズが DDE_UNADVISE を実行できる場合、パーソナル・コミュニケーションズは肯定の状況情報が入った、次の ACK メッセージをクライアントへ戻します。

```
WM_DDE_ACK(wStatus, aMOUSE)
```

パーソナル・コミュニケーションズが DDE_UNADVISE を実行できない場合、パーソナル・コミュニケーションズは、否定の状況情報、および wStatus ワードの下位バイトに次の戻りコードのうち 1 つが入っている ACK メッセージを戻します。

戻りコード	説明
1	アドバイスはまだ始動していません。
9	システム・エラーが発生しました。

Stop Read SF

3270	*5250*	*VT*
Yes	*No*	*No*

Stop Read SF 関数は、構造化フィールド・データを読み取るクライアント・アプリケーションの機能を終了させます。

クライアントは、**Stop Read SF** 関数を実行するために次のコマンドを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_UNADVISE,
             hClientWnd,
             MAKELPARAM(NULL, aSF) );
```

ここで、

aSF　構造化フィールドをアイテムとして識別します。

PC/3270 の応答

PC/3270 が DDE_UNADVISE を実行できる場合、PC/3270 は肯定の状況情報が入った、次の ACK メッセージをクライアントへ戻します。

```
WM_DDE_ACK(wStatus, aSF)
```

PC/3270が DDE_UNADVISE を実行できない場合、PC/3270は、否定の状況情報、および wStatus ワードの下位バイトに次の戻りコードのうち 1 つが入っている ACK メッセージを戻します。

戻りコード	説明
1	アドバイスはまだ始動していません。
9	システム・エラーが発生しました。

Stop Session Advise

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

Stop Session Advise 関数は、パーソナル・コミュニケーションズとクライアントの間のリンクを切断します。クライアントは **Stop Session Advise** 関数を実行するために次のコマンドを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_UNADVISE,
             hClientWnd,
             MAKELPARAM(NULL, aItem) );
```

ここで、

aItem 要求されている情報のアイテムを指定します。この場合、値は PS、OIA、TRIMRECT、または NULL です。

aItem が NULL である場合、クライアントはその会話に対して活動状態にあるすべての通知の終了を要求しました。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズが DDE_UNADVISE を実行できる場合、パーソナル・コミュニケーションズは肯定の状況情報が入った、次の ACK メッセージをクライアントへ戻します。

```
WM_DDE_ACK(wStatus, aItem)
```

パーソナル・コミュニケーションズが DDE_UNADVISE を実行できない場合、パーソナル・コミュニケーションズは、否定の状況情報、および wStatus ワードの下位バイトに次の戻りコードのうち 1 つが入っている ACK メッセージを戻します。

戻りコード	説明
1	アドバイスはまだ始動していません。
9	システム・エラーが発生しました。

Terminate Session Conversation

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

Terminate Session Conversation 関数は、クライアントが事前に会話を始動しているパーソナル・コミュニケーションズのセッションからクライアントを切断します。

クライアントはセッション会話を終了させるために次のコマンドを送信します。

```
SendMessage( hServerWnd,
             WM_DDE_TERMINATE,
             hClientWnd,
             0 );
```

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズは、次の終了メッセージで終了コマンドを確認します。

```
WM_DDE_TERMINATE
```

Terminate Structured Field Conversation

3270	5250	VT
Yes	No	No

Terminate Structured Field Conversation 関数は、クライアントを構造化フィールド会話から切断します。

クライアントは構造化フィールド会話を終了させるために次のコマンドを送信します。

```
SendMessage( hServerWnd,
             WM_DDE_TERMINATE,
             hClientWnd,
             0 );
```

PC/3270 の応答

PC/3270 は、次の終了メッセージで終了コマンドを確認します。

```
WM_DDE_TERMINATE
```

Terminate System Conversation

3270	5250	VT
Yes	Yes	Yes

この関数は、クライアントをシステム会話から切断します。

クライアントはシステム会話を終了させるために次のコマンドを送信します。

```
SendMessage( hServerWnd,
             WM_DDE_TERMINATE,
             hClientWnd,
             0 );
```

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズは、次の終了メッセージで終了コマンドを確認します。

```
WM_DDE_TERMINATE
```

ユーザーがパーソナル・コミュニケーションズ・セッションをクローズすると、パーソナル・コミュニケーションズが割り振った任意のグローバル・メモリー・ブロックが Windows によって解放されます。これらのグローバル・メモリー・オブジェクトをクライアントが長時間保持している場合は、クライアントに問題が生じる恐れがあります。クライアント・アプリケーションがグローバル・メモリー・アイテム内に情報を長時間保持する必要がある場合は、グローバル・メモリー・アイテムを、クライアント・アプリケーションが所有するグローバル・メモリー・アイテム内にコピーすることをお勧めします。

Write SF

3270	5250	VT
Yes	No	No

Write SF 関数を使用すると、クライアント・アプリケーションが構造化フィールド・データをホスト・アプリケーションへ書き込むことができます。

注: クライアントは、**Write SF** 関数を呼び出す前に **Set Structured Field Service Condition** 関数を呼び出さなければなりません。

クライアントは、構造化フィールド・データを書き込むために次のメッセージを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_POKE,
             hClientWnd,
             PackDDELParam(WM_DDE_POKE,
             hData, aSF) );
```

ここで、

hData 次の内容を含む Windows グローバル・メモリー・オブジェクトへのハンドルを識別します。

```
typedef struct tagWRITESF
{
  unsigned char  poke[(sizeof(DDEPOKE)-1)];
  unsigned short uSFLength;      /* Length of SF data
  unsigned char  Work[8];        /* Work area
  unsigned char  szSFData[1];    /* SF data
} WRITESF;

typedef union tagDDE_WRITESF
{
  DDEPOKE  DDEpoke;
  WRITESF  DDEwritesf;
} DDE_WRITESF, *lpDDE_WRITESF;
```

aSF 構造化フィールドをアイテムとして識別します。

PC/3270 の応答

PC/3270 は構造化フィールド・データを受け取って、そのデータをホスト・アプリケーションへ送信します。データ伝送が正常終了した場合、PC/3270 は次の ACK メッセージを戻します。

```
WM_DDE_ACK(wStatus, aSF)
```

それ以外の場合、PC/3270 は wStatus の下位バイトに、次の戻りコードのうち 1 つが入った否定の ACK メッセージを戻します。

戻りコード	説明
2	長さが無効です。
6	無効な形式が指定されました。
9	システム・エラーが発生しました。

Windows 32 ビット環境での DDE メニュー・アイテム API

パーソナル・コミュニケーションズは、セッション・メニュー・バーに対する 動的メニュー・アイテムの属性の追加、削除、および変更を サポートしています。その場合、このメニュー・アイテムに対して最大 16 個までのサブメニュー・アイテム用のスペースを持つメニューが作成されます。

パーソナル・コミュニケーションズは 2 種類の DDE 会話をサポートしています。1 つは DDE メニュー・クライアント・アプリケーションとして機能するもの、もう 1 つは DDE メニュー・サーバーとして機能するものです。

DDE メニュー・クライアント

メニュー・アイテムを追加、削除、および変更するためには、セッションと DDE メニュー・サーバー・アプリケーションとの間で次の DDE 会話が行われなければなりません。

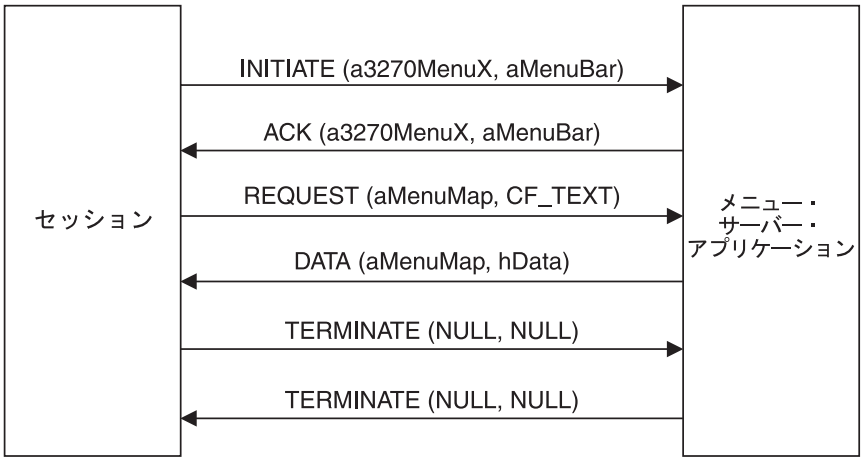

図 3. DDE メニュー・サーバーの会話

次のデータ階層は、セッション・メニュー・バーへ動的メニュー・アイテムとサブメニューを追加するときに、パーソナル・コミュニケーションズが予期するメニュー・マップを詳しく説明したものです。

```
POPUP  "MyMenu"
BEGIN
  MENUITEM "Send Files to Host",        SEND
  MENUITEM "Receive Files from Host",   RECEIVE
  MENUITEM SEPARATOR
  MENUITEM "Convert Files",             CONVERT
END
```

ユーザーが新しいメニューからメニュー・アイテムを選択すると、パーソナル・コミュニケーションズはアプリケーションとして、3270MenuN または 5250MenuN を持ち、トピックとして itemN トークンを持った DDE Initiate を送信します。 DDE アプリケーションから ACK を受け取った場合、パーソナル・コミュニケーションズはセッションがユーザー入力を受け入れるのを禁止します。その後、メニュー・クライアント・アプリケーションはダイアログなどを表示することができます。メニュー・サーバー・アプリケーションは、そのメニュー・アイテムの処理を完了した時点で DDE Terminate を送信してパーソナル・コミュニケーションズにプロセス

が完了したことを知らせます。その後、パーソナル・コミュニケーションズはユーザー用のウィンドウを再び使用可能にします。

DDE メニュー・サーバー

メニュー・アイテムを追加、削除、および変更するためには、セッションと DDE メニューのクライアント・アプリケーションとの間で次に示す DDE 会話が行われなければなりません。

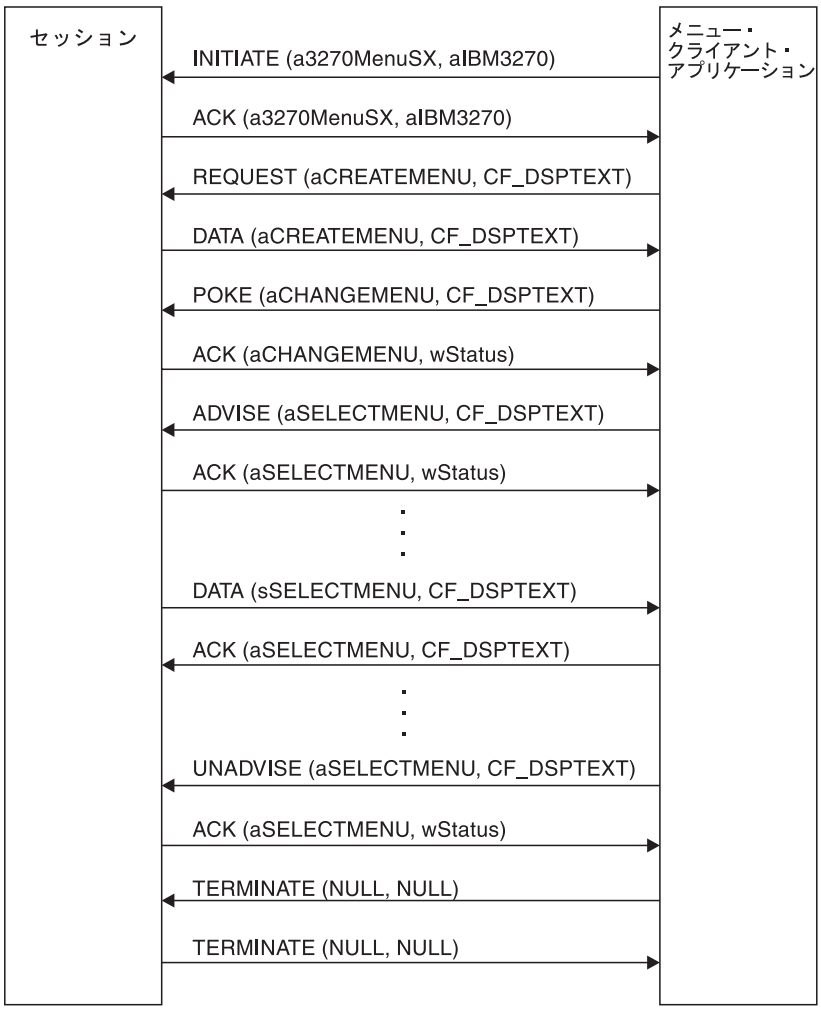

図 4. DDE メニュー・クライアントの会話

ユーザーが新しいメニューからメニュー・アイテムを選択すると、パーソナル・コミュニケーションズはアイテムとして aSELECTMENU を持った DDE DATA を送信します。パーソナル・コミュニケーションズが DDE DATA をクライアント・アプリケーションへ送信した時点で、パーソナル・コミュニケーションズはセッションがユーザー入力を受け入れることを禁止します。その後、メニュー・クライアント・アプリケーションはダイアログなどを表示することができます。メニュー・クライアント・アプリケーションは、そのメニュー・アイテムの処理を完了すると DDE ACK を送信し、パーソナル・コミュニケーションズにプロセスが完了したことを知らせます。その後、パーソナル・コミュニケーションズはユーザー用のウィンドウを再び使用可能にします。

DDE メニュー関数

パーソナル・コミュニケーションズで使用可能な DDE メニュー・アイテム API 関数を、以下のリストに示します。PC/3270 Windows モードおよび PC400 では、以下の関数をすべて提供しています。

- 『Change Menu Item』
- 312 ページの『Create Menu Item』
- 314 ページの『Initiate Menu Conversation』
- 314 ページの『Start Menu Advise』
- 316 ページの『Stop Menu Advise』
- 316 ページの『Terminate Menu Conversation』

Change Menu Item

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

Change Menu Item 関数には、メニュー・アイテムの追加、削除、挿入、変更、および除去を行います。クライアントは、メニューを変更するためにセッションに対して次のメッセージを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_POKE,
             hClientWnd,
             PackDDElParam(WM_DDE_POKE,
             hData,aCHANGEMENU));
```

ここで、

hData

メニューの変更要求を含む Windows グローバル・メモリー・オブジェクトへのハンドルを識別します。このグローバル・メモリー・オブジェクトには次の構造体が入っています。

```
typedef struct tagChangeMenu
{
  unsigned char  poke[(sizeof(DDEPOKE)-1)];
  HWND           hMenu;                    /* Window handle of menu item
  unsigned long  wIDNew;                   /* Menu ID of new menu item
  unsigned short wPosition;                /* The position of menu item
  unsigned short wOperation;               /* Specifies the operation
  unsigned short wFlags;                   /* Specifies the options
  unsigned char  szItemName[1];  /* String of the item
} CHANGEMENU;

typedef union tagDDE_CHANGEMENU
{
  DDEPOKE     DDEpoke;
  CHANGEMENU  DDEmenu;
} DDE_CHANGEMENU,*lpDDE_CHANGEMENU;
```

次の操作がサポートされています。

```
# MF_APPEND,MF_CHANGE ... MF_BYCOMMANDS are replaced with below commands.
 PCS_INSERT       0x0000   /* Inserts a menu item into a menu.
 PCS_CHANGE       0x0080   /* Modifies a menu item in a menu.
```

```
PCS_APPEND        0x0100   /* Appends a menu item to the end of a menu
PCS_DELETE        0x0200   /* Deletes a menu item from a menu,
                           /* destroying the menu item.
PCS_REMOVE        0x1000   /* Removes a menu item from a menu but
                           /* does not destroy the menu item.

PCS_CHECKED       0x0008   /* Places a check mark next to the item.
PCS_DISABLED      0x0002   /* Disables the menu item so that it cannot
                           /* be selected, but does not gray it.
PCS_ENABLED       0x0000   /* Enables the menu item so that it can be
                           /* selected and restores from its grayed
                           /* state.
PCS_GRAYED        0x0001   /* Disables the menu item so that it cannot
                           /* be selected, and grays it.
PCS_MENUBARBREAK  0x0020   /* Same as PCS_MENUBREAK except that for
                           /* popup menus, separates the new column
                           /* from the old column with a vertical line
PCS_MENUBREAK     0x0040   /* Places the item on a new line for menu
                           /* bar items. For popup menus, places the
                           /* item in a new column, with no dividing
                           /* line between the columns.
PCS_SEPARATOR     0x0800   /* Draws a horizontal dividing line. Can
                           /* only be used in a popup menu. This line
                           /* cannot be grayed, disabled, or
                           /* highlighted. The wIDNew and szItemName
                           /* fields are ignored.
PCS_UNCHAKED      0x0000   /* Does not place a check mark next to the
                           /* item (default).

PCS_BYCOMMAND     0x0000   /* Specifies that the nPosition parameter
                           /* gives the menu item control ID number.
                           /* This is the default if neither item
                           /* control ID number. This is the default
                           /* if neither PCS_BYCOMMAND nor
                           /* PCS_POSITION is set.
PCS_BYPOSITION    0x0400   /* Specifies that the nPosition parameter
                           /* gives the position of the menu item
                           /* to be deleted rather than an ID number.
```

wOperation フィールドに MF_APPEND を指定した場合は、次のフィールドも記入しなければなりません。

hMenu

追加変更したいメニューを識別します。ポップアップ・メニューへ新しいアイテムを追加するには、 **Create Menu Item** 関数を実行したときにパーソナル・コミュニケーションズから戻すハンドルを指定します。新しいメニュー・アイテムを最上位レベルのメニュー・バーへ追加するには NULL を指定します。

wIDNew

新しいメニュー・アイテムのコマンド ID を指定します。新しいメニュー・アイテムを最上位レベルのメニュー・バーへ追加する場合には、**Create Menu Item** 関数を実行したときにパーソナル・コミュニケーションズから戻すメニュー・アイテムのハンドルを指定します。

wFlags

次のオプションを設定できます。

```
MF_CHECKED       // Places a check mark next to
                 // the item.
MF_DISABLED      // Disables the menu item so
                 // that it cannot be selected,
                 // but does not gray it.
MF_ENABLED       // Enables the menu item so that
                 // it can be selected and
                 // restores from its grayed
                 // state.
MF_GRAYED        // Disables the menu item so
                 // that it cannot be selected,
                 // and grays it.
MF_MENUBARBREAK  // Same as MF_MENUBREAK except
                 // that for pop-up menus,
                 // separates the new column from
                 // the old column with a
                 // vertical line.
MF_MENUBREAK     // Places the item on a new line
                 // for menu bar items.
                 // For pop-up menus, places the
                 // item in a new column, with
                 // no dividing line between the
                 // columns.
MF_SEPARATOR     // Draws a horizontal dividing
                 // line.  Can only be used in a
                 // pop-up menu.  This line cannot
                 // be grayed, disabled, or
                 // highlighted.  The wIDNew and
                 // szItemName fields are
                 // ignored.
MF_UNCHECKED     // Does not place a check mark
                 // next to the item (default).
```

szItemName

新しいメニュー・アイテムの内容を指定します。NULL で終了する文字ストリングが入っています。

wOperation フィールドに MF_CHANGE を指定した場合は、次のフィールドにも記入しなければなりません。

hMenu

変更したいメニューを識別します。ポップアップ・メニューのアイテムを変更するには、 **Create Menu Item** 関数を実行したときにパーソナル・コミュニケーションズから戻すハンドルを指定します。アイテムを最上位レベルのメニュー・バーへ追加するには NULL を指定します。

nPosition

変更したいアイテム・メニューを指定します。 wPosition パラメーターの解釈は、wFlags パラメーターの設定によって異なります。

MF_BYPOSITION
既存のメニュー・アイテムの位置を指定します。メニュー内の最初のアイテムは位置ゼロです。

MF_BYCOMMAND
既存のメニュー・アイテムのコマンド ID を指定します。

wIDNew

メニュー・アイテムのコマンド ID を指定します。最上位レベルのメニュー・バーのアイテムを変更する場合には、**Create Menu Item** 関数を実行したときにパーソナル・コミュニケーションズから戻すメニュー・アイテムのハンドルを指定します。

wFlags

次のオプションを設定できます。

```
MF_BYCOMMAND     // Specifies that the nPosition
                 // parameter gives the menu
                 // item control ID number.
                 // This is the default if
                 // neither MF_BYCOMMAND nor
                 // MF_BYPOSITION is set.
MF_BYPOSITION    // Specifies that the nPosition
                 // parameter gives the position
                 // of the menu item to be
                 // changed rather than an ID
                 // number.
MF_CHECKED       // Places a check mark next to
                 // the item.
MF_DISABLED      // Disables the menu item so
                 // that it cannot be selected,
                 // but does not gray it.
MF_ENABLED       // Enables the menu item so
                 // that it can be selected and
                 // restores from its grayed
                 // state.
MF_GRAYED        // Disables the menu item so
                 // that it cannot be selected,
                 // and grays it.
MF_MENUBARBREAK  // Same as MF_MENUBREAK except
                 // that for pop-up menus,
                 // separates the new column
                 // from the old column with a
                 // vertical line.
MF_MENUBREAK     // Places the item on a new
                 // line for menu bar items.
                 // For pop-up menus, places the
                 // item in a new column, with
                 // no dividing line between
                 // the columns.
MF_SEPARATOR     // Draws a horizontal dividing
                 // line.  Can only be used in
                 // a pop-up menu.  This line
                 // cannot be grayed, disabled,
                 // or highlighted.  The wIDNew
                 // and szItemName fields are
                 // ignored.
MF_UNCHECKED     // Does not place a check mark
                 // next to the item (default).
```

szItemName

メニュー・アイテムの内容を指定します。NULL で終了する文字ストリングが入っています。

wOperation フィールドに MF_DELETE を指定した場合は、次のフィールドにも記入しなければなりません。

hMenu

削除したいメニューを識別します。アイテムをポップアップ・メニューから削除するには、 **Create Menu Item** 関数を実行したときに

パーソナル・コミュニケーションズから戻すハンドルを指定します。最上位レベルのメニュー・バーからアイテムを削除するには NULL を指定します。

nPosition

削除したいアイテム・メニューを指定します。nPosition パラメーターの解釈は、wFlags パラメーターの設定によって異なります。

MF_BYPOSITION
既存のメニュー・アイテムの位置を指定します。メニュー内の最初のアイテムは位置ゼロです。

MF_BYCOMMAND
既存のメニュー・アイテムのコマンド ID を指定します。

wFlags

次のオプションを設定できます。

```
MF_BYCOMMAND     // Specifies that the nPosition
                 // parameter gives the menu
                 // item control ID number.
                 // This is the default if
                 // neither MF_BYCOMMAND nor
                 // MF_BYPOSITION is set.
MF_BYPOSITION    // Specifies that the nPosition
                 // parameter gives the position
                 // of the menu item to be
                 // deleted rather than an ID
                 // number.
```

wOperation フィールドに MF_INSERT を指定した場合は、次のフィールドも記入しなければなりません。

hMenu

挿入したいメニューを識別します。アイテムをポップアップ・メニューへ挿入するには、 **Create Menu Item** 関数を実行したときにパーソナル・コミュニケーションズから戻すハンドルを指定します。アイテムを最上位レベルのメニュー・バーへ追加するには NULL を指定します。

nPosition

新しいメニュー・アイテムを挿入する前にメニュー・アイテムを指定します。 nPosition パラメーターの解釈は、wFlags パラメーターの設定によって異なります。

MF_BYPOSITION
既存のメニュー・アイテムの位置を指定します。メニュー内の最初のアイテムは位置ゼロです。

MF_BYCOMMAND
既存のメニュー・アイテムのコマンド ID を指定します。

wIDNew

メニュー・アイテムのコマンド ID を指定します。あるいは、最上位レベルのメニュー・バーのアイテムを変更する場合には、**Create**

Menu Item 関数を実行したときにパーソナル・コミュニケーションズから戻すメニュー・アイテムのハンドルを指定します。

wFlags

次のオプションを設定できます。

```
MF_BYCOMMAND         // Specifies that the nPosition
                     // parameter gives the menu
                     // item control ID number. This
                     // is the default if neither
                     // MF_BYCOMMAND nor MF_BYPOSITION
                     // is set.
MF_BYPOSITION         // Specifies that the nPosition
                     // parameter gives the position
                     // of the menu item to be
                     // changed rather than an ID
                     // number.
MF_CHECKED            // Places a check mark next to
                     // the item.
MF_DISABLED           // Disables the menu item so
                     // that it cannot be selected,
                     // but does not gray it.
MF_ENABLED            // Enables the menu item so
                     // that it can be selected and
                     // restores from its grayed
                     // state.
MF_GRAYED             // Disables the menu item so
                     // that it cannot be selected,
                     // and grays it.
MF_MENUBARBREAK       // Same as MF_MENUBREAK except
                     // that for pop-up menus,
                     // separates the new column
                     // from the old column with a
                     // vertical line.
MF_MENUBREAK          // Places the item on a new
                     // line for menu bar items.
                     // For pop-up menus, places the
                     // item in a new column, with
                     // no dividing line between the
                     // columns.
MF_SEPARATOR          // Draws a horizontal dividing
                     // line.  Can only be used in
                     // a pop-up menu.  This line
                     // cannot be grayed, disabled,
                     // or highlighted. The wIDNew
                     // and szItemName fields are
                     // ignored.
MF_UNCHECKED          // Does not place a check mark
                     // next to the item (default).
```

szItemName

メニュー・アイテムの内容を指定します。NULL で終了する文字ストリングが入っています。

wOperation フィールドに MF_REMOVE を指定した場合は、次のフィールドも記入しなければなりません。

hMenu

除去したいメニューを識別します。ポップアップ・メニューのアイテムから除去するには、 **Create Menu Item** 関数を実行したときに

パーソナル・コミュニケーションズから戻すハンドルを指定します。最上位レベルのメニュー・バーからアイテムを除去するには NULL を指定します。

nPosition

除去したいアイテム・メニューを指定します。nPosition パラメーターの解釈は、wFlags パラメーターの設定によって異なります。

MF_BYPOSITION
既存のメニュー・アイテムの位置を指定します。メニュー内の最初のアイテムは位置ゼロです。

MF_BYCOMMAND
既存のメニュー・アイテムのコマンド ID を指定します。

wFlags

次のオプションを設定できます。

```
MF_BYCOMMAND         // Specifies that the nPosition
                    // parameter gives the menu
                    // item control ID number.
                    // This is the default if
                    // neither MF_BYCOMMAND nor
                    // MF_BYPOSITION is set.
MF_BYPOSITION        // Specifies that the nPosition
                    // parameter gives the
                    // position of the menu item to
                    // be removed rather than an ID
                    // number.
```

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズはメニューの変更要求を受け取って処理します。要求を受け入れられない場合、パーソナル・コミュニケーションズは wStatus ワードの下位バイトに、次の状況コードのうち 1 つが入った否定の ACK メッセージを戻します。それ以外の場合、パーソナル・コミュニケーションズはキー・ストロークが送信されたことを知らせる 肯定の ACK メッセージを戻します。

```
WM_DDE_ACK(wStatus,aCHANGEMENU)
```

戻りコード	説明
1	無効なパラメーターが指定されました。
6	無効な形式が指定されました。
9	システム・エラーが発生しました。

Create Menu Item

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

Create Menu Item 関数は、メニュー・アイテムをメニュー・バーへ追加するようパーソナル・コミュニケーションズに要求します。それと同時にポップアップ・メニューが作成されますが、ポップアップ・メニューは最初は空なので、Create Menu

Item 関数を使用してメニュー・アイテムに記入します。また、最上位レベルのメニュー・バーへ追加される新しいメニュー・アイテムのストリングも **Change Menu Item** 関数を使用して指定します。

クライアントはメニュー・アイテムを作成するために次のメッセージを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
             MAKELPARAM(cfFormat,aCREATEMENU));
```

ここで、

cfFormat

新しいメニュー・アイテムの ID の形式を識別します。有効値は CF_DSPTEXT です。

aCREATEMENU

作成メニュー・アイテムを識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズは、メニュー・アイテムを作成できる場合には、新しく作成したメニュー・アイテムのハンドルを DDE データ・メッセージ内に戻します。

```
WM_DDE_DATA(hData,aCREATEMENU)
```

ここで、

hData

メニュー・アイテムのハンドルを含む Windows グローバル・メモリー・オブジェクトへのハンドルを識別します。このグローバル・メモリー・オブジェクトには次の構造体が入っています。

```
typedef struct tagCreateMenu
{
  unsigned char    data[(sizeof(DDEDATA)-1)];
  HWND             hMemuItem;     /* Handle of the menu item
} CREATEMENU;

typedef union tagDDE_CREATEMENU
{
  DDEDATA     DDEdata;
  CREATEMENU  DDEmenu;
} DDE_CREATEMENU,*lpDDE_CREATEMENU;
```

または

```
WM_DDE_ACK(wStatus,aCREATEMENU)
```

パーソナル・コミュニケーションズがメニュー・アイテムを作成できない場合は、次の状況コードのうち 1 つが wStatus ワードの下位バイト内に戻されます。

戻りコード	説明
6	無効な形式が指定されました。
9	システム・エラーが発生しました。

Initiate Menu Conversation

3270	5250	VT
Yes	Yes	Yes

Initiate Menu Conversation 関数は、クライアント・アプリケーションを、使用可能なパーソナル・コミュニケーションズのセッションへ接続します。メニュー会話が確立されると、会話が終了するまで、そのセッションのメニューはそのクライアント専用のものとして予約されます。

クライアント・アプリケーションは、メニューとの DDE 会話を開始するために次のメッセージを送信します。

```
SendMessage( -1,
             WM_DDE_INITIATE,
             hClientWnd,
             MAKELPARAM(aIBM327032,SN));
```

ここで、

aIBM327032

アプリケーション・アトムを識別します。アトム aIBM327032 を作成するために使用するストリングは IBM327032 です。PC400 の場合は、アプリケーション・アトムは aIBM525032 で、これを作成するために使用するストリングは IBM525032 です。

SN

トピック・アトムを識別します。アトム a3270MenuSN を作成するために使用するストリングは、セッション ID A、B、...、Z のいずれかを付けた 3270MenuS です。 PC400 の場合、トピック・アトム a5250MenuSN を作成するために使用するストリングは、セッション ID A、B、...、Z のいずれかを付けた 5250MenuS です。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズがクライアント・アプリケーションとの会話をサポートできる場合、パーソナル・コミュニケーションズは次のもので INITIATE トランザクションを確認します。

```
WM_DDE_ACK(aIBM327032,SN)
```

Start Menu Advise

3270	5250	VT
Yes	Yes	Yes

Start Menu Advise 関数を使用すると、クライアント・アプリケーションで追加したメニュー・アイテムが選択されたときに、クライアント・アプリケーションはユーザー定義ルーチンを処理できます。この関数を使用した後、クライアントは選択されたメニュー・アイテムを知らせる DATA メッセージを受け取ります。

クライアントはメニュー・アドバイスを開始するために次のコマンドを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_ADVISE,
             hClientWnd,
             PackDDElParam(WM_DDE_ADVISE,
             hOptions,aSELECTMENU));
```

ここで、

hOptions

次の構造を持つ Windows グローバル・メモリー・オブジェクトへのハンドルです。

```
typedef struct tagOPTIONS
{
 unsigned reserved:14;       // Reserved
 unsigned fDeferUpd:1;       // Send notification only
                             // (Must be 0)
 unsigned fAckReq:1;         // Client will ACK all notices
                             // (Must be 1)
 WORD     cfFormat;          // Always CF_DSPTEXT
}  OPTIONS,FAR *lpOPTIONS;
```

aSELECTMENU

メニュー・アドバイスをアイテムとして識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズは、**Start Menu Advise** 関数を始動できる場合は、Start Menu Advise を受け取って ACK メッセージを戻します。それ以外の場合、wStatus フィールドの下位バイトに次の戻りコードの 1 つが入っている否定の ACK メッセージを、クライアントに戻します。

戻りコード	説明
1	メニュー・アドバイスはすでに始動しています。
6	無効な形式が指定されました。
9	システム・エラーが発生しました。

```
WM_DDE_ACK(wStatus,aSELECTMENU)
```

クライアント・アプリケーションに追加したメニュー・アイテムが選択されると、クライアントは選択されたメニュー・アイテムを知らせる次の DATA メッセージを受け取ります。

```
WM_DDE_DATA(hData,aSELECTMENU)
```

ここで、

hData

次の内容を含む Windows グローバル・メモリー・オブジェクトへのハンドルを識別します。

```
typedef struct tagSELECTMENU
{
  unsigned char  data[(sizeof(DDEDATA)-1)];
  unsigned short uIDSelected; /* Command ID of the selected menu item
} SELECTMENU;
```

```
typedef union tagDDE_SELECTMENU
{
  DDEDATA     DDEdata;
  SELECTMENU  DDEmenu;
} DDE_SELECTMENU,*lpDDE_SELECTMENU;
```

この DATA メッセージは、Stop Menu Advise メッセージがパーソナル・コミュニケーションズへ送信されるまで続きます。

Stop Menu Advise

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

Stop Menu Advise 関数は、クライアント・アプリケーションで追加したメニュー・アイテムが選択されたときにユーザー定義ルーチンを処理するクライアント・アプリケーションの機能を終了させます。クライアントは **Stop Menu Advise** 関数を実行するために次のコマンドを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_UNADVISE,
             hClientWnd,
             MAKELPARAM(NULL,aSELECTMENU));
```

ここで、

aSELECTMENU

メニュー・アドバイスをアイテムとして識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズが DDE_UNADVISE を実行できる場合、パーソナル・コミュニケーションズは肯定の状況情報が入った、次の ACK メッセージをクライアントへ戻します。

```
WM_DDE_ACK(wStatus,aCLOSE)
```

パーソナル・コミュニケーションズが DDE_UNADVISE を実行できない場合、パーソナル・コミュニケーションズは、否定の状況情報、および wStatus ワードの下位バイトに次の戻りコードのうち 1 つが入っている ACK メッセージを戻します。

戻りコード	説明
1	アドバイスはまだ始動していません。
9	システム・エラーが発生しました。

Terminate Menu Conversation

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

Terminate Menu Conversation 関数は、クライアントが事前に会話を始動しているパーソナル・コミュニケーションズのセッションからそのクライアントを切断します。

クライアントはセッション会話を終了させるために次のコマンドを送信します。

```
SendMessage( hServerWnd,
             WM_DDE_TERMINATE,
             hClientWnd,
             0 );
```

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズは、次の終了メッセージで終了コマンドを確認します。

```
WM_DDE_TERMINATE
```

Windows 32 ビット環境での DDE 関数の要約

次の表は、PC/3270 または PC400 で使用できる DDE 関数を示したものです。この表では、DDE 関数の名前、クライアントから PC/3270 または PC400 へ送信するコマンド、およびクライアントのコマンド内で変数に使用できる値を示しています。

表 20. DDE 関数の要約

関数名	クライアントのコマンド	サーバーの応答
Code Conversion (システム)	`PostMessage(hServerWnd WM_DDE_POKE, hClientWnd, PackDDEIParam(WM_DDE_POKE,hData, aCONV));`	`UnPackDDElParam(WM_DDE_ACK,wStatus,aCONV)`
Initiate System Conversation (システム)	`SendMessage(-1, WM_DDE_INITIATE, hClientWnd, MAKELPARAM(aIBM327032,aSystem));`	`LOWORD/HIWORD to unpack WM_DDE_ACK(aIBM327032,aSystem)`

Get System Configuration (システム)	`PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aSYSCON));`
	`cfFormat = CF_TEXT`
		`UnPackDDElParam(WM_DDE_DATA,hData,aSYSCON)`
		`or`
		`UnPackDDElParam(WM_DDE_ACK,wStatus,aSYSCON)`

Get System Formats (システム)	`PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aFORMATS));`
	`cfFormat = CF_TEXT`
		`UnPackDDElParam(WM_DDE_DATA,hData,aFORMATS)`
		`or`
		`UnPackDDElParam(WM_DDE_ACK,wStatus,aFORMATS)`

Get System Status (システム)	`PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aSTATUS));`
	`cfFormat = CF_TEXT`
		`UnPackDDElParam(WM_DDE_DATA,hData,aSTATUS)`
		`or`
		`UnPackDDElParam(WM_DDE_ACK,wStatus,aSTATUS)`

Get System SysItems (システム)	`PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aSYSITEMS));`
	`cfFormat = CF_TEXT`
		`UnPackDDElParam(WM_DDE_DATA,hData,aSYSITEMS)`
		`or`
		`UnPackDDElParam(WM_DDE_ACK,wStatus,aSYSITEMS)`

表 20. DDE 関数の要約 (続き)

関数名	クライアントのコマンド	サーバーの応答
Get System Topics (システム)	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aTOPICS)); cfFormat = CF_TEXT	UnPackDDElParam(WM_DDE_DATA,hData,aTOPICS) or UnPackDDElParam(WM_DDE_ACK,wStatus,aTOPICS)
Terminate System Conversation (システム)	SendMessage(hServerWnd, WM_DDE_TERMINATE, hClientWnd, 0);	WM_DDE_TERMINATE
Initiate Session Conversation (セッション)	SendMessage(-1, WM_DDE_INITIATE, hClientWnd, MAKELPARAM(aIBM327032,aSessionN)); N = a session letter A through Z.	LOWORD/HIWORD to unpack WM_DDE_ACK(aIBM327032,aSessionN)
Find Field (セッション)	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aFIELD)); cfFormat = CF_DSPTEXT	UnPackDDElParam(WM_DDE_DATA,hData,aFIELD) or UnPackDDElParam(WM_DDE_ACK,wStatus,aFIELD)
Get Keystrokes (セッション)	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aKEYS)); cfFormat = CF_DSPTEXT	UnPackDDElParam(WM_DDE_DATA,hData,aKEYS) or UnPackDDElParam(WM_DDE_ACK,wStatus,aKEYS)
Get Mouse Input (セッション)	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aMOUSE)); cfFormat = CF_TEXT | CF_DSPTEXT	UnPackDDElParam(WM_DDE_DATA,hData,aMOUSE) or UnPackDDElParam(WM_DDE_ACK,wStatus,aMOUSE)
Get Number of Close Requests (セッション)	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aCLOSE)); cfFormat = CF_DSPTEXT	UnPackDDElParam(WM_DDE_DATA,hData,aCLOSE) or UnPackDDElParam(WM_DDE_ACK,wStatus,aCLOSE)
Get Operator Information Area (セッション)	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aOIA)); cfFormat = CF_DSPTEXT	UnPackDDElParam(WM_DDE_DATA,hData,aOIA) or UnPackDDElParam(WM_DDE_ACK,wStatus,aOIA)
Get Partial Presentation Space (セッション)	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aEPS)); cfFormat = CF_TEXT | CF_DSPTEXT	UnPackDDElParam(WM_DDE_DATA,hData,aEPS) or UnPackDDElParam(WM_DDE_ACK,wStatus,aEPS)
Get Presentation Space (セッション)	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aPS)); cfFormat = CF_TEXT | CF_DSPTEXT	UnPackDDElParam(WM_DDE_DATA,hData,aPS) or UnPackDDElParam(WM_DDE_ACK,wStatus,aPS)

表 20. DDE 関数の要約 (続き)

関数名	クライアントのコマンド	サーバーの応答
Get Session Status (セッション)	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aSSTAT)); cfFormat = CF_TEXT	UnPackDDElParam(WM_DDE_DATA,hData,aSSTAT) or UnPackDDElParam(WM_DDE_ACK,wStatus,aSSTAT)
Get Trim Rectangle (セッション)	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aTRIMRECT)); cfFormat = CF_TEXT	UnPackDDElParam(WM_DDE_DATA,hData,aTRIMRECT) or UnPackDDElParam(WM_DDE_ACK,wStatus,aTRIMRECT)
Put Data to Presentation Space (セッション)	PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, PackDDElParam(WM_DDE_POKE, hData,aEPS)); hData = Handle to a global memory object	UnPackDDElParam(WM_DDE_ACK,wStatus,aEPS)
Search for String (セッション)	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aSTRING)); cfFormat = CF_DSPTEXT	UnPackDDElParam(WM_DDE_DATA,hData,aSTRING) or UnPackDDElParam(WM_DDE_ACK,wStatus,aSTRING)
Send Keystrokes (セッション)	PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, PackDDElParam(WM_DDE_POKE, hData,aKEYS)); hData = Handle to a global memory object	UnPackDDElParam(WM_DDE_ACK,wStatus,aKEYS)
Session Execute Macro (セッション)	PostMessage(hServerWnd, WM_DDE_EXECUTE, hClientWnd, (LPARAM)hCommands); hCommands = Handle to a global memory object	UnPackDDElParam(WM_DDE_ACK,wStatus, NULL)
Set Cursor Position (セッション)	PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, PackDDElParam(WM_DDE_POKE, hData,aSETCURSOR)); hData = Handle to a global memory object	UnPackDDElParam(WM_DDE_ACK,wStatus,aSETCURSOR)
Set Mouse Intercept Condition (セッション)	PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, PackDDElParam(WM_DDE_POKE, hData,aMOUSE)); cfFormat = CF_TEXT | CF_DSPTEXT hData = Handle to a global memory object	UnPackDDElParam(WM_DDE_ACK,wStatus,aMOUSE)

表 20. DDE 関数の要約 (続き)

関数名	クライアントのコマンド	サーバーの応答
Set Presentation Space Service Condition (セッション)	PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, PackDDElParam(WM_DDE_POKE, hData,aEPSCOND)); hData = Handle to a global memory object	UnPackDDElParam(WM_DDE_ACK,wStatus,aEPSCOND)
Set Session Advise Condition (セッション)	PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, PackDDElParam(WM_DDE_POKE, hData,aPSCOND)); hData = Handle to a global memory object	UnPackDDElParam(WM_DDE_ACK,wStatus,aPSCOND)
Start Close Intercept (セッション)	SendMessage(hServerWnd, WM_DDE_ADVISE, hClientWnd, PackDDElParam(WM_DDE_ADVISE, hOptions,aCLOSE)); hOptions = Handle to a global memory object	UnPackDDElParam(WM_DDE_ACK,wStatus,aCLOSE) or UnPackDDElParam(WM_DDE_DATA,hData,aCLOSE)
Start Keystroke Intercept (セッション)	SendMessage(hServerWnd, WM_DDE_ADVISE, hClientWnd, PackDDElParam(WM_DDE_ADVISE, hOptions,aKEYS)); hOptions = Handle to a global memory object	UnPackDDElParam(WM_DDE_ACK,wStatus,aKEYS) or UnPackDDElParam(WM_DDE_DATA,hData,aKEYS)
Start Mouse Input Intercept (セッション)	SendMessage(hServerWnd, WM_DDE_ADVISE, hClientWnd, PackDDElParam(WM_DDE_ADVISE, hOptions,aMOUSE)); hOptions = Handle to a global memory object	UnPackDDElParam(WM_DDE_ACK,wStatus,aMOUSE) or UnPackDDElParam(WM_DDE_DATA,hData,aMOUSE)
Start Session Advise (セッション)	PostMessage(hServerWnd, WM_DDE_ADVISE, hClientWnd, PackDDElParam(WM_DDE_ADVISE, hOptions,aItem)); hOptions = Handle to a global memory object aItem = OIA | PS | TRIMRECT	UnPackDDElParam(WM_DDE_ACK,wStatus,aItem) or UnPackDDElParam(WM_DDE_DATA,hData,aItem)
Stop Close Intercept (セッション)	PostMessage(hServerWnd, WM_DDE_UNADVISE, hClientWnd, MAKELPARAM(NULL,aCLOSE));	UnPackDDElParam(WM_DDE_ACK,wStatus,aCLOSE)
Stop Keystroke Intercept (セッション)	PostMessage(hServerWnd, WM_DDE_UNADVISE, hClientWnd, MAKELPARAM(NULL,aKEYS));	UnPackDDElParam(WM_DDE_ACK,wStatus,aKEYS)

表 20. DDE 関数の要約 (続き)

関数名	クライアントのコマンド	サーバーの応答
Stop Mouse Input Intercept (セッション)	PostMessage(hServerWnd, WM_DDE_UNADVISE, hClientWnd, MAKELPARAM(NULL,aMOUSE));	UnPackDDElParam(WM_DDE_ACK,wStatus,aMOUSE)
Stop Session Advise (セッション)	PostMessage(hServerWnd, WM_DDE_UNADVISE, hClientWnd, MAKELPARAM(NULL,aItem)); aItem = SysItems | Topics | NULL	UnPackDDElParam(WM_DDE_ACK,wStatus,aItem)
Terminate Session Conversation (セッション)	SendMessage(hServerWnd, WM_DDE_TERMINATE, hClientWnd, 0);	WM_DDE_TERMINATE
Initiate Structured Field Conversation (構造化フィールド)	SendMessage(-1, WM_DDE_INITIATE, hClientWnd, MAKELPARAM(aIBM327032,aLUN_xxxx)); N = a session letter A through Z. xxxx = a user defined string.	LOWORD/HIWORD to unpack WM_DDE_ACK(aIBM327032,aLUN_xxxx)
Terminate Structured Field Conversation (構造化フィールド)	SendMessage(hServerWnd, WM_DDE_TERMINATE, hClientWnd, 0);	WM_DDE_TERMINATE
Set Structured Field Service Condition (構造化フィールド)	PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, PackDDElParam(WM_DDE_POKE, hData,aSFCOND)); hData = Handle to a global memory object	UnPackDDElParam(WM_DDE_ACK,wStatus,aSFCOND)
Start Read SF (構造化フィールド)	PostMessage(hServerWnd, WM_DDE_ADVISE, hClientWnd, PackDDElParam(WM_DDE_ADVISE, hOptions,aSF)); hOptions = Handle to a global memory object	UnPackDDElParam(WM_DDE_ACK,wStatus,aSF) or UnPackDDElParam(WM_DDE_DATA,hData,aSF)
Stop Read SF (構造化フィールド)	PostMessage(hServerWnd, WM_DDE_UNADVISE, hClientWnd, MAKELPARAM(NULL,aSF));	UnPackDDElParam(WM_DDE_ACK,wStatus,aSF)
Write SF (構造化フィールド)	PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, PackDDElParam(WM_DDE_POKE, hData,aSF)); hData = Handle to a global memory object	UnPackDDElParam(WM_DDE_ACK,wStatus,aSF)

表 20. DDE 関数の要約 (続き)

関数名	クライアントのコマンド	サーバーの応答
Initiate Menu Conversation (メニュー)	SendMessage(-1, WM_DDE_INITIATE, hClientWnd, MAKELPARAM(aIBM327032,a3270MenuSN)); N = a session letter A through Z	LOWORD/HIWORD to unpack WM_DDE_ACK(aIBM327032,a3270MenuSN)
Change Menu Item (メニュー)	PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, PackDDElParam(WM_DDE_POKE, hData,aCHANGEMENU)); hData = Handle to a global memory object	UnPackDDElParam(WM_DDE_ACK,wStatus,aCHANGEMENU)
Create Menu Item (メニュー)	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aCREATEMENU)); cfFormat = CF_DSPTEXT	UnPackDDElParam(WM_DDE_DATA,hData,aCREATEMENU) or UnPackDDElParam(WM_DDE_ACK,wStatus,aCREATEMENU)
Start Menu Advise (メニュー)	PostMessage(hServerWnd, WM_DDE_ADVISE, hClientWnd, PackDDElParam(WM_DDE_ADVISE, hOption,aSELECTMENU)); hData = Handle to a global memory object	UnPackDDElParam(WM_DDE_ACK,wStatus,aSELECTMENU) or UnPackDDElParam(WM_DDE_DATA,hData,aSELECTMENU)
Stop Menu Advise (メニュー)	PostMessage(hServerWnd, WM_DDE_UNADVISE, hClientWnd, MAKELPARAM(NULL,aSELECTMENU));	UnPackDDElParam(WM_DDE_ACK,wStatus,aCLOSE)
Terminate Menu Conversation (メニュー)	SendMessage(hServerWnd, WM_DDE_TERMINATE, hClientWnd, 0);	WM_DDE_TERMINATE

第 7 章 DDE クライアント・アプリケーションを使った DDE 関数の使用方法

Windows では、複数のアプリケーション・プログラムを同時に実行でき、Windows アプリケーション・プログラム間でデータを交換もできます。動的データ交換 (DDE) を使用すれば、このデータ交換が可能になります。Windows アプリケーション・プログラム間でのデータ交換は、クライアント とサーバー のアプリケーション・プログラム間での会話と考えることができます。クライアント・アプリケーションとは DDE を開始する側のアプリケーション・プログラムのことで、サーバー・アプリケーションとはクライアント・アプリケーションに対して応答する側のアプリケーション・プログラムのことです。

クライアント・アプリケーションがサーバー・アプリケーションとの間の DDE 会話を開始し、データ交換を行うには、サーバー・アプリケーションが認識できる 3 種類の名前 (アプリケーション・プログラム名、トピック名、アイテム名) が必要です。クライアント・アプリケーションは、アプリケーション名、トピック名を指定してサーバー・アプリケーションとの DDE 会話を開始し、アイテム名を指定して交換データを定義します。

パーソナル・コミュニケーションズは DDE サーバーとしての機能を持っているので、DDE クライアントとしての機能を持つ他の Windows アプリケーション・プログラム (Microsoft Visual Basic、 Microsoft Excel、 Microsoft Word など) との間に DDE 会話を確立できます。

パーソナル・コミュニケーションズの DDE インターフェースの使用

パーソナル・コミュニケーションズとの DDE 会話を開始しデータ交換を行うには、クライアント・アプリケーション・プログラム側で、パーソナル・コミュニケーションズが認識できるアプリケーション・プログラム名、トピック名、アイテム名を認識する必要があります。このアプリケーション名、トピック名、アイテム名の組み合わせによって、アプリケーション・プログラムとパーソナル・コミュニケーションズとの間で交換されるデータ・タイプが定義されます。

表 21. データ・アイテム用の命名形式

レベル	説明	例
アプリケーション	Windows タスク、またはアプリケーションの特定のタスク。本書ではアプリケーション・プログラムとはパーソナル・コミュニケーションズのことです。	IBM327032
トピック	アプリケーションの特定の部分。	SessionA
アイテム	DDE 会話中に受け渡されるデータのタイプ。	PS (表示スペース)

アプリケーション

パーソナル・コミュニケーションズは、Windows DDE サーバーとして、32 ビット・アプリケーション用に *IBM327032* または *IBM525032* をサポートし、16 ビット・アプリケーション用には *IBM3270* または *IBM5250* をサポートします。

トピック

アプリケーション内の会話を行うトピックを指定します。ユーザーが使用できるトピックは次の表のとおりです。

表 22. パーソナル・コミュニケーションズ用のトピック

トピック	会話名	会話タイプ
System	システム会話	コールド・リンク
SessionA、SessionB、...、SessionZ	セッション会話	コールド・リンクとホット・リンク
SessA_xxxx、SessB_xxxx、...、SessZ_xxxx	セッション会話	ホット・リンク

アイテム

クライアント・アプリケーション・プログラムは、パーソナル・コミュニケーションズとデータや情報を交換できます。データと情報のタイプは、アイテム名によって指定します。

次に、それぞれのトピックに対する会話を行う手順および使用できるデータ・アイテムについて説明します。

システム会話

パーソナル・コミュニケーションズのシステム DDE インターフェースを使用するには、次の手順に従ってください。

1. システム会話を開始します。
2. システム情報を要求します。
3. システム会話を終了します。

パーソナル・コミュニケーションズを使用して DDE システム会話を開始する

パーソナル・コミュニケーションズで DDE インターフェースを使用するには、最初に、クライアント・アプリケーションが、パーソナル・コミュニケーションズと DDE 会話を開始します。クライアント・アプリケーションの DDE 関数 (Initiate) に、アプリケーション名として 32 ビット・アプリケーションでは *IBM327032* または *IBM525032* を、 16 ビット・アプリケーションでは *IBM3270* または *IBM5250* を、およびトピック名には *System* を指定して DDE 会話を開始してください。

システム情報を要求する

DDE 会話を開始すると、クライアント・アプリケーションは DDE 関数を使用してデータまたは情報を要求できます。次の表に示されたアイテム名をクライアント・アプリケーションの DDE 関数 (Request) に指定することによって、システム情報を要求することができます。

アイテム	戻りデータ	DDE 関数
Formats	サポートされている Windows クリップボード・フォーマットのリスト	Get System Formats
状況	各セッション状況情報	Get System Status
SysCon	エミュレーターのサポート・レベルとその他のシステムに関連した値	Get System Configuration
SysItems	使用可能なデータ・アイテムのリスト	Get System SysItems
Topics	使用可能なトピックのリスト	Get System Topics

パーソナル・コミュニケーションズを使用して DDE システム会話を終了する

会話を完了するには、クライアント・アプリケーションはパーソナル・コミュニケーションズを使用して DDE 会話を終了する必要があります。クライアント・アプリケーションの DDE 関数 (Terminate) を使用して DDE 関数を終了させてください。

セッション会話

パーソナル・コミュニケーションズのセッション DDE インターフェースを使用するには、次の手順に従ってください。

1. セッション会話を開始します。
2. DDE 関数を使用します (Request、Poke、Execute)。
3. セッション会話を終了します。

DDE セッション会話を開始する

パーソナル・コミュニケーションズのセッションで DDE インターフェースを使用するには、クライアン ト・アプリケーションはパーソナル・コミュニケーションズと DDE 会話を開始する必要があります。 DDE 会話を開始するには、 *IBM327032* または *IBM525032* を 32 ビット・アプリケーションのアプリケーション名として指定します。 16 ビット・アプリケーションには *IBM3270* または *IBM5250* を指定し、 *SessionA*、*SessionB*、 ...、*SessionZ* をクライアント・アプリケーションの DDE 関数 (Initiate) にトピック名として指定します。

データを要求する

DDE 会話を開始すると、クライアント・アプリケーションは DDE 関数を使用してデータを要求できます。次の表に示されたアイテム名をクライアント・アプリケーションの DDE 関数 (Request) に指定することによって、セッション情報を要求できます。

アイテム	戻りデータ	DDE 関数
EPS(pos,len,bEOF)	セッション表示スペースの全部または一部	Get Partial Presentation Space
FIELD (pos, "type")	フィールド情報	Find Field
OIA	オペレーター情報域 (OIA) の状況行情報	Get Operator Information Area
PS	セッション表示スペース	Get Presentation Space

アイテム	戻りデータ	DDE 関数
SSTAT	セッション状況情報	Get System Status
STRING(pos,bDir,"string")	ストリング・オフセット開始	Search for string
TRIMRECT *	トリミング長方形のセッション表示スペース	Get Trim Rectangle
***:** パラメーターを追加する必要があります。		

エミュレーター・ウィンドウにデータを送信する (Poke)

DDE 会話を開始すると、クライアント・アプリケーションは DDE 関数を使用してパーソナル・コミュニケーションズのセッションにデータを送信できます。次の表に DDE 関数用の有効なアイテムを示します。

アイテム	説明	DDE 関数
EPS(pos,bEOF)	ASCII データ・ストリングをホスト表示スペースに送ります。	Put Data to Presentation Space
SETCURSOR	カーソル位置を設定します。	Set Cursor Position

コマンドを実行する

DDE 会話を開始すると、クライアント・アプリケーションは DDE 関数を使用して、コマンドをパーソナル・コミュニケーションズのセッション・ウィンドウに送信できます。コマンドをクライアント・アプリケーションの DDE 関数 (Execute) に指定してください。詳細については、273 ページの『Session Execute Macro』を参照してください。

DDE セッション会話を終了する

クライアント・アプリケーションは、タスク完了時にパーソナル・コミュニケーションズとの DDE 会話を終了する必要があります。クライアント・アプリケーションの DDE 関数 (Terminate) を使用して DDE 関数を終了させてください。

セッション会話 (ホット・リンク)

パーソナル・コミュニケーションズのセッション DDE インターフェースを使用するには、次の手順に従ってください。

1. セッション会話を開始します。
2. **Advise** 関数を開始します。
3. **Advise** 関数を停止します。
4. セッション会話を終了します。

DDE セッション会話を開始する (ホット・リンク)

パーソナル・コミュニケーションズのセッションで DDE インターフェースを使用するには、クライアン ト・アプリケーションはパーソナル・コミュニケーションズと DDE 会話を開始する必要があります。クライアント・アプリケーションの DDE 関数 (Initiate) に、アプリケーション名として 32 ビット・アプリケーションでは、*IBM327032* または *IBM525032* を、 16 ビット・アプリケーションでは *IBM3270* または *IBM5250* を、トピック名として *SessionA*、 *SessionB*、...、 *SessionZ* を指定して、DDE 会話を開始してください。

ホット・リンクをセッション・ウィンドウで開始する

DDE 会話を開始した後、クライアント・アプリケーションは、**Advise** 関数を開始することができます。ホット・リンクを開始するには、クライアント・アプリケーションで DDE 関数 (Advise) に以下のアイテム名を指定します。

アイテム	説明	DDE 関数
CLOSE	Window Close 要求の代行受信を開始します。	Start Close Intercept
KEYS	キー・ストロークの代行受信を開始します。	Start Keystroke Intercept
PS * OIA TRIMRECT *	PS、OIA、またはトリミング長方形のデータ取り出しを開始します。	Start Session Advise
***:** パラメーターを追加する必要があります。		

ホット・リンクをセッション・ウィンドウで停止する

Advise 関数を終了するには、クライアント・アプリケーションで DDE 関数を使用する必要があります。ホット・リンクを停止するには、クライアント・アプリケーションで DDE 関数 **Unadvise** に以下のアイテム名を指定します。

アイテム	説明	DDE 関数
CLOSE	ウィンドウの Close 要求の代行受信を停止します。	Stop Close Intercept
KEYS	キー・ストロークの代行受信を停止します。	Stop Keystroke Intercept
PS * OIA TRIMRECT *	セッションのアドバイス関数を停止します。	Stop Session Advise
***:** Start Session Advise が呼び出された時に使用していたパラメーターと同じものを使用します。		

DDE セッション会話を終了する

クライアント・アプリケーションは、タスク完了時にパーソナル・コミュニケーションズとの DDE 会話を終了する必要があります。クライアント・アプリケーションの DDE 関数 (Terminate) を使用して DDE 関数を終了させてください。

パーソナル・コミュニケーションズ DDE インターフェース

次の表に、他のアプリケーション (Microsoft Excel、Microsoft Word、Microsoft Visual Basic など) から使用できる DDE 関数を示します。

システム会話用の DDE 関数

- 331 ページの『Initiate System Conversation』
- 328 ページの『Get System Configuration』
- 329 ページの『Get System Formats』
- 329 ページの『Get System Status』

- 330 ページの『Get System SysItems』
- 330 ページの『Get System Topics』
- 331 ページの『Terminate System Conversation』

セッション会話用の DDE 関数

- 337 ページの『Initiate Session Conversation』 *1
- 331 ページの『Find Field』
- 333 ページの『Get Operator Information Area』
- 334 ページの『Get Partial Presentation Space』
- 335 ページの『Get Presentation Space』
- 336 ページの『Get Session Status』
- 336 ページの『Get Trim Rectangle』
- 337 ページの『Put Data to Presentation Space』
- 338 ページの『Search for String』
- 339 ページの『Session Execute Macro』
- 339 ページの『Set Cursor Position』
- 340 ページの『Terminate Session Conversation』 *2

セッション会話用の DDE 関数 (ホット・リンク)

- 340 ページの『Initiate Session Conversation』 (*1 と同じ)
- 341 ページの『Start Close Intercept』
- 341 ページの『Start Keystroke Intercept』
- 342 ページの『Start Session Advise』
- 343 ページの『Stop Close Intercept』
- 344 ページの『Stop Keystroke Intercept』
- 344 ページの『Stop Session Advise』
- 345 ページの『Terminate Session Conversation』 (*2 と同じ)

システム会話用の DDE 関数

パーソナル・コミュニケーションズのシステム会話に使用できる DDE 関数は、次のとおりです。

Get System Configuration

Get System Configuration 関数は、パーソナル・コミュニケーションズのサポート・レベルとその他のシステム関連の値を戻します。

DDE パラメーター	値
アイテム	SysCon

クライアント・アプリケーションは、トピック名 (System) でクライアント・アプリケーションの DDE 関数 (Request) を指定することにより、パーソナル・コミュニケーションズの DDE 関数を利用できます。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズのシステムは、パーソナル・コミュニケーションズのシステム構成データ・アイテムを戻します。

戻り情報: 詳細については、 261 ページの『Get System Configuration』を参照してください。

パーソナル・コミュニケーションズがシステム構成データ・アイテムを戻さない場合、次のような理由が考えられます。

- 無効なアイテム名が指定されました。
- システム・エラーが発生しました。

Get System Formats

Get System Formats 関数は、パーソナル・コミュニケーションズでサポートされている Windows クリップボード・フォーマットのリストを戻します。

DDE パラメーター	値
アイテム	Formats

クライアント・アプリケーションは、前述のアイテム名 (Formats) をクライアント・アプリケーションの DDE 関数 (Request) に指定することにより、パーソナル・コミュニケーションズの DDE 関数 を使用できます。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズは、サポートされている Windows クリップボード・フォーマットのリストを戻します。

パーソナル・コミュニケーションズがフォーマット・データ・アイテムを戻さない場合、次のような理由が考えられます。

- 無効なアイテム名が指定されました。
- システム・エラーが発生しました。

Get System Status

Get System Status 関数は、構成されているパーソナル・コミュニケーションズの各セッションの状況を戻します。

DDE パラメーター	値
アイテム	SysCon

クライアント・アプリケーションは、前述のアイテム名 (SysCon) をクライアント・アプリケーションの DDE 関数 (Request) に指定することにより、パーソナル・コミュニケーションズの DDE 関数を使用できます。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズは、各オープン・セッションに対して一連の状況情報を戻します。

戻り情報: 詳細については、 263 ページの『Get System Status』を参照してください。

パーソナル・コミュニケーションズが状況データ・アイテムを戻さない場合、次のような理由が考えられます。

- 無効なアイテム名が指定されました。
- システム・エラーが発生しました。

Get System SysItems

Get System SysItems 関数は、パーソナル・コミュニケーションズのシステム・トピックで使用できるデータ・アイテムのリストを戻します。

DDE パラメーター	値
アイテム	SysItems

クライアント・アプリケーションは、前述のアイテム名 (SysItem) をクライアント・アプリケーションの DDE 関数 (Request) に指定することにより、パーソナル・コミュニケーションズの DDE 関数を使用できます。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズは、パーソナル・コミュニケーションズのシステム・トピック・データ・アイテムのリストを戻します。パーソナル・コミュニケーションズでサポートされるデータ・アイテムは次のとおりです。

- SysItems
- Topics
- Status
- Formats
- SysCon

パーソナル・コミュニケーションズがシステム・データ・アイテムを戻さない場合、次のような理由が考えられます。

- 無効なアイテム名が指定されました。
- システム・エラーが発生しました。

Get System Topics

Get System Topics は、パーソナル・コミュニケーションズによってサポートされている活動状態の DDE トピックのリストを戻します。

DDE パラメーター	値
アイテム	Topics

クライアント・アプリケーションは、前述のアイテム名 (Topics) をクライアント・アプリケーションの DDE 関数 (Request) に指定することにより、パーソナル・コミュニケーションズの DDE 関数を使用できます。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズでサポートされるトピックは次のとおりです。

- System
- SessionA、SessionB、...、SessionZ

パーソナル・コミュニケーションズがシステム・データ・アイテムを戻さない場合、次のような理由が考えられます。

- 無効なアイテム名が指定されました。
- システム・エラーが発生しました。

Initiate System Conversation

Initiate System Conversation 関数はシステム会話を開始します。 1 つのシステムに接続できるのは、1 つのクライアント・アプリケーションだけです。

DDE パラメーター	**値**
トピック	System

クライアント・アプリケーションは、パーソナル・コミュニケーションズのアプリケーション名 (32 ビット・アプリケーションの場合は IBM327032 または IBM525032) または (16 ビット・アプリケーションの場合は IBM3270 または IBM5250) 、およびトピック名 (System) を指定した DDE 関数 (Initiate) を使用して、DDE 会話を開始する必要があります。

Terminate System Conversation

Terminate System Conversation 関数は、システム会話を終了させます。 DDE 関数 (Terminate) を使用して、クライアント・アプリケーションから DDE 会話を終了します。

セッション会話用の DDE 関数

パーソナル・コミュニケーションズのセッション会話に使用できる DDE 関数は、次のとおりです。

Find Field

Find Field 関数は、クライアント・アプリケーションにフィールド情報を渡します。

DDE パラメーター	**値**
アイテム	FIELD (pos, "type")

パラメーター	値	説明
pos	NNNN	PS 位置

パラメーター	値	説明
"タイプ"	"␣␣" または "T␣" "P␣" "N␣" "NP" "NU" "PP" "PU"	このフィールド。前のフィールド。保護、無保護のいずれでも可。次のフィールド。保護または無保護。次の保護フィールド。次の無保護フィールド。前の保護フィールド。前の無保護フィールド。

注: ␣ 記号は、必要なブランクを表します。

IBM パーソナル・コミュニケーションズ J3.1 フォーマットのアイテムもサポートされています。

クライアント・アプリケーションは、前述のアイテム名を伴ったクライアント・アプリケーションの DDE 関数 (Request) を指定することにより、パーソナル・コミュニケーションズの DDE 関数を使用することができます。

パーソナル・コミュニケーションズの応答

PC/3270 によって戻されるフィールド情報は次のとおりです。

フィールド	戻り情報	説明
Formatted/Unformatted	フォーマット済み、不定様式	表示スペースはフォーマット済みまたは不定様式。不定様式を指定すると、その他のどのフィールド情報も戻されません。
Unprotected/Protected	N	0 = 無保護データ・フィールド。1 = 保護データ・フィールド。
A/N	N	0 = 英数字。1 = 数字。
I/SPD	N	0 = 普通輝度、検出不能 1 = 普通輝度、検出可能 2 = 高輝度、検出可能 3 = 非表示、検出不能
MDT	N	0 = フィールドは変更されていません。1 = フィールドは変更されています。
Field start offset	NNNN	フィールドはこのフィールド位置から始まります
Field length	NNNN	フィールド長

PC400 によって戻されるフィールド情報は次のとおりです。

フィールド	戻り情報	説明
Formatted/Unformatted	フォーマット済み、不定様式	表示スペースはフォーマット済みまたは不定様式。不定様式を指定すると、その他のどのフィールド情報も戻されません。
Field attribute	N	0 = 非フィールド属性バイト。1 = フィールド属性バイト。
可視性	N	0 = 非表示。1 = 表示。
Unprotected/Protected	N	0 = 無保護データ・フィールド。1 = 保護データ・フィールド。
輝度	N	0 = 通常輝度。1 = 高輝度。

フィールド	戻り情報	説明
Field Type	N	0 = 英数字: 全文字を許可します。 1 = 英字のみ: 大文字小文字、コンマ、ピリオド、ハイフン、ブランク、複写キーを許可します。2 = 数字シフト: 数字の自動シフト。 3 = 数字のみ: 0 から 9 までの数字、コンマ、ピリオド、プラス、マイナス、ブランク、複写キーを許可します。5 = 数字のみ: 0 から 9 までの数字と複写キーを許可します。 6 = 磁気ストライプ読み取り装置データのみ。 7 = 符号付き数字: 0 ～ 9 の数字、プラス、マイナス、複写キーを許可します。
MDT	N	0 = フィールドは変更されていません。1 = フィールドは変更されています。
Field start offset	NNNN	フィールドはこのフィールド位置から始まります
Field length	NNNN	フィールド長

パーソナル・コミュニケーションズがフィールド情報を戻さない場合、次のような理由が考えられます。

- 無効なアイテム名が指定されました。
- システム・エラーが発生しました。

Get Operator Information Area

Get Operator Information Area 関数は、 OIA 情報をクライアント・アプリケーションに戻します。

DDE パラメーター **値**
アイテム OIA

クライアント・アプリケーションは、前述のアイテム名 (OIA) をクライアント・アプリケーションの DDE 関数 (Request) に指定することにより、パーソナル・コミュニケーションズの DDE 関数を使用できます。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズが戻す OIA 情報は次のとおりです。

オフセット	戻り情報	意味
0	ONLINE LU-LU SSCP-LU	オンラインで、画面を所有していません、LU-LU セッションが画面を所有しています、 SSCP-LU セッションが画面を所有しています

オフセット	戻り情報	意味
9	X X MCHK X CCHK X PCHK X DNW X BUSY X TWAIT X -S X -f X MUCH X UA X -fUA X DEAD X WRONG X SYSTEM X II	入力禁止、マシン・チェック、通信チェック、プログラム・チェック、装置が動作していない、印刷中、端末が待機中、マイナス記号、マイナス関数、入力過多、オペレーターが許可されていない、マイナス関数、無効なキーの組み合わせ、位置が無効、システムが待機中、オペレーターの入力エラー (PC400)
19	COMM	通信エラー
25	MW	メッセージ待ち (PC400)
36	APL	APL (PC/3270)
42	U NUM	大文字 数字
43	A	Caps lock
47	S I	高輝度、オペレーターが選択可能高輝度、フィールド継承
49	CS CI	カラー、オペレーターが選択可能カラー、フィールド継承
52	^	挿入モード
61	P-MAL P-PRN P-ASS	プリンター誤動作、プリンターが印刷中、プリンター割り当て

パーソナル・コミュニケーションズが OIA 情報を戻さない場合、次のような理由が考えられます。

- 無効なアイテム名が指定されました。
- システム・エラーが発生しました。

Get Partial Presentation Space

Get Partial Presentation Space 関数は、セッション表示スペースの一部または全部をクライアント・アプリケーションに戻します。

DDE パラメーター 値
アイテム EPS (pos, len, bEOF)

パラメーター	値	説明
pos	NNNN	PS 位置
len	NNNN	PS の長さ
bEOF	1 または 0	EOF 切り替え **1** YES **0** NO

注: IBM パーソナル・コミュニケーションズ J3.1 フォーマットのアイテムもサポートされています。

クライアント・アプリケーションは、前述のアイテム名を伴ったクライアント・アプリケーションの DDE 関数 (Request) を指定することにより、パーソナル・コミュニケーションズの DDE 関数を使用することができます。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズが戻す情報は次のとおりです。

フィールド	戻り情報	説明
PS 開始位置	NNNN	pos パラメーターで指定されます
PS の長さ	NNNN	len パラメーターで指定されます
PS 行数	NNNN	行数で指定されます
PS 桁数	NNNN	桁数で指定されます
PS	NNNN	PS データはこの位置から始まります

パーソナル・コミュニケーションズがフォーマット・データ・アイテムを戻さない場合、次のような理由が考えられます。

- 無効なアイテム名が指定されました。
- システム・エラーが発生しました。

Get Presentation Space

Get Presentation Space 関数は、セッション表示スペース・データをクライアント・アプリケーションに戻します。

DDE パラメーター	値
アイテム	PS

クライアント・アプリケーションは、前述のアイテム名 (PS) をクライアント・アプリケーションの DDE 関数 (Request) に指定することにより、パーソナル・コミュニケーションズの DDE 関数を使用できます。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズが戻す情報は次のとおりです。

フィールド	戻り情報	説明
PS サイズ	NNNN	表示スペースのサイズ
PS 行数	NNNN	行数
PS 桁数	NNNN	桁数
PS	NNNN	PS データはこの位置から始まります

パーソナル・コミュニケーションズがフォーマット・データ・アイテムを戻さない場合、次のような理由が考えられます。

- 無効なアイテム名が指定されました。
- システム・エラーが発生しました。

Get Session Status

Get Session Status 関数は、接続セッション状況をクライアント・アプリケーションに戻します。

DDE パラメーター	**値**
アイテム	SSTAT

クライアント・アプリケーションは、前述のアイテム名 (SSTAT) をクライアント・アプリケーションの DDE 関数 (Request) に指定することにより、パーソナル・コミュニケーションズの DDE 関数を使用できます。

パーソナル・コミュニケーションズの応答

戻り情報については、 410 ページの『Get Session Status』を参照してください。

パーソナル・コミュニケーションズがフォーマット・データ・アイテムを戻さない場合、次のような理由が考えられます。

- 無効なアイテム名が指定されました。
- システム・エラーが発生しました。

Get Trim Rectangle

Get Trim Rectangle 関数は、現在の (または指定された) トリミング長方形の表示スペース・エリアをクライアント・アプリケーションに戻します。

DDE パラメーター	**値**
アイテム	TRIMRECT (row1, col1, row2, col2) TRIMRECT (pos1, pos2) TRIMRECT

パラメーター	値	説明
row1	NN	トリミング長方形の左上隅の行
col1	NN	トリミング長方形の左上隅の桁
row2	NN	トリミング長方形の右下隅の行
col2	NN	トリミング長方形の右下隅の桁
pos1	NNNN	トリミング長方形の左上隅の PS 位置
pos2	NNNN	トリミング長方形の右下隅の PS 位置

注: IBM パーソナル・コミュニケーションズ J3.1 フォーマットのアイテムもサポートされています。

クライアント・アプリケーションがパラメーターに PS トリミング長方形を指定しなければ、現在指定されている PS トリミング長方形が使用されます。

クライアント・アプリケーションは、前述のアイテム名 (TRIMRECT) をクライアント・アプリケーションの DDE 関数 (Request) に指定することにより、パーソナル・コミュニケーションズの DDE 関数を利用できます。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズが戻す情報は次のとおりです。

フィールド	戻り情報	説明
PS		PS データはこの位置から始まります

パーソナル・コミュニケーションズがトリミング長方形アイテムを戻さない場合、次のような理由が考えられます。

- 無効なアイテム名が指定されました。
- システム・エラーが発生しました。

Initiate Session Conversation

Initiate Session Conversation 関数は、使用可能なセッション・ウィンドウで DDE 会話を開始します。 1 つのセッション会話に接続できるのは、1 つのクライアント・アプリケーションだけです。

DDE パラメーター　値
トピック　SessionA、SessionB、...、SessionZ

パラメーター値	説明
SessionA、SessionB、...、SessionZ	"SessionA" とは、"Session" にセッション ID『A』、『B』、...、『Z』のいずれかを追加したストリングです。

クライアント・アプリケーションは、前述のトピック名 (SessionA、SessionB、...、SessionZ) をクライアント・アプリケーションの DDE 関数 (Initiate) に指定することにより、DDE 会話を開始する必要があります。

パーソナル・コミュニケーションズの応答

トピックが指定されていない場合、パーソナル・コミュニケーションズは次の使用可能なトピックを確認してから応答します。

- System
- SessionA、SessionB、...、SessionZ

Put Data to Presentation Space

Put Data to Presentation Space 関数は、指定されたホストの表示スペースに ASCII データ・ストリングを送信し、書き込みます。

DDE パラメーター　値
アイテム　EPS (pos, bEOF)

パラメーター	値	説明
pos	NNNN	データの書き込みを開始する PS 位置

パラメーター	値	説明
bEOF	1 または 0	EOF 切り替え **1** YES **0** NO

注: IBM パーソナル・コミュニケーションズ J3.1 フォーマットのアイテムもサポートされています。

クライアント・アプリケーションは、前述のアイテム名を伴ったクライアント・アプリケーションの DDE 関数 (Poke) を指定することにより、パーソナル・コミュニケーションズの DDE 関数を使用することができます。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズがストリング・データを受け入れない場合、次のような理由が考えられます。

- 無効なアイテム名が指定されました。
- PS 位置が無効です。
- 長さが無効です。
- PS 入力が禁止されました。
- システム・エラーが発生しました。

Search for String

Search for String 関数により、クライアント・アプリケーションは指定されたストリングが指定された表示スペース区域内に存在するかどうかをチェックできます。

DDE パラメーター 値
アイテム STRING (pos, bDir, "string")

パラメーター	値	説明
pos	NNNN	ストリング検索の PS 開始位置
bDir	1 または 0	検索方向 **1** 前方 **0** 後方
"string"		検索ストリング • ブランクを含むストリングは二重引用符で囲みます。 • ストリング内に二重引用符を指定する場合は、二重引用符を二重引用符で囲みます。例: *This is a double quotation" mark.* は、*"This is a double quotation"" mark."* と指定します。

注: IBM パーソナル・コミュニケーションズ J3.1 フォーマットのアイテムもサポートされています。

検索ストリングの最大長は 255 バイトです。

クライアント・アプリケーションは、前述のアイテム名を伴ったクライアント・アプリケーションの DDE 関数 (Request) を指定することにより、パーソナル・コミュニケーションズの DDE 関数を使用することができます。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズが戻す情報は次のとおりです。

フィールド	戻り情報	説明
ストリング開始オフセット	NNNN、なし	ストリングが見つからない場合、 "None" が戻されます

パーソナル・コミュニケーションズがストリングの開始位置を戻さない場合、次のような理由が考えられます。

- 無効なアイテム名が指定されました。
- PS 位置が無効、またはストリングが長すぎます。
- システム・エラーが発生しました。

Session Execute Macro

Session Execute Macro 関数は、ユーザーがコマンドとマクロ・ストリングをパーソナル・コミュニケーションズに送信できるようにします。

コマンドおよびストリングの詳細については、 423 ページの『Session Execute Macro』を参照してください。

クライアント・アプリケーションは、クライアント・アプリケーションの DDE 関数 (Execute) を指定することにより、パーソナル・コミュニケーションズの DDE 関数を使用できます。

パーソナル・コミュニケーションズの応答

システム・エラーのためにパーソナル・コミュニケーションズがストリング開始位置を戻さないことがあります。

Set Cursor Position

Set Cursor Position 関数により、クライアント・アプリケーションはセッション・ウィンドウのカーソル位置を設定できます。

DDE パラメーター **値**
アイテム SETCURSOR
データ (カーソル位置) NNNN または Rn1Rn2

パラメーター値	説明
NNNN	PS オフセット

パラメーター値	説明
Rn1Rn2	行/桁 **n1** PS 位置の行 **n2** PS 位置の桁

クライアント・アプリケーションは、前述のアイテム名をクライアント・アプリケーションの DDE 関数 (Poke) に指定することにより、パーソナル・コミュニケーションズの DDE 関数を利用できます。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズが指定された PS 位置にカーソルを移動できない場合、次のような理由が考えられます。

- 無効なアイテム名が指定されました。
- カーソルの PS オフセットが無効です。(有効値: 0 から (PS サイズ - 1))
- カーソル行数の値が無効です。(有効値: 0 から (PS 行数 - 1))
- カーソル桁数の値が無効です。(有効値: 0 から (PS 桁数 - 1))
- システム・エラーが発生しました。

Terminate Session Conversation

Terminate Session Conversation 関数は、クライアント・アプリケーションとパーソナル・コミュニケーションズ間の DDE 会話を終了させます。

クライアント・アプリケーションの DDE 関数 (Terminate) を使用して、DDE 会話を終了させてください。

セッション会話用の DDE 関数 (ホット・リンク)

パーソナル・コミュニケーションズのセッション会話をホット・リンク接続によって使用できる DDE 関数は、次のとおりです。

Initiate Session Conversation

Initiate Session Conversation 関数は、使用可能なセッション・ウィンドウで DDE 会話を開始します。

DDE パラメーター	値
トピック	SessionA、SessionB、...、SessionZ または SessA_xxxx、SessB_xxxx、...、 SessZ_xxxx

注: SessA_xxxx、SessB_xxxx、...、SessZ_xxxx が指定されると、クライアント・アプリケーションはホット・リンク・セッション会話だけを許可します。

パラメーター値	説明
SessA_xxxx、SessB_xxxx、...、SessZ_xxxx	'SessA_xxxx' は、(SessA_) の後ろに任意のユーザー定義ストリング (xxxx) を付けたストリングです。ユーザー定義ストリングの長さに制限はありません。

クライアント・アプリケーションの DDE 関数 (Initiate) にパーソナル・コミュニケーションズのアプリケーション名と前述のトピック名を指定して、DDE 会話を開始します。

Start Close Intercept

Start Close Intercept 関数を使用すると、クライアント・アプリケーションは、エミュレーター・セッション・ウィンドウからクローズ・オプションを選択して生成したクローズ要求を代行受信できます。このサービスが開始されると、クライアント・アプリケーションはクローズ要求イベント・データを受信します。

DDE パラメーター　値
アイテム　CLOSE

クライアント・アプリケーションは、前述のアイテム名をクライアント・アプリケーションの DDE 関数 (Advise) に指定することにより、パーソナル・コミュニケーションズの DDE 関数を利用できます。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズが戻す情報は次のとおりです。

フィールド	戻り情報	説明
PS クローズ要求数	NNNN	クローズ要求が生成されると、クライアント・アプリケーションは "0001" を受信します。

パーソナル・コミュニケーションズが Close Intercept を開始しない場合、次のような理由が考えられます。

- 無効なアイテム名が指定されました。
- セッションのクローズ要求の代行受信が、同じトピック名ですでに開始されています。
- システム・エラーが発生しました。

Start Keystroke Intercept

Start Keystroke Intercept 関数を使用すると、クライアント・アプリケーションは、端末オペレーターが入力するキー・ストロークをフィルターに掛けられます。開始後、キー・ストロークはクライアント・アプリケーションによって代行受信されます。

DDE パラメーター　値
アイテム　KEYS

クライアント・アプリケーションは、前述のアイテム名をクライアント・アプリケーションの DDE 関数 (Advise) に指定することにより、パーソナル・コミュニケーションズの DDE 関数を利用できます。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズが戻す情報は次のとおりです。

フィールド	戻り情報	説明
Keys		275 ページの表 19を参照してください。

パーソナル・コミュニケーションズが KeyStroke Intercept を開始しない場合、次のような理由が考えられます。

- 無効なアイテム名が指定されました。
- セクションのキー・ストローク代行受信が、同じトピック名ですでに開始されています。
- システム・エラーが発生しました。

Start Session Advise

Start Session Advise 関数は、クライアント・アプリケーションとパーソナル・コミュニケーションズとの間のリンクを確立します。データ・アイテムが変更されると、クライアント・アプリケーションは表示スペース (PS)、オペレーター情報域 (OIA)、トリミング長方形 (TRIMRECT) の変更データを受信します。

DDE パラメーター **値**
アイテム PS (pos, len, bCaseSen, "string") PS TRIMRECT (row1, col1, row2, col2) TRIMRECT (pos1, pos2) TRIMRECT OIA

検索ストリングの最大長は 255 バイトです。

パラメーター	値	説明
pos	NNNN	ストリング検索の PS 開始位置 (PS オフセット)
len	NNNN	検索ストリングの長さ
bCaseSen	1 または 0	大文字・小文字の区別 **1** YES **0** NO
"string"		検索ストリング • ブランクを含むストリングは二重引用符で囲みます。 • ストリング内に二重引用符を指定する場合は、二重引用符を二重引用符で囲みます。例: *This is a double quotation" mark.* は、*"This is a double quotation"" mark."* と指定します。

注: IBM パーソナル・コミュニケーションズ J3.1 フォーマットのアイテムもサポートされています。

表示スペースの更新時に条件付きアドバイスを受信したい場合は、クライアント・アプリケーションが前述のパラメーター値およびアドバイス条件を設定する必要があります。前述のパラメーターは、表示スペースがアイテム名として指定されているときに使用できます。

パラメーター	値	説明
row1	NN	トリミング長方形の左上隅の行
col1	NN	トリミング長方形の左上隅の桁
row2	NN	トリミング長方形の右下隅の行
col2	NN	トリミング長方形の右下隅の桁
pos1	NNNN	トリミング長方形の左上隅の PS 位置
pos2	NNNN	トリミング長方形の右下隅の PS 位置

注: IBM パーソナル・コミュニケーションズ J3.1 フォーマットのアイテムもサポートされています。

クライアント・アプリケーションがアイテム名パラメーターに表示スペース・トリミング長方形を指定しなければ、現在指定されている表示スペース・トリミング長方形が使用されます。アイテム名に TRIMRECT が指定されると、このパラメーター値を使用できます。

クライアント・アプリケーションは、前述のアイテム名をクライアント・アプリケーションの DDE 関数 (Advise) に指定することにより、パーソナル・コミュニケーションズの DDE 関数を利用できます。

パーソナル・コミュニケーションズの応答

334 ページの『Get Partial Presentation Space』、 333 ページの『Get Operator Information Area』、および 336 ページの『Get Trim Rectangle』を参照してください。

パーソナル・コミュニケーションズがアドバイスを開始しない場合、次のような理由が考えられます。

- 無効なアイテム名が指定されました。
- そのセッションのアドバイスが、同じトピック名ですでに開始されています。
- システム・エラーが発生しました。

Stop Close Intercept

Stop Close Intercept 関数を使用して、クライアント・アプリケーションはクローズ要求の代行受信を中止します。

DDE パラメーター　値
アイテム　CLOSE

クライアント・アプリケーションは、前述のアイテム名をクライアント・アプリケーションの DDE 関数 **Unadvise** に指定することにより、パーソナル・コミュニケーションズの DDE 関数を利用できます。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズが Close Intercept を中止しない場合、次のような理由が考えられます。

- アドバイスはまだ始動していません。
- システム・エラーが発生しました。

Stop Keystroke Intercept

Stop Keystroke Intercept 関数を使用して、クライアント・アプリケーションはキー・ストロークの代行受信を中止します。

DDE パラメーター	値
アイテム	KEYS

クライアント・アプリケーションは、前述のアイテム名をクライアント・アプリケーションの DDE 関数 **Unadvise** に指定することにより、パーソナル・コミュニケーションズの DDE 関数を利用できます。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズが Keystroke Intercept を中止しない場合、次のような理由が考えられます。

- 無効なアイテム名が指定されました。
- アドバイスはまだ始動していません。
- システム・エラーが発生しました。

Stop Session Advise

Stop Session Advise 関数は、クライアント・アプリケーションとパーソナル・コミュニケーションズとの間のリンクをクローズします。

DDE パラメーター	値
アイテム	PS (pos, len, bCaseSen, "string") PS TRIMRECT (row1, col1, row2, col2) TRIMRECT (pos1, pos2) TRIMRECT OIA

検索ストリングの最大長は 255 バイトです。

アイテム名は、**Start Session Advise** が呼び出された時に指定したアイテム名と同じでなければなりません。

クライアント・アプリケーションは、前述のアイテム名をクライアント・アプリケーションの DDE 関数 **Unadvise** に指定することにより、パーソナル・コミュニケーションズの DDE 関数を利用できます。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズがアドバイスを中止しない場合、次のような理由が考えられます。

- 無効なアイテム名が指定されました。
- アドバイスはまだ始動していません。
- システム・エラーが発生しました。

Terminate Session Conversation

Terminate Session Conversation 関数は、クライアント・アプリケーションとパーソナル・コミュニケーションズ・セッション間の DDE 会話を終了させます。

クライアント・アプリケーションの DDE 関数 (Terminate) を使用して、DDE 会話を終了させてください。

Visual Basic のサンプル・プログラム

次のプログラムは Visual Basic のサンプル・プログラムです。

注: なお、このサンプル・プログラムは簡略化されており、提供される実際のサンプル・ファイルとは異なります。

```
'/************************************************/
'/*                                              */
'/*     System conversation                      */
'/*                                              */
'/************************************************/

'************************************************
'***                                          ***
'***    Initiate System Conversation          ***
'***                                          ***
'************************************************
'
'  Start DDE Conversation with system
'
Sub Command1_Click ()
On Error GoTo ErrHandlerInit
   Dim COLD As Integer
   COLD = 2
   FunctionComp& = True

   DoEvents
   Text1.LinkTopic = "|System"
   Text1.LinkMode  = COLD

   If FunctionComp&= False Then
      MsgBox "Error has occurred", 48, "DDE sample"
   End If
   Exit Sub

ErrHandlerInit:
   FunctionComp& = False
   Resume Next
End Sub

'-- note ----------------------------------------
'
' If you use VisualBasic Version2.0, use "DoEvents"
' function before starting DDE conversation by
```

```
' calling "LinkMode" function.
'
'-------------------------------------------------
'*************************************************
'***                                           ***
'***    Get System Format                      ***
'***                                           ***
'*************************************************
'
'   Request a list of パーソナル・コミュニケーションズ'   Clipboard Format
'
Sub Command2_Click ()
On Error GoTo ErrHandler
   FunctionComp& = True

   Text1.LinkItem = "Formats"
   Text1.LinkRequest

   If FunctionComp&= False Then
      MsgBox "Error has occurred", 48, "DDE sample"
   End If
   Exit Sub

ErrHandler:
   FunctionComp& = False
   Resume Next
End Sub

'*************************************************
'***                                           ***
'***    Get System Status                      ***
'***                                           ***
'*************************************************
'
'   Requests each パーソナル・コミュニケーションズ'   Session Status
'
Sub Command2_Click ()
On Error GoTo ErrHandler
   FunctionComp& = True

   Text1.LinkItem = "Status"
   Text1.LinkRequest

   If FunctionComp&= False Then
      MsgBox "Error has occurred", 48, "DDE sample"
   End If
   Exit Sub

ErrHandler:
   FunctionComp& = False
   Resume Next
End Sub

'*************************************************
'***                                           ***
'***    Get System Configuration               ***
'***                                           ***
'*************************************************
'
'   Requests パーソナル・コミュニケーションズ'   System Configuration Values
'
Sub Command2_Click ()
On Error GoTo ErrHandler
   FunctionComp& = True

   Text1.LinkItem = "SysCon"
   Text1.LinkRequest
```

```
   If FunctionComp&= False Then
      MsgBox "Error has occurred", 48, "DDE sample"
   End If
   Exit Sub

ErrHandler:
   FunctionComp& = False
   Resume Next
End Sub

'*************************************************
'***                                           ***
'***    Get System SysItems                    ***
'***                                           ***
'*************************************************
'
'   Requests a list of Data Items for
'   パーソナル・コミュニケーションズ System Conversation
'
Sub Command2_Click ()
On Error GoTo ErrHandler
   FunctionComp& = True

   Text1.LinkItem = "SysItems"
   Text1.LinkRequest

   If FunctionComp&= False Then
      MsgBox "Error has occurred", 48, "DDE sample"
   End If
   Exit Sub

ErrHandler:
   FunctionComp& = False
   Resume Next
End Sub

'*************************************************
'***                                           ***
'***    Get System Topics                      ***
'***                                           ***
'*************************************************
'
'   Requests a list of パーソナル・コミュニケーションズ'   Topics
'
Sub Command2_Click ()
On Error GoTo ErrHandler
   FunctionComp& = True

   Text1.LinkItem = "Topics"
   Text1.LinkRequest

   If FunctionComp&= False Then
      MsgBox "Error has occurred", 48, "DDE sample"
   End If
   Exit Sub

ErrHandler:
   FunctionComp& = False
   Resume Next
End Sub

'*************************************************
'***                                           ***
'***    Terminate System Conversation          ***
'***                                           ***
'*************************************************
```

```
'
'  Terminates DDE Conversation with system
'
Sub Command3_Click ()
On Error GoTo ErrHandler
   Dim NONE As IntegerTerm
   NONE = 0
   FunctionComp& = True

   Text1.LinkMode = NONE

   If FunctionComp&= False Then
      MsgBox "Error has occurred", 48, "DDE sample"
   End If
   Exit Sub

ErrHandlerTerm:
   FunctionComp& = False
   Resume Next
End Sub

'/*************************************************/
'/*                                               */
'/*     Session conversation                      */
'/*                                               */
'/*************************************************/

'*************************************************
'***                                           ***
'***    Initiate Session Conversation          ***
'***                                           ***
'*************************************************
'
'  Initiate DDE Conversation with system
'
Sub Command1_Click ()
On Error GoTo ErrHandlerInit
   Dim COLD As Integer
   COLD = 2
   FunctionComp& = True

   DoEvents
   Text1.LinkTopic = "|SessionA"
   Text1.LinkMode = COLD

   If FunctionComp&= False Then
      MsgBox "Error has occurred", 48, "DDE sample"
   End If
   Exit Sub

ErrHandlerInit:
   FunctionComp& = False
   Resume Next
End Sub

'-- note --------------------------------------------
'
' If you use VisualBasic Version2.0, use "DoEvents"
' function before starting DDE conversation by
' calling "LinkMode" function.
'
'--------------------------------------------------
'*************************************************
'***                                           ***
'***    Find Field                             ***
'***                                           ***
'*************************************************
```

```
'
'   Requests 100 Field Information of PS Position
'
Sub Command2_Click ()
On Error GoTo ErrHandler
   FunctionComp& = True

   Text1.LinkItem = "FILED(100,""  "")"
   Text1.LinkRequest

   If FunctionComp&= False Then
      MsgBox "Error has occurred", 48, "DDE sample"
   End If
   Exit Sub

ErrHandler:
   FunctionComp& = False
   Resume Next
End Sub

'************************************************
'***                                          ***
'***    Get Operator Information Area         ***
'***                                          ***
'************************************************
'
'   Requests OIA Data
'
Sub Command2_Click ()
On Error GoTo ErrHandler
   FunctionComp& = True

   Text1.LinkItem = "OIA"
   Text1.LinkRequest

   If FunctionComp&= False Then
      MsgBox "Error has occurred", 48, "DDE sample"
   End If
   Exit Sub

ErrHandler:
   FunctionComp& = False
   Resume Next
End Sub
'************************************************
'***                                          ***
'***    Get Partial Presentation Space        ***
'***                                          ***
'************************************************
'
'   Requests PS Data Bytes from PS Position from 100 to 1000
'
Sub Command2_Click ()
On Error GoTo ErrHandler
   FunctionComp& = True

   Text1.LinkItem = "EPS(100,1000,1)"
   Text1.LinkRequest

   If FunctionComp&= False Then
      MsgBox "Error has occurred", 48, "DDE sample"
   End If
   Exit Sub

ErrHandler:
   FunctionComp& = False
   Resume Next
```

```
End Sub

'*************************************************
'***                                           ***
'***    Get Presentation Space                 ***
'***                                           ***
'*************************************************
'
'   Requests PS Data
'
Sub Command2_Click ()
On Error GoTo ErrHandler
   FunctionComp& = True

   Text1.LinkItem = "PS"
   Text1.LinkRequest

   If FunctionComp&= False Then
      MsgBox "Error has occurred", 48, "DDE sample"
   End If
   Exit Sub

ErrHandler:
   FunctionComp& = False
   Resume Next
End Sub

'*************************************************
'***                                           ***
'***    Get Session Status                     ***
'***                                           ***
'*************************************************
'
'   Requests Session Connection Status
'
Sub Command2_Click ()
On Error GoTo ErrHandler
   FunctionComp& = True

   Text1.LinkItem = "SSTAT"
   Text1.LinkRequest

   If FunctionComp&= False Then
      MsgBox "Error has occurred", 48, "DDE sample"
   End If
   Exit Sub

ErrHandler:
   FunctionComp& = False
   Resume Next
End Sub
'*************************************************
'***                                           ***
'***    Get Trim Rectangle                     ***
'***                                           ***
'*************************************************
'
'   Requests PS Data in Current Specified Trim Rectangle
'
Sub Command2_Click ()
On Error GoTo ErrHandler
   FunctionComp& = True

   Text1.LinkItem = "TRIMRECT"
   Text1.LinkRequest

   If FunctionComp&= False Then
```

```
      MsgBox "Error has occurred", 48, "DDE sample"
   End If
   Exit Sub

ErrHandler:
   FunctionComp& = False
   Resume Next
End Sub

'*************************************************
'***                                           ***
'***    Put Data to Presentation Space         ***
'***                                           ***
'*************************************************
'
'   Writes string "Hello, World!" from PS Position 200
'
Sub Command2_Click ()
On Error GoTo ErrHandler
   FunctionComp& = True

   Text1.Text     = "Hello, World!"
   Text1.LinkItem = "EPS(200,1)"
   Text1.LinkPoke

   If FunctionComp&= False Then
      MsgBox "Error has occurred", 48, "DDE sample"
   End If
   Exit Sub

ErrHandler:
   FunctionComp& = False
   Resume Next
End Sub

'*************************************************
'***                                           ***
'***    Search for String                      ***
'***                                           ***
'*************************************************

'
'   Search forward for string "Hello!" from PS Position 1
'
Sub Command2_Click ()
On Error GoTo ErrHandler
   FunctionComp& = True

   Text1.LinkItem  = "STRING(1,1,""Hello!"")"
   Text1.LinkRequest

   If FunctionComp&= False Then
      MsgBox "Error has occurred", 48, "DDE sample"
   End If
   Exit Sub

ErrHandler:
   FunctionComp& = False
   Resume Next
End Sub

'*************************************************
'***                                           ***
'***    Session Execute Macro                  ***
'***                                           ***
'*************************************************
'
```

```
'   Maximize the Session
'
Sub Command2_Click ()
On Error GoTo ErrHandler
   FunctionComp& = True

   Text1.LinkExecute "[WINDOW(MAXIMIZE)]"

   If FunctionComp&= False Then
      MsgBox "Error has occurred", 48, "DDE sample"
   End If
   Exit Sub

ErrHandler:
   FunctionComp& = False
   Resume Next
End Sub

'*************************************************
'***                                           ***
'***    Set Cursor Position                    ***
'***                                           ***
'*************************************************
'
'   Set Cursor Position (Row,Column) = (1,1)
'
Sub Command2_Click ()
On Error GoTo ErrHandler
   FunctionComp& = True

   Text1.Text     = "R1C1"
   Text1.LinkItem = "SETCURSOR"
   Text1.LinkPoke

   If FunctionComp&= False Then
      MsgBox "Error has occurred", 48, "DDE sample"
   End If
   Exit Sub

ErrHandler:
   FunctionComp& = False
   Resume Next
End Sub

'*************************************************
'***                                           ***
'***    Terminate Session Conversation         ***
'***                                           ***
'*************************************************
'
'  Terminate DDE Conversation with session
'
Sub Command3_Click ()
On Error GoTo ErrHandlerTerm
   Dim NONE As Integer
   NONE = 0
   FunctionComp& = True

   Text1.LinkMode = NONE

   If FunctionComp&= False Then
      MsgBox "Error has occurred", 48, "DDE sample"
   End If
   Exit Sub

ErrHandlerTerm:
   FunctionComp& = False
```

```
   Resume Next
End Sub

'/*********************************************/
'/*                                           */
'/*     Session conversation(Hot Link)        */
'/*                                           */
'/*********************************************/

'*************************************************
'***                                           ***
'***    Start Close Intercept                  ***
'***                                           ***
'*************************************************
'
'  Start Intercepting Close request
'
Sub Command1_Click ()
On Error GoTo ErrHandlerInit
   Dim HOT As Integer
   HOT = 1
   FunctionComp& = True

   DoEvents
   Text1.LinkTopic = "|SessionA"
   Text1.LinkItem  = "CLOSE"
   Text1.LinkMode  = HOT

   If FunctionComp&= False Then
      MsgBox "Error has occurred", 48, "DDE sample"
   End If
   Exit Sub

ErrHandlerInit:
   FunctionComp& = False
   Resume Next
End Sub

'-- note --------------------------------------
'
' If you use VisualBasic Version2.0, use "DoEvents"
' function before starting DDE conversation by
' calling "LinkMode" function.
'
'----------------------------------------------
'*************************************************
'***                                           ***
'***    Start Keystroke Intercept              ***
'***                                           ***
'*************************************************
'
'  Start Intercepting Keystrokes
'
Sub Command1_Click ()
On Error GoTo ErrHandlerInit
   Dim HOT As Integer
   HOT = 1
   FunctionComp& = True

   DoEvents
   Text1.LinkTopic = "|SessionA"
   Text1.LinkItem  = "KEYS"
   Text1.LinkMode  = HOT

   If FunctionComp&= False Then
      MsgBox "Error has occurred", 48, "DDE sample"
   End If
```

```
   Exit Sub

ErrHandlerInit:
   FunctionComp& = False
   Resume Next
End Sub

'-- note --------------------------------------------
'
' If you use VisualBasic Version2.0, use "DoEvents"
' function before starting DDE conversation by
' calling "LinkMode" function.
'
'-------------------------------------------------
'*************************************************
'***                                           ***
'***    Start Session Advise(PS)               ***
'***                                           ***
'*************************************************
'
'  Receives PS Data when updated
'  (only when "Hello!" is displayed from PS Position 1)
'
Sub Command1_Click ()
On Error GoTo ErrHandlerInit
   Dim HOT As Integer
   HOT = 1
   FunctionComp& = True

   DoEvents
   Text1.LinkTopic = "|SessA_PS"
   Text1.LinkItem  = "PS(1,6,1,""Hello!"")"
   Text1.LinkMode  = HOT

   If FunctionComp&= False Then
      MsgBox "Error has occurred", 48, "DDE sample"
   End If

   Exit Sub

ErrHandlerInit:
   FunctionComp& = False
   Resume Next
End Sub

'-- note --------------------------------------------
'
' If you use VisualBasic Version2.0, use "DoEvents"
' function before starting DDE conversation by
' calling "LinkMode" function.
'
'-------------------------------------------------
'*************************************************
'***                                           ***
'***    Start Session Advise(TRIMRECT)         ***
'***                                           ***
'*************************************************
'
'  Receives PS Data in Trim Rectangle when PS Data in Trim Rectangle
'  specified by R1C1:R20C40 is changed
'
Sub Command1_Click ()
On Error GoTo ErrHandlerInit
   Dim HOT As Integer
   HOT = 1
   FunctionComp& = True
```

```
   DoEvents
   Text1.LinkTopic = "|SessA_TRIMRECT"
   Text1.LinkItem  = "TRIMRECT(1,1,20,40)"
   Text1.LinkMode  = HOT

   If FunctionComp&= False Then
      MsgBox "Error has occurred", 48, "DDE sample"
   End If
   Exit Sub

ErrHandlerInit:
   FunctionComp& = False
   Resume Next
End Sub

'-- note --------------------------------------------
'
' If you use VisualBasic Version2.0, use "DoEvents"
' function before starting DDE conversation by
' calling "LinkMode" function.
'
'-----------------------------------------------------
'*****************************************************
'***                                               ***
'***    Start Session Advise(OIA)                  ***
'***                                               ***
'*****************************************************
'
'  Receives OIA Data when changed
'
Sub Command1_Click ()
On Error GoTo ErrHandlerInit
   Dim HOT As Integer
   HOT = 1
   FunctionComp& = True

   DoEvents
   Text1.LinkTopic = "|SessA_OIA"
   Text1.LinkItem  = "OIA"
   Text1.LinkMode  = HOT

   If FunctionComp&= False Then
      MsgBox "Error has occurred", 48, "DDE sample"
   End If
   Exit Sub

ErrHandlerInit:
   FunctionComp& = False
   Resume Next
End Sub

'-- note --------------------------------------------
'
' If you use VisualBasic Version2.0, use "DoEvents"
' function before starting DDE conversation by
' calling "LinkMode" function.
'
'-----------------------------------------------------
'*****************************************************
'***                                               ***
'***    Terminate Session Conversation(Hot Link)   ***
'***                                               ***
'*****************************************************
'
'  Terminate DDE Conversation with session (Hot Link)
'
Sub Command3_Click ()
```

```
On Error GoTo ErrHandlerTerm
   Dim NONE As Integer
   NONE = 0
   FunctionComp& = True

   Text1.LinkMode = NONE

   If FunctionComp&= False Then
      MsgBox "Error has occurred", 48, "DDE sample"
   End If
   Exit Sub

ErrHandlerTerm:
   FunctionComp& = False
   Resume Next
End Sub
```

第 8 章 サーバー/リクエスター・プログラム・インターフェース (SRPI) のサポート

サーバー/リクエスター・プログラミング・インターフェース (SRPI) は、 IBM 拡張接続機構 (ECF) にアクセスするための API で、SRPI リクエスター・プログラムを書くためのツールを提供しています。SRPI は、リモート・サーバー・プログラムへの同期呼び出し/戻りのインターフェースを提供するために、単一の verb の SEND_REQUEST を使用します。

注: SRPI は、パーソナル・コミュニケーションズ iSeries 版では使用できず、iSeries、eServer i5、またはSystem i5 ホストに接続されている場合には作動しません。

PC/3270 SRPI は C または C++ で作成された 32 ビット SRPI リクエスター・プログラムをサポートします。

SRPI の使用方法

C または C++ で SRPI を使用してアプリケーション・プログラムを作成することができます。 SRPI アプリケーションを開発するためには、以下のことを実行してください。

1. ソース・コードを用意して、適切な SRPI 呼び出しを追加してください。
2. アプリケーション・プログラムにヘッダー・ファイル UUCCPRB.H を指定してください。
3. ソース・コードをコンパイルしてください。
4. コンパイルしてできた .OBJ ファイルを、適切なオブジェクト・ファイルまたはライブラリーにリンクしてください。

また、SRPI インポート・ライブラリーの PCSCALLS.LIB (16 ビットの場合) および PCSCAL32.LIB (32 ビットの場合) も、この .OBJ ファイルをリンクしてください。

SRPI の互換性

PC/3270 は、以下のように SRPI 関数をサポートしています。

- SRPI インターフェースは、パーソナル・コミュニケーションズ J3.1 と同じです。
- SRPI インターフェースは、ホストへの物理的な接続がトークンリングまたは同軸ケーブル、あるいは SNA または非 SNA プロトコルを介している場合に、エミュレーターのホスト接続を介して (非同期および制御装置端末 (CUT) 接続を除く) すべてのモードで使用できます。
- SRPI インターフェースに対して呼び出しが行われても、通信障害のために応答がない場合は、関連するエラーが呼び出し側に戻されます。
- SRPI と EHLLAPI は、同時操作が可能です。

- SRPI は、C リクエスター用にのみサポートされます。
- サーバーの別名はサポートされません。
- 3270 画面更新通知はサポートされません。

パーソナル・コミュニケーションズ用 の既存の 16 ビット SRPI アプリケーションをサポートするために、PCSSRPI.DLL が提供されています。PCSSRPI.DLL は、16 ビット・アドレッシングを 32 ビット・アドレッシングに変換し、それを PC/3270 SRPI DLL に渡します。

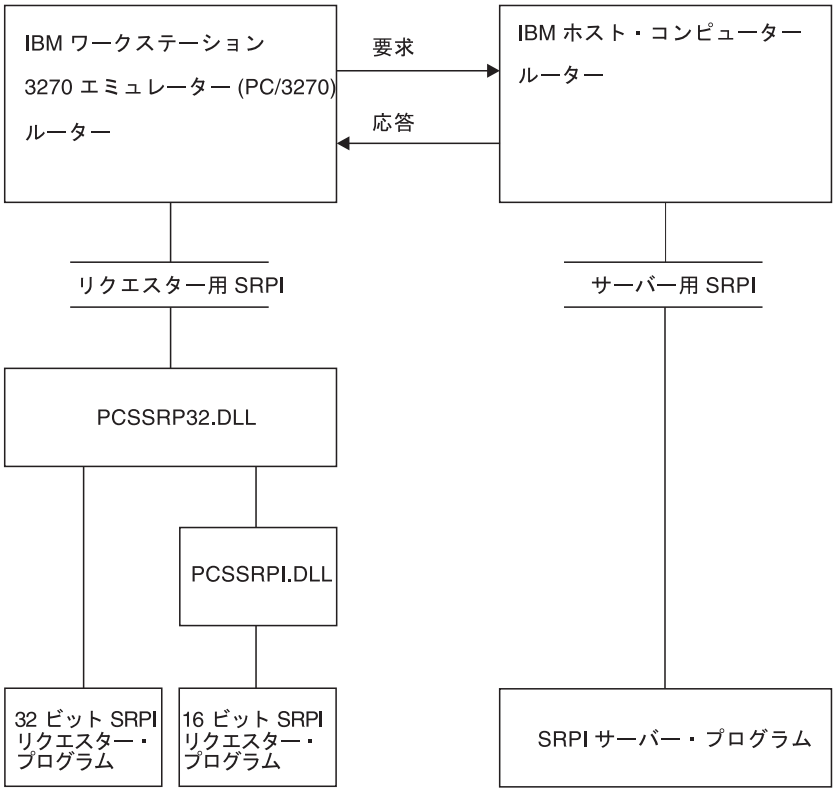

図 5. PC/3270 SRPI リクエスターとサーバーの例

サーバー/リクエスター・プログラム・インターフェースの使用

ワークステーションからの SRPI リクエスターとホスト・コンピューターのサーバーとの間の API を、サーバー/リクエスター・プログラミング・インターフェース (SRPI) と呼びます。

注: これに対応する IBM ホスト・コンピューターのサーバー用インターフェースの説明は、以下の資料を参照してください。

- *TSO/E Version 2 Guide to the Server - Requester Programming Interface*
- *IBM Programmer's Guide to the Server - Requester Programming Interface for VM/System Product*

ワークステーション上で SRPI を使用すると、SRPI リクエスターだけがサポートされます。 SRPI は、アプリケーション間通信の呼び出し/戻り関数を提供します。ワークステーション上のプログラムは、**SEND_REQUEST** 関数を使用してホスト・コンピューター上のパートナー・プログラムのサービスを呼び出します (要求)。このパートナー・プログラムは結果を戻します (サービス)。

ワークステーションとホスト・コンピューターとの関係については、図 6を参照してください。

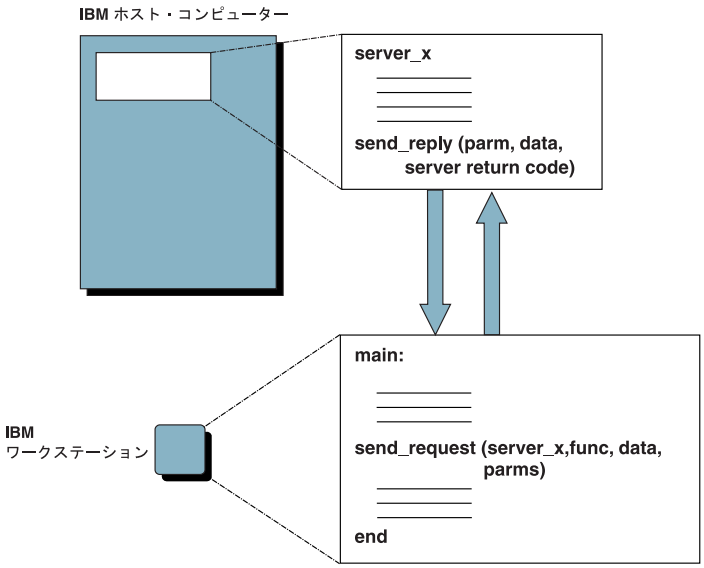

図 6. IBM ワークステーション・リクエスターと IBM ホスト・コンピューター・サーバーの関係

アプリケーションは、**SEND_REQUEST** verb を発行することによって SRPI を使用します。

ワークステーションの SRPI リクエスターが SRPI を使用して **SEND_REQUEST** verb を発行すると、次の処理が行われます。

1. SRPI ルーターが、その要求をホスト・コンピューター・ルーターが認識できる構造に変換する。
2. SRPI ルーターが、適切な 3270 端末エミュレーション・セッションを使用して、要求をホスト・コンピューターのルーターに渡す。
3. ホスト・コンピューターのルーターが、その要求を、適切なホスト・コンピューター・サーバーに渡す。
4. ホスト・コンピューター・サーバーが要求を処理し、 ホスト・コンピューター・ルーターに応答を渡す。
5. ホスト・コンピューター・ルーターが、応答を SRPI ルーターに返す。
6. SRPI ルーターは、応答を変換して、要求元の SRPI リクエスター・アプリケーションに返す。 図 7 は、 リクエスターとサーバー間のフローを図示しています。

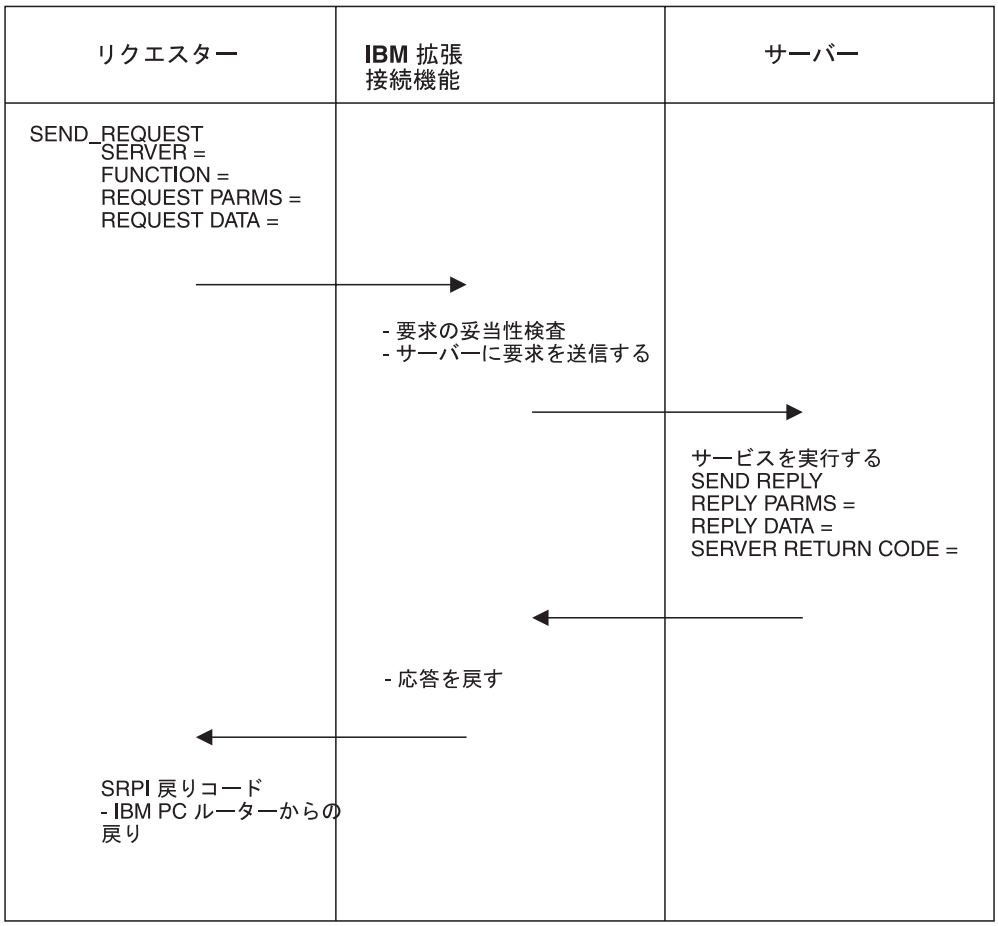

図 7. SRPI リクエスターとサーバーの間のフローの例

SEND_REQUEST パラメーター

SRPI ルーターは、3270 端末エミュレーションが提供する通信機能を使用して、ホスト・コンピューターに要求を送信します。 SRPI は、該当する戻りコード、オプション・パラメーター、およびオプション・データとともに、 SRPI リクエスターに制御を戻します。

SEND_REQUEST 関数に関連するパラメーターおよびデータは、 360 ページの表 23 および 362 ページの表 24 に説明があります。

指定パラメーター

表 23. SRPI リクエスターで指定するパラメーター

パラメーター名	必須/オプション	デフォルト値	説明
関数 ID	オプション	0	要求するサーバー関数を指定する、2 バイトの符号なしの 2 進数。 SRPI リクエスターによる指定では、0〜65535 の値が有効。

表 23. SRPI リクエスターで指定するパラメーター (続き)

パラメーター名	必須/オプション	デフォルト値	説明
応答データ・バッファーの長さ	オプション	0	SRPI リクエスターで指定する応答データ・バッファーの長さをバイト数で指定する、2 バイトの符号なしの 2 進数。 0〜65535 の値が有効。値 0 は、サーバーからの応答データを予期していないことを示す。
応答データ・バッファー・ポインター	オプション	0	応答データ・バッファーの 4 バイトのアドレス。応答データ・バッファーの長さが 0 でなければ、受信する応答データが存在することを示す。
応答パラメーター・バッファーの長さ	オプション	0	SRPI リクエスターで指定する応答パラメーター・バッファーの長さをバイト数で指定する、2 バイトの符号なしの 2 進数。 0〜32763 の値が有効。値 0 は、サーバーからの応答パラメーターを予期していないことを示す。
応答パラメーター・バッファー・ポインター	オプション	0	応答パラメーター・バッファーの 4 バイトのアドレス。応答パラメーター・バッファーの長さの値が 0 でない場合、このバッファーが存在することを示す。
要求データの長さ	オプション	0	サーバーに渡す要求データの長さをバイト数で指定する、2 バイトの符号なしの 2 進数。 0〜65535 の値が有効。値 0 は、渡す要求データがないことを示す。
要求データ・ポインター	オプション	0	サーバーに渡すデータの 4 バイトのアドレス (渡すデータが存在する場合)。要求データの長さが 0 でなければ、渡すデータが存在することを示す。
要求パラメーターの長さ	オプション	0	サーバーに渡す要求パラメーターの長さをバイト数で指定する、2 バイトの符号なしの 2 進数。 0〜32763 の値が有効。値 0 は、渡す要求パラメーターがないことを示す。

表 23. SRPI リクエスターで指定するパラメーター (続き)

パラメーター名	必須/オプション	デフォルト値	説明
要求パラメーター・ポインター	オプション	0	サーバーに渡すパラメーターの 4 バイトのアドレス (渡すパラメーターが存在する場合)。要求パラメーターの長さが 0 でなければ、渡すパラメーターが存在することを示す。
サーバー名	必須	ブランク	ホスト・コンピューター・サーバー名は、長さ 8 バイト (PC/ASCII)、左そろえで、ブランク (X'20') で余白を埋めてあること、先頭または途中にブランクがあったり、全桁がブランクであったりするサーバー名は、無効。有効な PC/ASCII 文字は *A*～*Z* (大文字および小文字)、0～9、$、#、および @。サーバー名は、要求がホスト・コンピューターに送信される前に、EBCDIC に変換される。

戻りパラメーター

表 24. SRPI リクエスターに戻されるパラメーター

パラメーター名	説明
SRPI 戻りコード	**SEND_REQUEST** の実行結果を指定する 4 バイトの値。SRPI 戻りコードの詳細については、付録 D を参照してください。
サーバー戻りコード	サーバーから戻される 4 バイトの値。戻り状況の内容と意味はリクエスターまたはサーバーによって定義されるが、フィールドの長さは常に 32 ビット。
応答パラメーターの長さ	サーバーから戻されるパラメーターの数をバイト数で指定する、 2 バイトの符号なしの 2 進数のメモリー位置。 0～32763 の値が有効。値 0 は、サーバーから戻りパラメーターを受信しなかったことを示す。
応答データの長さ	サーバーから戻されるデータの長さをバイト数で指定する、2 バイトの符号なしの 2 進数のメモリー位置。 0～65535 の値が有効。値 0 は、サーバーから応答データを受信しなかったことを示す。

注:

1. デフォルト値は、対応する要求レコードの初期化関数を使用して設定できます。
2. **SEND_REQUEST** を 3270 セッションにルーティングしてホスト・サーバーを呼び出すには、サーバー名を使用します。
3. 接続プログラミング要求ブロック (CPRB) にあるアドレスが指示するアプリケーションおよびパラメーターの内容と意味は、 SRPI リクエスターおよびサーバーによって決まります。

PC/3270 で SRPI を使用する方法

PC/3270 で実行するローカル・アプリケーションでは、接続されたリモート・コンピューターのアプリケーションに対して **SEND_REQUEST** verb を発行できます。ローカル・アプリケーションが SRPI リクエスターであり、リモート・アプリケーションはサーバーに相当します。関数 ID を指定することにより、SRPI リクエスターでサーバー固有の関数を識別できます。

コンタクトが正常である場合、リモート・アプリケーションのサービスを SRPI リクエスターで使用できます。以下に、**SEND_REQUEST** 関数の呼び出し方法および使用方法を説明します。

SEND_REQUEST の呼び出し

アプリケーションで、**SEND_REQUEST** を呼び出す場合、プログラムから見ると、メイン・ルーチン (ローカル・アプリケーション) がサブルーチン (リモート・アプリケーション) を呼び出していることになります。リクエスター・アプリケーションを作成するプログラマーは、以下のことを実行しなければなりません。

1. 接続プログラミング要求ブロック (CPRB) 用のストレージを確保する。
2. CPRB の初期化。これには、デフォルト値の設定およびアプリケーション・パラメーターの値の指定が含まれます。

   PC/3270 は、サポートされる各言語用の初期化ルーチンおよびマクロを提供します。これらの初期化関数があるので、アプリケーションで CPRB マッピングおよび呼び出しメカニズムを考慮する必要はありません。
3. **SEND_REQUEST** を発行して、ダイナミック・リンク・ライブラリー (DLL) を呼び出す。
4. CPRB で受け取った SRPI 戻りコードの妥当性を検査する。

SEND_REQUEST 関数は、DLL として使用されます。

パフォーマンスに関する考慮事項

SRPI ルーターがホスト・コンピューターとのデータ交換に使用するデータ転送バッファーのサイズは、PC/3270 が自動的に計算します。 SRPI リクエスターで、サーバーとの間で大きなサイズのデータ・ブロックをやりとりする要求を作成する場合、PC/3270 が自動的に計算したバッファー・サイズを指定変更したほうがパフォーマンスが向上する場合があります。そのためには、SRPI に使用する 3270 論理表示端末の定義を変更します。

3270 論理表示端末ウィンドウの作成/変更で指定されるデータ転送バッファー・サイズ指定変更パラメーターを使用して、SRPI で使用するバッファー・サイズを変更します。値 0 は、PC/3270 がバッファー・サイズを計算することを示します。 0 以外の値 (1〜32) は、1024 バイトの倍数で、バッファー・サイズを指定します。大きな値 (30 など) を指定すると、SRPI のパフォーマンスは向上してもシステム全体のパフォーマンスが低下する場合があるので注意してください。データ転送バッファー・サイズ指定変更パラメーターが、ファイル転送機能で使用するデータ転送バッファーのサイズも設定することに注意してください。

中断 (Ctrl+Break) キーの取り扱い

SEND_REQUEST verb の処理中は、すべての信号 (数値演算共用プロセッサー信号以外) が、verb の処理完了まで遅らされます。特に、中断 (Ctrl+Break) キーを押しても、 **SEND_REQUEST** verb 実行中はプログラムを取り消せません。

C リクエスター

このセクションでは、C 言語でリクエスターを作成するプログラマーを対象にしています。次の事項について説明します。

- C の send_request 関数
- SRPI レコード定義
- **Send_request** 関数の定義
- SRPI 戻りコード

サンプル・プログラムは、パーソナル・コミュニケーションズをインストールされているお客様に提供されています。

注: 他のセクションで **SEND_REQUEST** と呼んでいる関数を、このセクションでは、C 言語の規則にしたがって **send_request** と表記します。

C の send_request 関数

send_request パラメーターは、タイプ UERCPRB の単一の C 構造体にまとめられます。 **init_send_req_parms** 関数は、UERCPRB 構造体の中のすべての **send_request** パラメーターをデフォルト値に初期化するために提供されています。これにより、リクエスターで使用しないパラメーターをいったんデフォルト値に設定することができます。**send_request** 関数は、サーバー・プログラムに対して同期呼び出しを行うためのものです。

init_send_req_parms 関数および **send_request** 関数は、使用する C アプリケーションとリンクさせる必要があります。32 ビット・インターフェースの場合は PCSSRP32.DLL、16 ビット・インターフェースの場合は PCSSRPI.DLL です。 2 つのオブジェクト・ファイルはどちらも PC/3270 で提供しています。

send_request 関数によっては、UERCPRB 構造体の内容を接続プログラミング要求ブロック (CPRB) にコピーし、PCSSRP32.DLL を呼び出します。サーバーの処理が完了すると、 **send_request** プロシージャーは、CPRB から戻されたパラメーターを UERCPRB 構造体にコピーし、C アプリケーションに制御を戻します。

要求パラメーターまたはデータが複数の構造体で構成されている場合、アプリケーションは、データまたはパラメーターを変換して、バッファーに保管される連続したバイトで構成される単一フラット構造体にしなければなりません。要求元プログラムでは、要求パラメーターおよびデータを、サーバーが識別できる形式にまとめる必要があります。

UERCPRB は、パック構造です。つまり、先頭の構造メンバーに続くそれぞれの構造メンバーは、使用可能な最初のバイトに保管されます。

要求パラメーターに使用するメモリーは、応答パラメーターにも使用できます。要求データに使用するメモリーは、応答データにも使用できます。アプリケーション・プログラム作成の際には、要求データおよび要求パラメーターが不要になった場合のみ、要求データと要求パラメーター用のバッファーに、応答データと応答パラメーターを書き込むようにしてください。

SRPI レコード定義

UERCPRB レコード・タイプは **send_request** 関数を使用して SRPI ルーターに渡すレコードを定義します。レコードは、UUCCPRB.H ファイルを含むように、`#include` プリプロセッサー・ディレクティブを使用して、アプリケーション・プログラム内に定義されます。指定パラメーターと戻りパラメーターの定義および値の範囲については、360 ページの『指定パラメーター』および 362 ページの『戻りパラメーター』を参照してください。

SRPI 戻りコード

SRPI の戻りコードについては、387 ページの『付録 D. SRPI 戻りコード』を参照してください。

付録 A. EHLLAPI がサポートする Query Reply データ構造

この付録では、EHLLAPI 構造化フィールド・インターフェースが PC/3270 用にサポートする Query Reply 構造を列挙し、定義します。「*IBM 3270 情報表示システム データストリーム プログラマー用解説書*」を参照してください。また、IBM ライセンス・プログラムの場合は、その特定のライセンス・プログラム用の資料を参照してください。

注:

1. EHLLAPI は、Query Reply バッファーを走査し、Destination (宛先)/Origin (発信元) ID (DOID) の自己定義パラメーター (SDP) を見つけることにより、構造化フィールド・サポートが確実に機能するようにします。これにより、DOID フィールドが割り当て ID で埋め込まれます。
2. アプリケーションは、Query Reply データ構造をアプリケーションの専用メモリー内に作成する必要があります。
3. Query Reply のデータについて簡単なチェックが実行されます。 ID と構造の長さについてのみ有効かどうかチェックされます。
4. 各 Query Reply の先頭の長さ 2 バイトのフィールドは、**バイト反転していません**。
5. ホスト・セッションごとに許可される分散データ管理 (DDM) の基本タイプは 1 つだけです。DDM 接続が DOID 用に SDP をサポートしている場合は、複数の接続が可能です。
6. アプリケーションが選択済みのセッションとすでに接続していることを示すゼロ以外の戻りコードを受信した (RC 32 または 39) 場合、表示スペースを使用する際には注意が必要です。 SRPI、ファイル転送、および他の EHLLAPI アプリケーションとの競合が生じる可能性があります。

DDM Query Reply

何種類かの DDM Query Reply 形式がサポートされます。形式の一部を次に示します。

表 25. DDM Query Reply の基本形式

オフセット	長さ	内容	意味
0	1 ワード	長さ	構造体の長さ
2	1 バイト	X'81'	Query reply ID
3	1 バイト	X'95'	Query reply のタイプ
4 ～ 5	2 バイト	FLAGS	予約済み
6 ～ 7	2 バイト	LIMIN	インバウンド伝送できる DDM バイトの最大数
8 ～ 9	2 バイト	LIMOUT	アウトバウンド伝送できる DDM バイトの最大数

表 25. DDM Query Reply の基本形式 (続き)

オフセット	長さ	内容	意味
10	1 バイト	NSS	サブセットの ID 数
11	1 バイト	DDMSS	DDM サブセットの ID

DDM アプリケーション名の自己定義パラメーター

DDM アプリケーション名の自己定義パラメーターは、DDM 補助装置の制御を含むアプリケーションの名前をホスト・アプリケーションに提供します。アプリケーションの制御は、直接アクセス自己定義パラメーター内の DOID で識別されます。

自己定義パラメーター (SDP) は任意指定ですが、複数のアプリケーションがリモート・ワークステーションに存在し、ホスト・アプリケーションが DDM 補助装置を明白に識別する必要がある場合には必須です。

表 26. DDM アプリケーション名の自己定義パラメーター

オフセット	長さ	内容	意味
0	1 バイト	長さ	パラメーター長
1	1 バイト	X'02'	DDM アプリケーション名
2 〜 n	n-2 バイト	NAME	リモート・アプリケーション・プログラムの名前

NAME この名前は 8 文字以下で構成され、この名前によってホスト・アプリケーションとリモート・ワークステーションのアプリケーションを対応付けることができます。ホストとリモートのアプリケーション・ユーザーは、その名前が相手側のアプリケーションに認識されていることを確認する必要があります。

PCLK プロトコル制御の自己定義パラメーター

PCLK プロトコル制御の自己定義パラメーターは、PCLK プロトコル制御の構造化フィールド ID = X'1013' が、DDM 補助装置プロセッサーの宛先になったり発信元になったりするインバウンドとアウトバウンドの両方のデータ・ストリームで使用できることを示します。

表 27. DDM PCLK 補助装置 SDP

オフセット	長さ	内容	意味
0	1 バイト	X'04'	パラメーター長
1	1 バイト	X'03'	PCLK プロトコル制御
2 〜 3	2 バイト	VERS	プロトコルのバージョン

VERS VERS に指定された値は、Query Reply が戻されるときに端末にイ

ンストールされる PCLK のバージョンを示すために使用されます。例えば、X'0001' は PCLK バージョン 1.1 を示します。

この Query Reply に対するフィールド定義については、「*IBM 3270 情報表示 システム データストリーム プログラマー用解説書*」を参照してください。

基本 DDM Query Reply の形式

次の Query Reply の形式は、基本 (BASE) + SDP の組み合わせの一部の*例* です。すべての組み合わせを示しているわけではありません。

表 28. 名前および直接アクセス SDP での基本 DDM Query Reply の形式

オフセット	長さ	内容	意味
0	1 ワード	長さ	構造体の長さ (SDP を含む)
2	1 バイト	X'81'	Query reply ID
3	1 バイト	X'95'	Query Reply のタイプ
4 〜 5	2 バイト	FLAGS	予約済み
6 〜 7	2 バイト	LIMIN	インバウンド伝送できる DDM バイトの最大数
8 〜 9	2 バイト	LIMOUT	アウトバウンド伝送できる DDM バイトの最大数
10	1 バイト	NSS	サポートされるサブセットの数
11	1 バイト	DDMSS	DDM サブセットの ID
12	1 バイト	長さ (n+2)	パラメーター長
13	1 バイト	X'02'	DDM アプリケーション名
14 〜 (13+n)	n バイト	名前	リモート・アプリケーション・プログラムの名前
14+n	1 バイト	X'04'	パラメーター長
15+n	1 バイト	X'01'	直接アクセス ID
16+n - 17+n	2 バイト	DOID	サブシステムが割り当てた Destination/Origin ID

自己定義パラメーター (SDP) はオフセット 12 および (14 + *n*) で始まります。ここで、*n* はオフセット 14 で指定されるアプリケーションの名前の長さです。

この Query Reply に対するフィールド定義については、「*IBM 3270 情報表示 システム データストリーム プログラマー用解説書*」を参照してください。

表 29. 直接アクセスおよび名前 SDP での基本 DDM Query Reply の形式

オフセット	長さ	内容	意味
0	1 ワード	長さ	構造体の長さ (SDP を含む)
2	1 バイト	X'81'	Query reply ID
3	1 バイト	X'95'	Query reply のタイプ
4 ～ 5	2 バイト	FLAGS	予約済み
6 ～ 7	2 バイト	LIMIN	インバウンド伝送できる DDM バイトの最大数
8 ～ 9	2 バイト	LIMOUT	アウトバウンド伝送できる DDM バイトの最大数
10	1 バイト	NSS	サポートされるサブセットの数
11	1 バイト	DDMSS	DDM サブセットの ID
12	1 バイト	X'04'	パラメーター長
13	1 バイト	X'01'	直接アクセス ID
14 ～ 15	2 バイト	DOID	サブシステムが割り当てた Destination/Origin ID
16	1 バイト	長さ (n+2)	パラメーター長
17	1 バイト	X'02'	DDM アプリケーション名
16+*n* ～ 17+*n*	*n* バイト	名前	リモート・アプリケーション・プログラムの名前

自己定義パラメーター (SDP) はオフセット 12 および 16 で始まります。

この Query Reply に対するフィールド定義については、「*IBM 3270 情報表示 システム データストリーム プログラマー用解説書*」を参照してください。

IBM 補助装置 Query Reply

補助装置 Query Reply は、IBM 定義のデータ・ストリームを使用する IBM 補助装置のサポートをホスト・アプリケーションに示します。詳しくは、「*IBM 3270 情報表示システム データストリーム プログラマー用解説書*」を参照してください。

機能がサポートされると、Query Reply は Query または Query List (QCODE リスト = X'9E'、等価のもの、またはすべて) を指定する Read Partition 構造化フィールドに応答してインバウンド伝送されます。

ワークステーションが複数の補助装置をサポートしている場合、IBM 補助装置 Query Reply は各装置用に送信されます。

任意指定パラメーター

Query Reply の基本部分に示されるパラメーターはすべて必須です。使用しないパラメーターは X'00' に設定されます。

自己定義パラメーターは最低 1 つ必要です。

表 30. 直接アクセス SDP での IBM 補助装置の基本形式

オフセット	長さ	内容	意味
0 〜 1	1 ワード	長さ	構造体の長さ (SDP を含む)
2	1 バイト	X'81'	Query reply ID
3	1 バイト	X'9E'	IBM 補助装置 Reply
4	1 バイト ビット 0 1 〜 7	FLAGS QUERY B'1' RES	予約済み 読み取り部分 (Query、Query List) IBM 補助装置が Query をサポートする 予約済み、必ず B'0’
5	1 バイト	FLAGS	予約済み
6 〜 7	2 バイト	LIMIN	インバウンド伝送できる DDM バイトの最大数
8 〜 9	2 バイト	LIMOUT	アウトバウンド伝送できる DDM バイトの最大数
10	1 バイト	TYPE X'01' X'02' その他	サポートされる補助装置のタイプ IBM 補助装置ディスプレイ IBM 補助装置プリンター 予約済み
11	1 バイト	X'04'	パラメーター長
12	1 バイト	X'01'	直接アクセス
13 〜 14	1 ワード	DOID	サブシステムが割り当てた Destination/Origin ID

QUERY IBM 補助装置が Read Partition (Query、Query List) の受信をサポートしていることを示すには、このビットを B'1' に設定する必要があります。これにより、ホスト・アプリケーションは補助装置にあてた Read Partition を使用して、特性を判別できます。 Destination/Origin 構造化フィールドは、Read Partition 構造化フィールドを補助装置にあてるのに使用されます。

IBM 補助装置に対するサポートを最小レベルにすると、Read Partition に応答して NULL の Query Reply を戻します。

LIMIN インバウンド伝送で送信できる最大バイト数を示します。 LIMIN の値が X'0000' の場合は、インバウンド伝送されるバイト数に対して使用制限がないことを示しています。

LIMOUT アウトバウンド伝送で IBM 補助装置に送信できる最大バイト数を示します。 LIMOUT の値が X'0000' の場合は、アウトバウンド伝送されるバイト数に対して使用制限がないことを示します。

TYPE サポートされる補助装置を識別します。 2 つの値が有効です。 1 つは補助ディスプレイを識別し、もう 1 つは補助プリンターを識別します。その他の値は予約済みです。

IBM 補助装置プロセッサーは 2 つの自己定義パラメーター (01 および 03) をサポートします。これらのパラメーターは表 31 で定義されています。

直接アクセス自己定義パラメーター

直接アクセス自己定義パラメーターは、IBM 補助装置の直接アクセスで Destination/Origin 構造化フィールドに使用する ID を提供します。

この SDP は、基本 Query Reply とともに必ず必要です。

表 31. IBM 補助装置直接アクセス SDP

オフセット	長さ	内容	意味
0	1 バイト	X'04'	パラメーター長
1	1 バイト	X'01'	直接アクセス ID
2 ～ 3	2 バイト	DOID	Destination/Origin ID

DOID これらのバイト内の値は、Destination/Origin 構造化フィールドの ID フィールド内で使用され、補助装置が後に続くデータの宛先かまたは発信元かを識別します。

PCLK プロトコル制御の自己定義パラメーター

PCLK プロトコル制御の自己定義パラメーターがある場合、PCLK プロトコル制御の構造化フィールド ID = X'1013' が、IBM 補助装置プロセッサーの宛先になったり発信元になったりするインバウンドとアウトバウンドの両方のデータ・ストリームで使用できることを示します。

表 32. IBM 補助装置 PCLK SDP

オフセット	長さ	内容	意味
0	1 バイト	X'04'	パラメーター長
1	1 バイト	X'03'	PCLK プロトコル制御
2 ～ 3	2 バイト	VERS	プロトコルのバージョン

VERS VERS に指定された値は、Query Reply が戻されるときに端末にインストールされる PCLK のバージョンを示すために使用されます。例えば、X'0001' は PCLK バージョン 1.1 を示します。

この Query Reply に対するフィールド定義については、「*IBM 3270 情報表示 システム データストリーム プログラマー用解説書*」を参照してください。

OEM 補助装置 Query Reply

OEM 補助装置の Query Reply 形式は次のとおりです。

表 33. 直接アクセス SDP での OEM 補助装置の基本形式

オフセット	長さ	内容	意味
0 ～ 1	1 ワード	長さ	構造体の長さ (SDP を含む)

表 33. 直接アクセス SDP での OEM 補助装置の基本形式 (続き)

オフセット	長さ	内容	意味
2	1 バイト	X'81'	Query reply ID
3	1 バイト	X'8F'	OEM Query Reply
4 ～ 5	2 バイト	FLAGS	予約済み
6 ～ 13	4 ワード	DTYPE	装置のタイプ
14 ～ 21	4 ワード	UNAME	ユーザーが割り当てる名前
22	1 バイト	X'04'	パラメーター長
23	1 バイト	X'01'	直接アクセス
24 ～ 25	1 ワード	DOID	サブシステムが割り当てた Destination/Origin ID

この Query Reply に対するフィールド定義については、「*IBM 3270 情報表示 システム データストリーム プログラマー用解説書*」を参照してください。

OEM 補助装置プロセッサーは 2 つの自己定義パラメーター (01 および 03) をサポートします。これらのパラメーターは表 34 で定義されています。

直接アクセス自己定義パラメーター

直接アクセス自己定義パラメーターは、OEM 補助装置の直接アクセスで Destination/Origin 構造化フィールドに使用する ID を提供します。

表 34. OEM 補助装置直接アクセス SDP

オフセット	長さ	内容	意味
0	1 バイト	X'04'	パラメーター長
1	1 バイト	X'01'	直接アクセス ID
2 ～ 3	2 バイト	DOID	Destination/Origin ID

DOID これらのバイト内の値は、Destination/Origin 構造化フィールドの ID フィールド内で使用され、補助装置が後に続くデータの宛先かまたは発信元かを識別します。

PCLK プロトコル制御の自己定義パラメーター

PCLK プロトコル制御の自己定義パラメーターがある場合、PCLK プロトコル制御の構造化フィールド ID = X'1013' が、OEM 補助装置プロセッサーの宛先になったり発信元になったりするインバウンドとアウトバウンドの両方のデータ・ストリームで使用できることを示します。

表 35. IBM 補助装置 PCLK SDP

オフセット	長さ	内容	意味
0	1 バイト	X'04'	パラメーター長
1	1 バイト	X'03'	PCLK プロトコル制御

表 35. IBM 補助装置 PCLK SDP (続き)

オフセット	長さ	内容	意味
2 〜 3	2 バイト	VERS	プロトコルのバージョン

VERS VERS に指定された値は、Query Reply が戻されるときに端末にインストールされる PCLK のバージョンを示すために使用されます。例えば、X'0001' は PCLK バージョン 1.1 を示します。

連携処理リクエスター Query Reply

連携処理リクエスター Query Reply は、SRPI Query Reply または CPSI Query Reply とも呼ばれています。その形式は次のとおりです。

表 36. CPR Query Reply バッファーの形式

オフセット	長さ	内容	意味
0	1 ワード	長さ	構造体の長さ (SDP を含む)
2	1 バイト	X'81'	Query reply ID
3	1 バイト	X'AB'	Query reply のタイプ
4 〜 5	2 バイト	FLAGS	予約済み
6 〜 7	2 バイト	LIMIN	インバウンド伝送できる DDM バイトの最大数
8 〜 9	2 バイト	LIMOUT	アウトバウンド伝送できる DDM バイトの最大数
10	1 バイト	FETAL	後に続く機能情報の長さ (バイト単位)
11 〜 12	1 ワード	FEATS	CPR 長および機能フラグ
13 〜 (N*2)+12	0 〜 2 バイト	FEATS	追加のフラグ
(N*2)+12	1 バイト	X'04'	DOID SDP の長さ
(N*2)+13	1 バイト	X'01'	D/O ID のタイプ
(N*2)+14	1 ワード	DOID	サブシステムが割り当てた Destination/Origin ID

この Query Reply に対するフィールド定義については、「*IBM 3270 情報表示 システム データストリーム プログラマー用解説書*」を参照してください。

プロダクト定義 Query Reply

この Query Reply は X'9C' データ構造体内で登録済みのサブ ID を使用している IBM プロダクトで使用します。プロダクト定義データ・ストリーム Query Reply は、IBM プロダクト定義のデータ・ストリームを使用する 3270DS ワークステーション補助装置のサポートを示します。データ・ストリームは、アーキテクチャー・レビュー・ボードなどの、識別可能な制御点を持つ形式アーキテクチャーの資料では定義されていません。

補助装置が IBM プロダクト定義のデータ・ストリームをサポートすると、この Query Reply は Query List (QCODE リスト = X'9C' またはすべて) に応答してインバウンド伝送されます。

任意指定パラメーター

Query Reply の基本部分のすべてのパラメーター、および直接アクセス自己定義パラメーターは必須です。

プロダクト定義 Query Reply の形式は次のとおりです。

表 37. IBM プロダクト定義 Query Reply の基本形式

オフセット	長さ	内容	意味
0 〜 1	1 ワード	長さ	構造体の長さ (SDP を含む)
2	1 バイト	X'81'	Query reply ID
3	1 バイト	X'9C'	IBM プロダクト定義データ・ストリーム
4 〜 5	2 バイト	FLAGS	予約済み
6	1 バイト	REFID	参照 ID
7	1 バイト	SSID	サブセット ID
8	1 バイト	X'04'	パラメーター長
9	1 バイト	X'01'	直接アクセス
10 〜 11	1 ワード	DOID	サブシステムが割り当てた Destination/Origin ID

プロダクト定義 Query Reply の REFID (オフセット 6) および SSID (オフセット 7) に対する有効値は次のとおりです。

表 38. IBM プロダクト定義 Query Reply に対する REFID と SSID の有効値

REFID	SSID	プロダクトおよびデータ・ストリームの情報
X'01'		5080 グラフィックス・システム: この参照 ID は、5080 グラフィックス・システム・データ・ストリームを補助装置がサポートしていることを示します。5080 グラフィックス・アーキテクチャー、構造化フィールド、サブセット ID、DOID、および関連機能セットの記述は、「*IBM 5080 Graphics System Principles of Operation*」に定義されています。
	X'01' X'02'	5080 HGFD グラフィックス・サブセット 5080 RS232 ポート・サブセット
X'02'		WHIP API (書き込み時に SRL 名に置き換えられる) この参照 ID は、WHIP API データ・ストリームを補助装置がサポートしていることを示します。WHIP API アーキテクチャーの記述は、「*IBM RT PC Workstation Host Interface Program Version 1.1 User's Guide and Reference Manual*」に定義されています。
	X'01'	WHIP サブセット 1
X'03' ～ X'FF'		その他の値は予約済みです。

IBM プロダクト定義プロセッサーは、直接アクセス自己定義パラメーターのみをサポートします。このパラメーターは表 39 で定義されています。

直接アクセス自己定義パラメーター

直接アクセス ID 自己定義パラメーターがある場合、補助装置が Destination/Origin 構造化フィールドを使用して直接アクセスされている可能性があることを示します。プロダクト定義のデータ・ストリームを使用する複数の補助装置がサポートされている場合、固有の DOID を持つ、個別のプロダクト定義のデータ・ストリーム Query Reply が提供されます。

表 39. IBM プロダクト定義直接アクセス SDP

オフセット	長さ	内容	意味
0	1 バイト	X'04'	パラメーター長
1	1 バイト	X'01'	直接アクセス ID
2 ～ 3	2 バイト	DOID	Destination/Origin ID

DOID これらのバイト内の値は、Destination/Origin 構造化フィールドの ID フィールド内で使用され、補助装置が後に続くデータの宛先かまたは発信元かを識別します。

文書交換アーキテクチャー Query Reply

この Query Reply は、サポートされている文書交換アーキテクチャー (DIA) 機能セットを示します。 DIA Query Reply の形式は次のとおりです。

表 40. IBM DIA の基本形式

オフセット	長さ	内容	意味
0	1 ワード	長さ	構造体の長さ (SDP を含む)
2	1 バイト	X'81'	Query reply ID
3	1 バイト	X'97'	IBM DIA
4 〜 5	2 バイト	FLAGS	予約済み
6 〜 7	2 バイト	LIMIN	インバウンド伝送できる DDM バイトの最大数
8 〜 9	2 バイト	LIMOUT	アウトバウンド伝送できる DDM バイトの最大数
10	1 バイト	NFS	後に続く 3 バイトの機能セット ID の数
11 〜 13	3 バイト	DIAFS	DIA 機能セット ID
14 〜 (13+(N*3))	N*3 バイト	DIAFS	追加の DIA 機能セット ID
14+(N*3)	1 バイト	X'04'	パラメーター長
15+(N*3)	1 バイト	X'01'	直接アクセス
16+(N*3)	1 ワード	DOID	サブシステムが割り当てた Destination/Origin ID

DIA 補助装置プロセッサーは、直接アクセス自己定義パラメーターのみをサポートします。このパラメーターは表 41 で定義されています。

直接アクセス ID 自己定義パラメーターがある場合、補助装置が Destination/Origin 構造化フィールドを使用して直接アクセスされている可能性があることを示します。

表 41. IBM プロダクト定義直接アクセス SDP

オフセット	長さ	内容	意味
0	1 バイト	X'04'	パラメーター長
1	1 バイト	X'01'	直接アクセス ID
2 〜 3	2 バイト	DOID	Destination/Origin ID

DOID これらのバイト内の値は、Destination/Origin 構造化フィールドの ID フィールド内で使用され、補助装置が後に続くデータの宛先かまたは発信元かを識別します。

この Query Reply に対するフィールド定義については、「*IBM 3270 情報表示 システム データストリーム プログラマー用解説書*」を参照してください。

付録 B. コミュニケーション・マネージャー/2 EHLLAPI との相違点

この付録では、パーソナル・コミュニケーションズの EHLLAPI と コミュニケーション・マネージャー/2 用 EHLLAPI との相違点を説明します。

以下の EHLLAPI 関数は、同名のコミュニケーション・マネージャー/2 の関数とは異なっているので、使用するときは十分注意してください。

- **Set Session Parameter** (9)
- **Copy OIA** (13)
- **Copy String to PS** (15)
- **Storage Manager** (17)
- **Copy String to Field** (33)
- **Get Key** (51)
- **Window Status** (104)
- **Query Sessions** (10)
- **Connect for Structured Field** (120)
- **Allocate Communications Buffer** (123)
- ASCII 略号

Set Session Parameter (9)

セット・オプション

パーソナル・コミュニケーションズでは、コミュニケーション・マネージャーが提供している以下のセット・オプションは提供されていません。

OLDOIA、NEWOIA

COMPCASE、COMPICASE

OLD5250OIA、NEW5250OIA

戻りパラメーター

Set Session Parameter (9) 関数が終了すると、コミュニケーション・マネージャーは第 3 パラメーターのデータ・ストリング長として、有効なデータ・ストリングの長さを戻します。しかし、パーソナル・コミュニケーションズでは、データ・ストリング長として、有効なセット・オプションの個数を戻します。

EAB オプション

Set Session Parameter (9) 関数で、EAB オプションが指定されたとき、コミュニケーション・マネージャー/2 では **Copy PS** (5) または **Copy PS to String** (8) 関数でコピーされる EAB 属性の中の文字色の値が、カラー再マップの影響を受けます。

ただし、パーソナル・コミュニケーションズでは、カラー再マップの有無にかかわらず、EAB 属性の文字色の値は、表示スペースの内容によって決まり、カラー再マップの影響を受けません。

Copy OIA (13)

Copy OIA (13) 関数は、コミュニケーション・マネージャー/2 とパーソナル・コミュニケーションズでは、以下の点が異なります。グループと桁位置についての詳しい説明は、57 ページの『Copy OIA (13)』を参照してください。

- 21 バイト目
  - パーソナル・コミュニケーションズは、X'F6' を戻します。
  - コミュニケーション・マネージャー/2 は、X'20' を戻します。
- 61 ～ 63 バイト目
  - パーソナル・コミュニケーションズは、プリンター情報を戻しません。
  - コミュニケーション・マネージャー/2 はプリンター情報を戻します。
- グループ 3: シフト状態

  コミュニケーション・マネージャー/2 は、ビット 2 の値を戻しません。ビット 2 は予約され、ビット 0 に上段シフトと大文字ロック (Caps Lock) の両方がセットされます。
- グループ 8 バイト 1: 入力禁止
  - パーソナル・コミュニケーションズは、ビット 6 の値 (装置作動不能) を戻しません。
  - コミュニケーション・マネージャー/2 は、ビット 6 の値を戻すことがあります。
- グループ 8 バイト 3: 入力禁止
  - パーソナル・コミュニケーションズは、ビット 1 (オペレーター許可なし) およびビット 2 (オペレーター許可なし -f) の値を戻しません。
  - コミュニケーション・マネージャー/2 は、ビット 1 および 2 の値を戻すことがあります。
- グループ 8 バイト 4: 入力禁止
  - パーソナル・コミュニケーションズは、ビット 2 (システム待ち状態) の値を戻しません。
  - コミュニケーション・マネージャー/2 は、ビット 2 の値を戻すことがあります。
- グループ 10: 強調表示グループ 2
  - パーソナル・コミュニケーションズは、ビット 0 の値 (選択済み) を戻しません。
  - コミュニケーション・マネージャー/2 は、ビット 0 の値を戻すことがあります。
- グループ 11: カラー・グループ 2
  - パーソナル・コミュニケーションズは、ビット 0 の値 (選択済み) を戻しません。

- コミュニケーション・マネージャー/2 は、ビット 0 の値を戻すことがあります。

- グループ 13: プリンターの状況
  - パーソナル・コミュニケーションズでは、このグループは予約済みです。
  - コミュニケーション・マネージャー/2 では、このグループを戻すことがあります。
- グループ 14: グラフィックス

  コミュニケーション・マネージャー/2 は、ビット 0 (グラフィックス・カーソル) の値を戻しません。

Copy String to PS (15)

コミュニケーション・マネージャー/2 では、**Set Session Parameter** (9) 関数の EAB オプションが **Copy String to PS** 関数に影響します。 EAB オプションを指定すると、テキスト・データと同じサイズの属性データを、テキスト・データと共にその関数に渡す必要があります。

パーソナル・コミュニケーションズでは、EAB オプションに関係なく、渡されるデータはテキスト・データのみです。コミュニケーション・マネージャー/2 と同様のインターフェースを使用する場合は、**Set Session Parameter** (9) の PUTEAB オプションを使用してください。

Storage Manager (17)

コミュニケーション・マネージャー/2 で提供されている **Storage Manager** (17) 関数は、パーソナル・コミュニケーションズでは、サポートされません。アプリケーション用のメモリーの割り振りには、Windows で提供されている API を使用してください。

Copy String to Field (33)

コミュニケーション・マネージャー/2 では、**Set Session Parameter** (9) 関数の EAB オプションが指定されると、属性データが、その関数への受け渡しのデータの一部として渡されます。そのため、EAB オプションを指定すると、テキスト・データと同じサイズの属性データを、テキスト・データと共にその関数に渡す必要があります。

パーソナル・コミュニケーションズでは、EAB オプションは、**Copy String to Field** (33) 関数のデータ内容には 影響しません。渡すデータはテキスト・データのみで、属性データは渡されません。コミュニケーション・マネージャー/2 と同様のインターフェースを使用する場合は、**Set Session Parameter** (9) の PUTEAB オプションを使用してください。

Get Key (51)

渡されたキーのシフト状態がエミュレーター・セッションで認識されるキーや関数でない場合、コミュニケーション・マネージャー/2 では @A、@S、または @r を使用してシフト状態を戻します。パーソナル・コミュニケーションズでは、これらの ASCII 略号をサポートしていません。

Window Status (104)

EHLLAPI 関数 104 (PM_WINDOW_STATUS) の'拡張状況照会' コマンド (0x03) は、エミュレーター表示スペース・ウィンドウのハンドルを戻します。これは、関数と コミュニケーション・マネージャー/2 のインプリメンテーションの定義によって構成されます。しかし、パーソナル・コミュニケーションズ Windows 版 EHLLAPI では、フレーム・ウィンドウのハンドルが戻されます。この関数を使用して パーソナル・コミュニケーションズ Windows 版用に書かれた、EHLLAPI アプリケーションでは、戻されたウィンドウ・ハンドルの親を使用する必要があります。

Query Sessions (10)

コミュニケーション・マネージャー/2 では、パーソナル・コンピューター用の記述子が戻されます。しかし、パーソナル・コミュニケーションズでは、記述子は戻されません。

Connect for Structured Fields (120)

コミュニケーション・マネージャー/2 で提供されている通信接続状況のためのイベント・オブジェクトは、パーソナル・コミュニケーションズにはありません。

Allocate Communications Buffer (123)

コミュニケーション・マネージャー/2 では、要求バッファー・サイズの最大値が、64 KB - 8 バイト (X'FFF8') です。

パーソナル・コミュニケーションズでは、この値は 64 KB - 256 バイト (X'FF00') です。

ASCII 略号

次の ASCII 略号は、パーソナル・コミュニケーションズではサポートされません。

略号	意味
@A@N	カーソル取り込み
@A@O	カーソル位置づけ
@A@X	16 進数
@A@Y	コマンド (機能) キー
@A@a	消去しながらバックスペース
@S@A	EOL 消去

略号	意味
@S@B	フィールド前進
@S@C	フィールド・バックスペース
@S@D	有効文字バックスペース
@S@P	POR (送信のみ)
@S@T	タスク・マネージャーにジャンプ
@/	キューのオーバーラン (**Get Key** 関数の場合のみ)

Get Request Completion (125)

パーソナル・コミュニケーションズは、ブランクまたは NULL のセッション ID をサポートしません。

付録 C. Windows 用 DOS モード EHLLAPI

パーソナル・コミュニケーションズは、DOS 用の EHLLAPI アプリケーションをサポートしています。この付録では、そのサポートについて説明します。

インストール

パーソナル・コミュニケーションズの DOS EHLLAPI サポートをインストールするには、以下のようにします。

1. 「IBM パーソナル・コミュニケーションズ」フォルダーの「ユーティリティー」フォルダーから「エミュレーター・ユーティリティー」フォルダーを選択する。
2. 「エミュレーター・ユーティリティー」フォルダーから DOS EHLLAPI アプリケーションを選択する。
3. DOS MODE EHLLAPI のチェック・ボックスを選択して、DOS EHLLAPI サポートを有効にする。
4. お手持ちの DOS EHLLAPI アプリケーションを使用できる、メジャー DOS バージョンを入力する。 (例えば、DOS Emulator Version 2.x の場合は 2。)
5. 「了解」を選択して変更を有効にする。
6. ワークステーションを遮断し、再起動する。

この手順により次のステートメントが config.nt に追加されます。

```
device=%SystemRoot%¥system32¥drivers¥hlldrvr.com
```

注: DOS EHLLAPI アプリケーションは、EHLLAPI サービスを要求するために、割り込み X'7F' を挿入します。割り込み X'7F' を使用する他の所有 DOS アプリケーションはいずれも、使用可能な DOS EHLLAPI と一緒には動作しません。その逆も同様です。

付録 D. SRPI 戻りコード

この付録では、SRPI 環境でのエラー処理について説明します。タイプ 0、1、2、および 3 の戻りコードおよびその定義が記載されています。また、例外クラス定義、コード値、およびオブジェクト値も記載されています。サーバー戻りコードについても説明します。

エラー処理

どの層で問題が起こっても、SRPI 環境ではサービス要求が正常に実行されません。SRPI は、アプリケーションをトランスポート層のエラーからできる限り切り離します。サーバー処理内部のエラーは、アプリケーションで処理します。その他のエラーは SRPI に原因があり、原因に応じた対応が行われます。

トランスポート層エラー

SRPI は、トランスポート層エラーからのリカバリーを試みます。リカバリーが不可能な場合、SRPI は、トランスポート層の障害を示す戻りコードをリクエスターに返します。この種の障害は、プログラマーがトランスポート機構の問題判別手順を使用して処理しなければなりません。

アプリケーション・エラー

SRPI は、要求をサーバーにルーティングし、応答をリクエスターに返す役割を果たします。リクエスターおよびサーバーは、サーバーで起こるエラー (異常終了以外) の処理を行います。サーバーが異常終了した場合は、SRPI は異常終了を通知する SRPI 戻りコードをリクエスターに返します。

サーバーの戻りコードは、VM または MVS の下で稼働する IBM ホスト・コンピューターのサーバーによって設定されます。サーバーの戻りコードの値および意味は、リクエスターまたはサーバーによって異なります。

SEND_REQUEST 処理エラー

SEND_REQUEST 関数の処理において、SRPI 戻りコードには数多くのエラーが指定されます。そうしたエラーには、以下のものがあります。

- 関数パラメーターが無効
- サーバーが認識されない
- サーバーとコンタクトできない

この他に、SRPI 内部エラーを示すシステム・エラー・コードがあります。

SRPI 戻りコードのタイプ

SRPI 戻りコードには、タイプ 0、1、2、3 があります。

タイプ 0

SEND_REQUEST 関数が正常終了したことを示します。

タイプ 1

SRPI ルーターによって検出され、要求の処理が行われなかったエラー原因を示します。

タイプ 2

SRPI ルーターによって検出され、確認交換単位によってリモート・コンピューターに報告されたエラーを示します。

タイプ 3

リモート・コンピューターによって検出され、確認交換単位によって SRPI ルーターに報告されたエラーを示します。

戻りコードの値は、ワード反転し、各ワード内ではバイト反転します。例えば、SRPI 戻りコード X'0100 0402' は、CPRB メモリーには X'0204 0001' と保管されます。

タイプ 0 戻りコードの定義

タイプ 0 の戻りコード (定数戻りコード UERERROK) は、X'0000 0000' の形式をとります。この戻りコードの値は、SRPI 関数が正常終了したことを示します。

タイプ 1 戻りコードの定義

タイプ 1 の戻りコードは、X'0100 nnnn' の形式をとります。

nnnn バイトは、検出された特定のエラーを示す 16 進値です。

戻りコードの定義および説明を、表 42 に示します。

表 42. タイプ 1 戻りコードの定義および説明

16 進数戻りコード	定数戻りコード	説明
X'0100 0402'	UERERRT1START	ホストの ECF プログラムが開始されていないため、SRPI が開始されない。
X'0100 0404'	UERERRT1LOAD	SRPI ルーターがロードされていない。
X'0100 0408'	UERERRT1BUSY	SRPI ルーターが使用中。この戻りコードは、パーソナル・コミュニケーションズ・プログラムでは使用されない。
X'0100 040A'	UERERRT1VER	SRPI ルーターに渡された CPRB のバージョン ID が、SRPI ルーターの常駐部でサポートされていない。バージョン ID は、C のインターフェース機能によって、自動的に CPRB にセットされる。
X'0100 040C'	UERERRT1EMU	パーソナル・コミュニケーションズがロードされていない。

表 42. タイプ 1 戻りコードの定義および説明 (続き)

16 進数戻りコード	定数戻りコード	説明
X'0100 040E'	UERERRT1ROUT	CPRB 内に指定したサーバー名が、サーバー・ルーティング・テーブルで定義されていない。デフォルト・ルーティングが構成されていないので、SRPI が要求をルーティングできない。有効なサーバー名を使用するか、または構成を更新して該当サーバー名を組み込んでください。
X'0100 0410'	UERERRT1COMMR	通信リソースが使用できない。
X'0100 0412'	UERERRT1REST	3270 エミュレーションが前回の使用以降に再び開始された。アプリケーションを終了し、再び開始してから SRPI を使用してください。
X'0100 0414'	UERERRT1INUSE	ファイル転送で使用中の通信セッションに、要求がルーティングされている。
X'0100 0602'	UERERRT1QPLEN	要求パラメーターの長さが最大値を超えている。許容最大値は 32763 である。
X'0100 0604'	UERERRT1RPLEN	応答パラメーター・バッファーの長さが最大値を超えている。許容最大値は 32763 である。
X'0100 0606'	UERERRT1VERB	verb タイプが無効である。または、サポートされていない。 SRPI ルーターに渡された CPRB 内の verb タイプは識別されない。 verb タイプは、C のインターフェース機能によって自動的に CPRB にセットされる。
X'0100 0608'	UERERRT1SERV	サーバー名が無効。ホストに送信する、サーバー名に使用されている文字 (1 つ以上) が EBCDIC に変換できない。
X'0100 060C'	UERERRT1QPAD	以下のいずれかの条件に該当する。 • 要求パラメーターのアドレスが無効である。 • 要求パラメーターの長さが、要求パラメーター・バッファーの終わりを超えている。 • 要求パラメーターのアドレスが 0 であるのに、要求パラメーターの長さが 0 でない。
X'0100 060E'	UERERRT1QDAD	以下のいずれかの条件に該当する。 • 要求データのアドレスが無効である。 • 要求データの長さが、要求データ・バッファーの終わりを超えている。 • 要求データのアドレスが 0 であるのに、要求データの長さが 0 でない。

表 42. タイプ 1 戻りコードの定義および説明 (続き)

16 進数戻りコード	定数戻りコード	説明
X'0100 0610'	UERERRT1RPAD	以下のいずれかの条件に該当する。 • 応答パラメーター・バッファーのアドレスが無効である。 • 応答パラメーター・バッファーの長さが、応答パラメーター・バッファーの終わりを超えている。 • 応答パラメーター・バッファーのアドレスが 0 であるのに、応答パラメーターの長さが 0 でない。
X'0100 0612'	UERERRT1RDAD	以下のいずれかの条件に該当する。 • 応答データ・バッファーのアドレスが無効である。 • 応答データ・バッファーの長さが、応答データ・バッファーの終わりを超えている。 • 応答データ・バッファーのアドレスが 0 であるのに、応答データの長さが 0 でない。
X'0100 0616'	UERERRT1TOPV	TopView 環境がサポートされていない。この戻りコードは、パーソナル・コミュニケーションズ・プログラムでは使用されない。
X'0100 0622'	UERERRT1INV3270 d	3270 画面更新の通知標識が無効である。 3270 画面更新の通知標識は、CPRB 内で X'00' (ユーザーに 3270 画面更新を通知する) または X'FF' (ユーザーに 3270 画面更新を通知しない) に設定しなければならない。
X'0100 0624'	UERERRT1INVCPRB	CPRB のセグメントが無効。 CPRB のアドレスが、切り捨てられた CPRB 構造体を指している。 CPRB 構造全体が十分入る大きさの読み取り/書き込みデータ・セグメントを使用してください。
X'0100 0802'	UERERRT1CNCL	リモート・コンピューターが、要求の処理中に通信セッションを取り消した。エミュレーター・セッションで F3 キーを押してリモート・プログラムを停止すると、この状態になる。ただし、この値は、ユーザーがセッションを取り消した場合以外にも使用される。要求の処理中にセッションが取り消されたという通知を SRPI がホストから受け取ると、常にこの値が使用される。

表 42. タイプ 1 戻りコードの定義および説明 (続き)

16 進数戻りコード	定数戻りコード	説明
X'0100 0C00'	UERERRT1CONV	システム・エラーが発生しました。以下のいずれかの理由により、ホストとの会話が終了した。 • ホストの通信セッションが活動状態ではない。 • リンク・レベルの通信エラーが発生した。 • システムが、確実にホストとのデータ送受信ができなかった。例えば、シーケンス・エラーが発生した。
X'0100 0C02'	UERERRT1ISE	SRPI ルーターの内部ソフトウェア・エラーのためシステム・エラーが発生した。
X'0100 0C04'	UERERRT1PROT	システム・エラーが発生しました。これは、プロトコル違反エラーでる。あるいは、SRPI ルーターまたはホストのシステム・ソフトウェア・エラーである。
X'0100 0C06'	UERERRT1SYIN	システム・エラーが発生しました。システムの不整合により、エラーが発生した。これは、SRPI ルーターのシステム・ソフトウェア・エラーである。

タイプ 2 戻りコードの定義

タイプ 2 の戻りコードは、X'02xx yyzz' の形式をとります。

特定のエラーを示す 3 バイトは、以下の確認交換単位の例外条件から構成されます。

- *xx* 例外クラス
- *yy* 例外コード
- *zz* 例外オブジェクト

注: 定数は、提供されていません。

タイプ 3 戻りコードの定義

タイプ 3 の戻りコードは、X'03xx yyzz' の形式をとります。

特定のエラーを示す 3 バイトは、以下の確認交換単位の例外条件から構成されます。

- *xx* 例外クラス
- *yy* 例外コード
- *zz* 例外オブジェクト

戻りコードの定義および説明を、表 43 に示します。

表 43. タイプ 3 戻りコードの定義および説明

16 進数戻りコード	定数戻りコード	説明
X'0304 1D00'	UERERRT3NORES	要求を処理するために、ホストの SRPI ルーターに必要なリソースが使用できない。これは、一時的な状態の場合がある。
X'0304 1E00'	UERERRT3NOSER	サーバーがホストで不明。
X'0304 1F00'	UERERRT3UNSER	ホストでサーバーを使用できない。
X'0304 2200'	UERERRT3TERMS	サーバーは正常終了したが、応答を送信していない。
X'0304 2300'	UERERRT3ABNDS	サーバーは異常終了し、応答を送信していない。

タイプ 2 およびタイプ 3 のクラス定義

例外クラスには、構文例外クラス、セマンティック例外クラス、および処理例外クラスがあります。

- **構文例外クラス:** このクラスは、伝送単位の構文規則違反 (例えば、サーバー戻りコード・パラメーターの省略: X'0202 1A08') を報告します。一般に、構文例外を報告する戻りコードは、SRPI ルーターまたはホストのシステム・ソフトウェア・エラーを示します。
- **セマンティック例外クラス:** このクラスは、パラメーターの競合 (例えば、相関値が無効: X'0203 1B00') を報告します。一般に、セマンティック例外を報告する戻りコードは、SRPI ルーターまたはホストのシステム・ソフトウェア・エラーを示します。
- **処理例外クラス:** このクラスは、要求処理中の例外条件 (例えば、サーバーが不明 : X'0304 1E00') を報告します。

例外クラスの定義を表 44 に示します。

表 44. タイプ 2 およびタイプ 3 のクラス定義

値	定義
X'00' ～ X'01'	予約済み
X'02'	構文
X'03'	構文
X'04'	処理
X'05' ～ X'FF'	予約済み

タイプ 2 およびタイプ 3 の例外コードの値

例外コードは特定のエラー条件を定義するものであり、各エラーごとに必要です。例外コード値を表 45 に示します。

表 45. タイプ 2 およびタイプ 3 の例外コードの値

値	定義
X'00'	予約済み
X'08'	セグメンテーション
X'0C'	無効なオペランド ID
X'0F'	無効な長さ
X'16'	無効なサブフィールドのタイプ
X'18'	無効なサブフィールドの値
X'19'	必要なオペランドがない
X'1A'	必要なサブフィールドがない
X'1B'	相関エラー
X'1C'	データが許容最大長を超えた
X'1D'	リソースが使用できない
X'1E'	サーバーが不明
X'1F'	サーバーが使用できない
X'20'	パラメーター長
X'21'	データ長
X'22'	正常終了
X'23'	異常終了 (サーバー異常終了)
X'24'	サブフィールドが複数回現れる
X'25'	オペランドが複数回現れる

注: この表にない例外コード値は、すべて予約済みです。

タイプ 2 およびタイプ 3 の例外オブジェクトの値

例外オブジェクトは、不正な伝送単位オブジェクトを定義します。例外オブジェクトは、構文エラーで必須です。例外オブジェクトの値を表 46 に示します。

表 46. タイプ 2 およびタイプ 3 の例外オブジェクトの値

値	定義
X'00'	指定なし
X'01'	接頭部
X'07'	コマンド・オペランド
X'08'	コマンド・サブフィールド
X'1C'	パラメーター・オペランド
X'1D'	データ・オペランド
X'13'	接尾部

注: この表にない例外オブジェクトの値は、すべて予約済みです。

サーバー戻りコード

サーバー戻りコードは、サーバー・プログラムが指定するダブルワード (4 バイト) の戻りコードです。サーバー戻りコードは、リクエスター・プログラムに返されます。戻り状況の内容および意味は、リクエスターまたはサーバーによって定義されます。サーバーの戻りコードの詳細は、ホスト担当者に問い合わせるか、または以下の資料を参照してください。

- *TSO/E Version 2 Guide to the Server - Requester Programming Interface*
- *IBM Programmer's Guide to the Server - Requester Programming Interface for VM/System Product*

付録 E. 16 ビット環境での DDE 関数

この付録では 16 ビット・モードでの DDE 関数について説明します。この情報は、16 ビットから 32 ビット・モードへ移行する場合に役立ちます。

PC/3270 Windows モード および PC400 は、動的データ交換 (DDE) インターフェースを備えており、これにより、アプリケーションでデータを交換することができます。2 つの Windows アプリケーション間のデータの交換は、クライアントとサーバー間の会話として考えることができます。クライアント は、DDE 会話を開始します。これに対しサーバー は、クライアントへの対応を行います。パーソナル・コミュニケーションズは、パーソナル・コミュニケーションズが管理しているオープン・セッションのための DDE サーバーです。 DDE の詳細については、「*Microsoft Windows Software Development Kit Guide to Programming*」を参照してください。

注: Visual Basic で DDE 関数を使用する場合は、323 ページの『第 7 章 DDE クライアント・アプリケーションを使った DDE 関数の使用方法』を参照してください。

パーソナル・コミュニケーションズ 16 ビット環境での DDE データ・アイテム

Microsoft Windows DDE は、データ・アイテムを識別するために 3 つのレベルの命名形式を使用します。つまり、アプリケーション、トピック、アイテムです。 表 47 はこれらのレベルを説明したものです。

表 47. データ・アイテム用の命名形式

レベル	説明	例
アプリケーション	Windows タスク、またはアプリケーションの特定のタスク。本書ではアプリケーションとは、パーソナル・コミュニケーションズのことです。	IBM3270
トピック	アプリケーションの特定の部分。	SessionA
アイテム	データ交換で受け渡しできるデータ・オブジェクト。アイテムとは、アプリケーションに定義されたデータ・アイテムのことで、Windows クリップボード形式の 1 つか、またはアプリケーションが独自に定義したクリップボード形式に従います。 Windows のクリップボード形式の詳細については、「*Microsoft Windows Software Development Kit Guide to Programming*」を参照してください。	PS (表示スペース)

パーソナル・コミュニケーションズは、Windows DDE サーバーとして IBM3270 IBM5250 をサポートします。

次のトピックを使用できます。

- System

- SessionA、SessionB、 ...、SessionZ
- LUA_xxxx、LUB_xxxx、 ...、LUZ_xxxx

DDE では、アトム を使用してアプリケーション名、トピック名、およびデータ・アイテムを識別します。アトムは、固有の整数値に縮小された文字ストリングを表します。この文字ストリングは、アトム・テーブルに追加され、このテーブルを参照すれば、アトムに関連付けられたストリングの値を見つけることができます。アトムは GlobalAddAtom 関数呼び出しによって作成します。アトムの作成方法と使用方法の詳細については、「*Microsoft Windows Software Development Kit Guide to Programming*」を参照してください。

システム・トピック・データ・アイテムの使用

DDE インターフェースを提供するアプリケーションは、SYSTEM という名前の特殊なトピックも提供することになっています。このトピックが、アプリケーションに一般的な関連をもつ情報アイテムに関するコンテキストを提供します。パーソナル・コミュニケーションズの SYSTEM トピックには、次の関連データ・アイテムが含まれます。

アイテム	関数
Formats	パーソナル・コミュニケーションズでレンダリング可能なクリップボード形式 (番号) のリストを戻します。
Status	各パーソナル・コミュニケーションズのセッションの状況に関する情報を戻します。
SysCon	パーソナル・コミュニケーションズのサポートのレベルと、その他のシステム関連の値を戻します。
SysItems	パーソナル・コミュニケーションズのシステム・トピックに接続したときに使用できるデータ・アイテムのリストを戻します。
Topics	使用可能なパーソナル・コミュニケーションズのトピックのリストを戻します。

セッション・トピック・データ・アイテムの使用

各セッション・トピックに対して、次のデータ・アイテムがサポートされています。

アイテム	関数
CLOSE	ウィンドウ・クローズ要求を取り出します。
EPS	追加データをもつセッションの表示スペースを取り出します。
EPSCOND	表示スペース・サービス条件を取り出します。
FIELD	セッションの表示スペース内のフィールドを取り出します。
KEYS	キー・ストロークを取り出します。
MOUSE	マウス入力を取り出します。
OIA	オペレーター情報域の状況表示行を取り出します。
PS	セッションの表示スペースを取り出します。
PSCOND	セッションのアドバイス条件を取り出します。
SSTAT	セッションの状況を取り出します。
STRING	ASCII ストリング・データを取り出します。
TRIMRECT	現行のトリミング長方形内にあるセッションの表示スペースを取り出します。

LU トピック・データ・アイテムの使用 (PC/3270 のみ)

各 LU トピックに対して、次のデータ・アイテムがサポートされています。

アイテム	関数
SF	Destination/Origin 構造化フィールド・データを取り出します。
SFCOND	Query Reply データを取り出します。

16 ビット環境での DDE 関数

表 48 は、パーソナル・コミュニケーションズで使用できる DDE 関数を示したものです。

表 48. 16 ビット環境での DDE 関数

関数	PC/3270 Windows	PC400
398 ページの『Find Field』	YES	YES
400 ページの『Get Keystrokes』	YES	YES
401 ページの『Get Mouse Input』	YES	YES
404 ページの『Get Number of Close Requests』	YES	YES
405 ページの『Get Operator Information Area』	YES	YES
405 ページの『Get Partial Presentation Space』	YES	YES
408 ページの『Get Presentation Space』	YES	YES
410 ページの『Get Session Status』	YES	YES
411 ページの『Get System Configuration』	YES	YES
413 ページの『Get System Formats』	YES	YES
413 ページの『Get System Status』	YES	YES
414 ページの『Get System SysItems』	YES	YES
415 ページの『Get System Topics』	YES	YES
416 ページの『Get Trim Rectangle』	YES	YES
417 ページの『Initiate Session Conversation』	YES	YES
418 ページの『Initiate Structured Field Conversation』	YES	NO
419 ページの『Initiate System Conversation』	YES	YES
419 ページの『Put Data to Presentation Space』	YES	YES
420 ページの『Search for String』	YES	YES
421 ページの『Send Keystrokes』	YES	YES
423 ページの『Session Execute Macro』	YES	YES
429 ページの『Set Cursor Position』	YES	YES
431 ページの『Set Mouse Intercept Condition』	YES	YES
433 ページの『Set Presentation Space Service Condition』	YES	YES
434 ページの『Set Session Advise Condition』	YES	YES
436 ページの『Set Structured Field Service Condition』	YES	NO
437 ページの『Start Close Intercept』	YES	YES
438 ページの『Start Keystroke Intercept』	YES	YES
440 ページの『Start Mouse Input Intercept』	YES	YES
443 ページの『Start Read SF』	YES	NO
445 ページの『Start Session Advise』	YES	YES
447 ページの『Stop Close Intercept』	YES	YES
447 ページの『Stop Keystroke Intercept』	YES	YES
448 ページの『Stop Mouse Input Intercept』	YES	YES
449 ページの『Stop Read SF』	YES	NO

表 48. 16 ビット環境での DDE 関数 (続き)

関数	PC/3270 Windows	PC400
449 ページの『Stop Session Advise』	YES	YES
450 ページの『Terminate Session Conversation』	YES	YES
450 ページの『Terminate Structured Field Conversation』	YES	NO
451 ページの『Terminate System Conversation』	YES	YES
451 ページの『Write SF』	YES	NO

16 ビット DDE 関数の要約については、465 ページの『16 ビット環境での DDE 関数の要約』を参照してください。

パラメーターの命名規則

ほとんどの DDE パラメーターは、名前およびローカル変数のパラメーターです。これらのローカル変数には、パラメーターの一般的なタイプを示す接頭部が付いており、その後にパラメーターの内容を説明する 1 つ以上の単語が続きます。本書で提示する接頭部は次のとおりです。

a アトム

c 文字 (1 バイト値)

f 16 ビット整数にパックされたビット・フラグ

h 16 ビット・ハンドル

p Short 型 (16 ビット) ポインター

lp Long 型 (32 ビット) ポインター

w Short 型 (16 ビット) 符号なし整数

u 符号なし整数

sz Null 文字で終了する文字ストリング

Find Field

3270	*5250*	*VT*
YES	YES	YES

Find Field 関数は、**Set Presentation Space Service Condition** 関数によって指定したフィールドの情報をクライアントへ戻します。

注: クライアントはこの関数を使用する前に、**Set Presentation Service Condition** 関数を使用して PS 位置を設定しなければなりません。

クライアントはフィールド情報を受け取るために次のメッセージを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
             MAKELONG(cfFormat, aFIELD) );
```

ここで、

cfFormat フィールド情報の形式を識別します。これは、CF_DSPTEXT でなければなりません。
aFIELD フィールド・データ・アイテムを識別します。

Find Field 関数は、 Visual Basic のような新しい形式をサポートします。新しい形式を使用する場合は、**Find Field** 関数は指定されたタイプでフィールドを見つけることが可能です。新しい形式は以下のとおりです。

```
FIELD (pos, type)
```

pos パーソナル・コミュニケーションズが、対象のフィールドを検索し始める位置。
type 対象フィールドのタイプ。フィールドのタイプは以下のとおりです。

タイプ	意味
␢␢ または T␢	このフィールド。
P␢	前のフィールド。保護、無保護のいずれでも可。
N␢	次のフィールド。保護または無保護。
NP	次の保護フィールド。
NU	次の無保護フィールド。
PP	前の保護フィールド。
PU	前の無保護フィールド。

注: ␢ 記号は必要なブランクを表します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズは、次のフィールド情報を DDE データ・メッセージ内に戻すか、

- PS 開始位置
- 長さ
- 属性値

```
WM_DDE_DATA(hData, aFIELD)
```

または状況情報が入った ACK メッセージで応答します。

```
WM_DDE_ACK(wStatus, aFIELD)
```

パーソナル・コミュニケーションズがフィールド情報を戻せない場合は、次の状況コードのうち 1 つが wStatus ワードの下位バイト内に戻されます。

戻りコード	説明
1	PS 位置が無効です。
2	PS が定様式化されていません。
6	無効な形式が指定されました。
9	システム・エラーが発生しました。

フィールド情報の構造体

パーソナル・コミュニケーションズは、フィールド情報を次の構造体で戻します。

```
typedef struct tagFINDFIELD
{
 unsigned unused:12;        // *** unused ***
 unsigned fResponse:1;      // TRUE = DDE_REQUEST response
 unsigned fRelease:1;       // TRUE = Client frees this data
 unsigned reserved:1;       // *** reserved ***
 unsigned fAckReq:1;        // TRUE = Client returns DDE_ACK
 int      cfFormat;         // Format of Field data CF_DSPTEXT
 unsigned char cAttribute;  // Attribute character
 unsigned uFieldStart;      // Field start offset
 unsigned uFieldLength;     // Field Length;
}  FINDFIELD, far *lpFINDFIELD;
```

Get Keystrokes

3270	*5250*	*VT*
YES	YES	YES

Get Keystrokes 関数は、**Start Keystroke Intercept** 関数によって代行受信したキー・ストロークをクライアントへ戻します。クライアントはキー・ストローク情報を受け取るために次のメッセージを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
             MAKELONG(cfFormat, aKEYS) );
```

ここで、

cfFormat キー・ストローク情報の形式を識別します。これは、CF_DSPTEXT でなければなりません。

aKEYS キー・ストローク・データ・アイテムを識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズは、キー・ストロークを DDE データ・メッセージ内に戻すか、または、状況情報が入った次の ACK メッセージの 1 つで応答します。

- WM_DDE_DATA(hData, aKEYS)
- WM_DDE_ACK(wStatus, aKEYS)

パーソナル・コミュニケーションズがキー・ストローク情報を戻せない場合は、次の状況コードのうち 1 つ が wStatus ワードの下位バイト内に戻されます。

戻りコード	説明
2	キー・ストロークは代行受信されませんでした。
6	無効な形式が指定されました。
9	システム・エラーが発生しました。

キー・ストローク情報の構造体

パーソナル・コミュニケーションズは、キー・ストローク情報を次の構造体で戻します。

```
typedef struct tagKEYSTROKE
{
 unsigned unused:12;        // *** unused ***
 unsigned fResponse:1;      // TRUE = DDE_REQUEST response
 unsigned fRelease:1;       // TRUE = Client frees this data
 unsigned reserved:1;       // *** reserved ***
 unsigned fAckReq:1;        // TRUE = Client returns DDE_ACK
 int      cfFormat;         // Format of keystroke data CF_DSPTEXT
 unsigned uTextType;        // Type of keystrokes
 unsigned char szKeyData[1];// Keystrokes
}  KEYSTROKE, far *lpKEYSTROKE;
```

キー・ストローク・パラメーターの形式は、**Session Execute Macro** 関数の SENDKEY コマンドのパラメーターと同じです。

次のキー・テキスト・タイプがサポートされています。

```
WC_CHARACTER  0 // Pure text, no command
WC_TOKEN      1 // including commands
```

Get Mouse Input

3270	*5250*	*VT*
YES	YES	YES

Get Mouse Input 関数は、**Start Mouse Input Intercept** 関数によって代行受信した最新のマウス入力をクライアントへ戻します。

注: クライアントは、この関数を使用する前に **Start Mouse Input Intercept** 関数を呼び出す必要があります。

クライアントはマウス入力情報を受け取るために次のコマンドを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
             MAKELONG(cfFormat, aMOUSE) );
```

ここで、

cfFormat 表示スペースの形式を識別します。有効値は、CF_TEXT または CF_DSPTEXT です。これらの 2 つの形式でのマウス入力データの構造体については、後で示します。

aMOUSE マウス入力をアイテムとして識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズは、マウス入力データを DDE データ・メッセージ内に戻すか、または、次の ACK メッセージの 1 つで応答します。

- WM_DDE_DATA(hData, aMOUSE)
- WM_DDE_ACK(wStatus, aMOUSE)

パーソナル・コミュニケーションズがマウス入力情報を戻せない場合は、次の状況コードのうち 1 つが wStatus ワードの下位バイト内に戻されます。

戻りコード	説明
2	マウス入力情報は、代行受信されませんでした。
6	無効な形式が指定されました。
9	システム・エラーが発生しました。

マウス入力情報の構造体

形式が CF_TEXT の場合は、パーソナル・コミュニケーションズは以下の形式でマウス入力情報を戻します。

```
typedef struct tagMOUSE_CF_TEXT
{
 unsigned      unused:12,       // **** Unused ****
 unsigned      fRespons:1,      // TRUE = DDE_REQUEST response
 unsigned      fRelease:1,      // TRUE = Client frees this data
 unsigned      reserved:1,      // **** Reserved ****
 unsigned      fAckReq:1;       // TRUE = Client returns DDE_ACK
 int           cfFormat;        // CF_TEXT
 unsigned char PSPos[4];        // PS position
 unsigned char Tab1[1];         // TAB character
 unsigned char PSRowPos[4];     // PS row position
 unsigned char Tab2[1];         // TAB character
 unsigned char PSColPos[4];     // PS columns position
 unsigned char Tab3[1];         // TAB character
 unsigned char PSSize[4];       // Size of the PS
 unsigned char Tab4[1];         // TAB character
 unsigned char PSRows[4];       // PS number of rows
 unsigned char Tab5[1];         // TAB character
 unsigned char PSCols[4];       // PS number of columns
 unsigned char Tab6[1];         // TAB character
 unsigned char ButtonType[1];   // Pressed button type
 unsigned char Tab7[1];         // TAB character
 unsigned char ClickType[1];    // Click type
 unsigned char Tab8[1];         // TAB character
 unsigned char ClickString[1];  // Retrieved string
} MOUSE_CF_TEXT, FAR *lpMOUSE_CF_TEXT;
```

次の表はパラメーターの値を示しています。

パラメーター名	意味	値
PSPos	マウスがクリックされた位置の PS オフセット	0 ... (PSSize - 1)
PSRowPos	マウスがクリックされた位置の行番号	0 ... (PSRows - 1)
PSColPos	マウスがクリックされた位置の桁番号	0 ... (PSCols - 1)
PSSize	表示スペースのサイズ	
PSRows	表示スペースの行数	
PSCols	表示スペースの桁数	
ButtonType	クリックされたマウス・ボタンのタイプ	**L** 左ボタン **M** 中央ボタン **R** 右ボタン

パラメーター名	意味	値
ClickType	クリックのタイプ	**S** シングルクリック **D** ダブルクリック
ClickString	マウスが指示した取り出しストリング	'\0' で終了する文字ストリング
Tab1～8	区切り文字としてのタブ文字	'\t'

形式が CF_DSPTEXT の場合は、パーソナル・コミュニケーションズは以下の形式でマウス入力情報を戻します。

```
typedef struct tagMOUSE_CF_DSPTEXT
{
 unsigned      unused:12,         // **** Unused ****
 unsigned      fRespons:1,        // TRUE = DDE_REQUEST response
 unsigned      fRelease:1,        // TRUE = client frees the storage
 unsigned      reserved:1,        // **** Reserved ****
 unsigned      fAckReq:1;         // TRUE = client returns DDE_ACK
 int           cfFormat;          // CF_DSPTEXT
 unsigned      uPSPos;            // PS position
 unsigned      uPSRowPos;         // PS row position
 unsigned      uPSColPos;         // PS column position
 unsigned      uPSSize;           // Size of the presentation space
 unsigned      uPSRows;           // PS number of rows
 unsigned      uPSCols;           // PS number of columns
 unsigned      uButtonType;       // Pressed button type
 unsigned      uClickType;        // Click type
 unsigned char szClickString[1];  // Retrieved string
} MOUSE_CF_DSPTEXT, FAR *lpMOUSE_CF_DSPTEXT;
```

次の表はパラメーターの値を示しています。

パラメーター名	意味	値
uPSPos	マウスがクリックされた位置の PS オフセット	0 ... (uPSSize - 1)
uPSRowPos	マウスがクリックされた位置の行番号	0 ... (uPSRows - 1)
uPSColPos	マウスがクリックされた位置の桁番号	0 ... (uPSCols - 1)
uPSSize	表示スペースのサイズ	
uPSRows	表示スペースの行数	
uPSCols	表示スペースの桁数	
uButtonType	クリックされたマウス・ボタンのタイプ	**0x0001** 左ボタン **0x0002** 中央ボタン **0x0003** 右ボタン
uClickType	クリックのタイプ	**0x0001** シングルクリック **0x0002** ダブルクリック
szClickString	マウスが指示したストリング	'\0' で終了する文字ストリング

Get Number of Close Requests

3270	*5250*	*VT*
YES	YES	YES

Get Number of Close Requests 関数は、 **Start Close Intercept** 関数によって代行したクローズ要求回数をクライアントへ戻します。クライアントはクローズ要求回数を受け取るために次のメッセージを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
             MAKELONG(cfFormat, aCLOSE) );
```

ここで、

cfFormat クローズ・インターセプト情報の形式を識別します。これは、CF_DSPTEXT でなければなりません。

aCLOSE クローズ・インターセプト・データ・アイテムを識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズはクローズ要求回数を DDE データ・メッセージ内に戻すか、または、状況情報が入った、次の ACK メッセージの 1 つで応答します。

- WM_DDE_DATA(hData, aCLOSE)
- WM_DDE_ACK(wStatus, aCLOSE)

パーソナル・コミュニケーションズがクローズ・インターセプト情報を戻せない場合は、次の状況コードのうち 1 つ が wStatus ワードの下位バイト内に戻されます。

戻りコード	説明
6	無効な形式が指定されました。
9	システム・エラーが発生しました。

クローズ要求回数情報の構造体

パーソナル・コミュニケーションズは、クローズ・インターセプト情報を次の構造体で戻します。

```
typedef struct tagCLOSEREQ
{
 unsigned unused:12;        // *** unused ***
 unsigned fResponse:1;      // TRUE = DDE_REQUEST response
 unsigned fRelease:1;       // TRUE = Client frees this data
 unsigned reserved:1;       // *** reserved ***
 unsigned fAckReq:1;        // TRUE = Client returns DDE_ACK
 int      cfFormat;         // Format of close intercept data CF_DSPTEXT
 unsigned uCloseReqCount;   // Number of the close requests.
}  CLOSEREQ, far *lpCLOSEREQ;
```

Get Operator Information Area

3270	*5250*	*VT*
YES	YES	YES

Get Operator Information Area (OIA) 関数は、 OIA のコピーをクライアントへ戻します。クライアントは OIA を要求するために次のメッセージを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
             MAKELONG(cfFormat, aOIA) );
```

ここで、

cfFormat OIA の形式を識別します。 OIA の場合、この形式は CF_DSPTEXT でなければなりません。

aOIA オペレーター情報域をアイテムとして識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズは OIA を DDE データ・メッセージ内に戻すか、または、次の ACK メッセージの 1 つで応答します。

- WM_DDE_DATA(hData, aOIA)
- WM_DDE_ACK(wStatus, aOIA)

パーソナル・コミュニケーションズが OIA を戻せない場合は、次の状況コードのうち 1 つ が wStatus ワードの下位バイト内に戻されます。

戻りコード	説明
6	無効な形式が指定されました。
9	システム・エラーが発生しました。

オペレーター情報域の構造体

パーソナル・コミュニケーションズは、オペレーター情報域を次の構造体で戻します。

```
typedef struct tagOIADATA
{
 unsigned unused:12;        // *** unused ***
 unsigned fResponse:1;      // TRUE = DDE_REQUEST response
 unsigned fRelease:1;       // TRUE = Client frees this data
 unsigned reserved:1;       // *** reserved ***
 unsigned fAckReq:1;        // TRUE = Client returns DDE_ACK
 int  cfFormat;             // Format of OIA data CF_DSPTEXT
}  OIADATA, far *lpOIADATA;
```

Get Partial Presentation Space

3270	*5250*	*VT*
YES	YES	YES

Get Partial Presentation Space 関数は、セッション表示スペースの一部または全部をクライアントへ戻します。

注: クライアントはこの関数を使用する前に、 **Set Presentation Space Service Condition** 関数を使用して PS 開始位置と長さを設定 (または EOF フラグを設定) しなければなりません。

クライアントは表示スペースを獲得するために次のコマンドを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
             MAKELONG(cfFormat, aEPS) );
```

ここで、

cfFormat 表示スペースの形式を識別します。有効値は、CF_TEXT または CF_DSPTEXT です。これら 2 つの形式での表示スペースの構造体については、後で示します。

aEPS 表示スペース・アトムをアイテムとして識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズは、表示スペース・データを戻すか、または、wStatus ワードの下位バイトにエラー・コードが入った、次の ACK メッセージの 1 つで応答します。

- WM_DDE_DATA(hData, aEPS)
- WM_DDE_ACK(wStatus, aEPS)

パーソナル・コミュニケーションズが表示スペースを戻せない場合は、次の状況コードのうち 1 つが wStatus ワードの下位バイト内に戻されます。

戻りコード	説明
1	事前に **Set Presentation Space Service Condition** 関数が呼び出されなかった、または無効なパラメーターが設定されました。
6	無効な形式が指定されました。
9	システム・エラーが発生しました。

表示スペースの構造体

パーソナル・コミュニケーションズは、**Get Partial Presentation Space** 要求に指定された形式で表示スペースの一部を戻します。

形式が CF_DSPTEXT の場合は、パーソナル・コミュニケーションズは次の形式で表示スペースを戻します。

```
typedef struct tagEPS_CF_DSPTEXT
{
 unsigned Unused:12,           // Unused
 unsigned fResponse:1,         // TRUE = DDE_REQUEST response
 unsigned fRelease:1,          // TRUE = client frees the storage
 unsigned reserved:1,          // **** Reserved ****
 unsigned fAckReq:1,           // TRUE = DDE_ACK requested
 int      cfFormat;            // Format data is rendered in
 unsigned uPSPosition;         // Start PS position
```

```
 unsigned uPSLength;           // Length of the part of the PS
 unsigned uPSRows;             // PS number of rows
 unsigned uPSCols;             // PS number of columns
 unsigned uPSOffset;           // Offset to the presentation space
 unsigned uFieldCount;         // Number of fields
 unsigned uFieldOffset;        // Offset to the field array
 unsigned char PSData[1];      // PS and Field list Array
} EPS_CF_DSPTEXT, FAR *lpEPS_CF_DSPTEXT;

typedef struct tagPSFIELDS
{
 unsigned char cAttribute;     // Attribute Character
 unsigned uFieldStart;         // Field start offset
 unsigned uFieldLength;        // Field Length
} PSFIELDS, FAR *lpPSFIELDS;
```

注: 次の例は、PS および PSFIELDS 配列に対して long 型のポインターを取得する方法を示しています。

```
  lpps = (lp_EPS_CF_DSPTEXT) lpEPS_CF_DSPTEXT->PSData
       + lpEPS_CF_DSPTEXT->uPSOffset;
  lppsfields = (lpPSFIELDS) lpEPS_CF_DSPTEXT->PSData
             + lpEPS_CF_DSPTEXT->uFieldOffset;
```

形式が CF_TEXT の場合、パーソナル・コミュニケーションズは次の形式で表示スペースの一部を戻します。

```
typedef struct tagEPS_CF_TEXT
{
 unsigned      Unused:12;       // **** Unused ****
 unsigned      fResponse:1;     // TRUE = DDE_REQUEST response
 unsigned      fRelease:1;      // TRUE = Client frees this data
 unsigned      reserved:1;      // **** Reserved ****
 unsigned      fAckReq:1;       // TRUE = Client returns DDE_ACK
 int           cfFormat;        // Format of the data
 unsigned char PSPOSITION[4];   // Start PS position
 unsigned char Tab1[1];         // Tab character
 unsigned char PSLENGTH[4];     // Length of the part of the PS
 unsigned char Tab2[1];         // Tab character
 unsigned char PSROWS[4];       // Number of rows in the Partial PS
 unsigned char Tab3[1];         // Tab character
 unsigned char PSCOLS[4];       // Number of columns in the PS
 unsigned char Tab4[1];         // Tab character
 unsigned char PS[1];           // PS
} EPS_CF_TEXT, FAR *lpEPS_CF_TEXT;
```

バッファー内の PS の後に、フィールド・リストを構成する、次のようなフィールドの追加構造体が続きます。

```
typedef struct tagFL_CF_TEXT
{
 unsigned char Tab5[1];       // Tab character
 unsigned char PSFldCount[4]; // Number of fields in the PS
 unsigned char Tab6[1];       // Tab character
 PS_FIELD      Field[1];      // Field List Array
} FL_CF_TEXT, FAR *lpFL_CF_TEXT;

typedef struct tagPS_FIELD
{
 unsigned char FieldStart[4];
 unsigned char TabF1[1];
 unsigned char FieldLength[4];
 unsigned char TabF2[1];
 unsigned char Attribute;
 unsigned char TabF3[1[;
} PS_FIELD, FAR *lpPS_FIELD;
```

注: 次の例は、PS および PS_FIELD 配列に対して long 型のポインターを取得する方法を示しています。

```
lpps = lpEPS_CF_TEXT->PS;
lpps_field = (lpPS_FIELD) lpEPS_CF_TEXT->PS
           + atoi(lpEPS_CF_TEXT->PSLENGTH)
           + ((atoi(lpEPS_CF_TEXT->PSROWS) - 1) * 2) // CR/LF
           + 1 + 1 + 4 + 1;   // Tabs + size of field count
```

Get Presentation Space

3270	*5250*	*VT*
YES	YES	YES

Get Presentation Space 関数は、セッション表示スペースをクライアントへ戻します。クライアントは表示スペースを獲得するために次のコマンドを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
             MAKELONG(cfFormat, aPS) );
```

ここで、

cfFormat 表示スペースの形式を識別します。有効値は、CF_TEXT または CF_DSPTEXT です。これら 2 つの形式での表示スペースの構造体については、後で示します。

aPS 表示スペース・アトムをアイテムとして識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズは、表示スペースおよびその表示スペースを構成するフィールドのリストを戻すか、または、wStatus ワードの下位バイトにエラー・コードが入った、次の ACK メッセージの 1 つで応答します。

- WM_DDE_DATA(hData, aPS)
- WM_DDE_ACK(wStatus, aPS)

パーソナル・コミュニケーションズが表示スペースを戻せない場合は、次の状況コードのうち 1 つが wStatus ワードの下位バイト内に戻されます。

戻りコード	説明
6	無効な形式が指定されました。
9	システム・エラーが発生しました。

表示スペースの構造体

パーソナル・コミュニケーションズは、**Get Presentation Space** 要求に指定された形式で表示スペースを戻します。

形式が CF_DSPTEXT の場合は、パーソナル・コミュニケーションズは次の形式で表示スペースを戻します。

```
typedef struct tagPS_CF_DSPTEXT
{
 unsigned Unused:12,           // Unused
 unsigned fResponse:1,         // TRUE = DDE_REQUEST response
 unsigned fRelease:1,          // TRUE = client frees the storage
 unsigned reserved:1,          // **** Reserved ****
 unsigned fAckReq:1,           // TRUE = DDE_ACK requested
 int      cfFormat;            // Format data is rendered in
 unsigned uPSSize;             // Size of the presentation space
 unsigned uPSRows;             // PS number of rows
 unsigned uPSCols;             // PS number of columns
 unsigned uPSOffset;           // Offset to the presentation space
 unsigned uFieldCount;         // Number of fields
 unsigned uFieldOffset;        // Offset to the field array
 unsigned char PSData[1];      // PS and Field list Array
} PS_CF_DSPTEXT, FAR *lpPS_CF_DSPTEXT;

typedef struct tagPSFIELDS
{
 unsigned char cAttribute;     // Attribute Character
 unsigned uFieldStart;         // Field start offset
 unsigned uFieldLength;        // Field Length
} PSFIELDS, FAR *lpPSFIELDS;
```

注: 次の例は、PS および PSFIELDS 配列に対して long 型のポインターを取得する方法を示しています。

```
  lpps = (lp_PS_CF_DSPTEXT) lpPS_CF_DSPTEXT->PSData
       + lpPS_CF_DSPTEXT->uPSOffset;
  lppsfields = (lpPSFIELDS) lpPS_CF_DSPTEXT->PSData
             + lpPS_CF_DSPTEXT->uFieldOffset;
```

形式が CF_TEXT の場合は、パーソナル・コミュニケーションズは次の形式で表示スペースを戻します。

```
typedef struct tagPS_CF_TEXT
{
 unsigned      Unused:12;       // **** Unused ****
 unsigned      fResponse:1;     // TRUE = DDE_REQUEST response
 unsigned      fRelease:1;      // TRUE = Client frees this data
 unsigned      reserved:1;      // **** Reserved ****
 unsigned      fAckReq:1;       // TRUE = Client returns DDE_ACK
 int           cfFormat;        // Format of the data
 unsigned char PSSIZE[4];       // Size of the PS
 unsigned char Tab1[1];         // Tab character
 unsigned char PSROWS[4];       // Number of rows in the PS
 unsigned char Tab2[1];         // Tab character
 unsigned char PSCOLS[4];       // Number of Cols in the PS
 unsigned char Tab3[1];         // Tab character
 unsigned char PS[1];           // PS
} PS_CF_TEXT, FAR *lpPS_CF_TEXT;
```

バッファー内の PS の後に、フィールド・リストを構成する、次のようなフィールドの追加構造体が続きます。

```
typedef struct tagFL_CF_TEXT
{
 unsigned char Tab4[1];       // Tab character
 unsigned char PSFldCount[4]; // Number of fields in the PS
 unsigned char Tab5[1];       // Tab character
 PS_FIELD      Field[1];      // Field List Array
} FL_CF_TEXT, FAR *lpFL_CF_TEXT;

typedef struct tagPS_FIELD
{
 unsigned char FieldStart[4];
```

```
 unsigned char TabF1[1];
 unsigned char FieldLength[4];
 unsigned char TabF2[1];
 unsigned char Attribute;
 unsigned char TabF3[1];
} PS_FIELD, FAR *lpPS_FIELD;
```

注: 次の例は、PS および PS 配列に対して long 型のポインターを取得する方法を示しています。

```
lpps = lpPS_CF_TEXT->PS;
lpps_field = (lpPS_FIELD) lpPS_CF_TEXT->PS
           + atoi(lpPS_CF_TEXT->PSSIZE)
           + ((atoi(lpPS_CF_TEXT->PSROWS) - 1) * 2) // CR/LF
           + 1 + 1 + 4 + 1;   // Tabs + size of field count
```

Get Session Status

3270	*5250*	*VT*
YES	YES	YES

Get Session Status 関数は、接続したセッションの状況を戻します。クライアントは、セッション状況を要求するために次のメッセージを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
             MAKELONG(cfFormat, aSSTAT) );
```

ここで、

cfFormat 状況情報の DDE 形式を識別します。使用する値は CF_TEXT です。
aSSTAT 要求するデータ・アイテムとしてセッション状況を識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズは、セッション状況を DDE データ・メッセージ内に戻すか、または状況情報が入った ACK メッセージの 1 つで応答します。

- WM_DDE_DATA(hData, aSSTAT)
- WM_DDE_ACK(wStatus, aSSTAT)

パーソナル・コミュニケーションズがセッション状況を戻せない場合は、次の状況コードのうち 1 つが wStatus ワードの下位バイト内に戻されます。

戻りコード	説明
6	無効な形式が指定されました。
9	システム・エラーが発生しました。

状況情報の形式

パーソナル・コミュニケーションズは、セッション状況を CF_TEXT 形式のテキストとして戻します。次の考えられる値と共に、次のフィールドが戻されます。

フィールド	戻される値	説明
状況	Closed、Invisible、Maximized、Minimized、Normal	ウィンドウはこれらの状態のいずれかです。
使用法	DDE、User	セッションは DDE セッションまたはユーザー・セッションで接続されています。
スクリーン X	NN	画面の水平方向のサイズを定義します。
スクリーン Y	NN	画面の垂直方向のサイズを定義します。
カーソル X	NN	カーソルの水平方向の位置を定義します。 (0 ... ScreenX - 1)
カーソル Y	NN	カーソルの垂直方向の位置を定義します。 (0 ... ScreenY - 1)
トリミング長方形状況	Closed、Moved、Sized	トリミング長方形の現在の設定。
トリミング長方形 X1	N	トリミング長方形の左上隅の X 位置 (文字座標)
トリミング長方形 Y1	N	トリミング長方形の左上隅の Y 位置 (文字座標)
トリミング長方形 X2	N	トリミング長方形の右下隅の X 位置 (文字座標)
トリミング長方形 Y2	N	トリミング長方形の右下隅の Y 位置 (文字座標)
セッション表示スペース状況	N	表示スペースの現在の設定。次の値が考えられます。 **0:** 表示スペースはロックされていません。 **4:** 表示スペースは使用中です。 **5:** 表示スペースはロックされています。
セッション・ウィンドウ・ハンドル	XXXX	セッションのウィンドウ・ハンドル

注:

1. 各フィールド状況は、その状況が要求されるたびに更新されます。
2. パーソナル・コミュニケーションズの将来のバージョンでは、新しいフィールドが追加されることもあります。

Get System Configuration

3270	*5250*	*VT*
YES	YES	YES

Get System Configuration 関数は、パーソナル・コミュニケーションズがサポートするレベルとその他のシステムに関連した値を戻します。この情報のほとんどは、ユーザーがシステム・エラーを受け取った後、IBM のサポート部門へ連絡したときにサービス・コーディネーターが使用するためのものです。

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
             MAKELONG(cfFormat, aSYSCON) );
```

ここで、

cfFormat 要求するデータ・アイテムの DDE 形式を識別します。使用する値は CF_TEXT です。

aSYSCON 要求するデータ・アイテムとしてシステム構成を識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズはシステム構成データ・アイテムを DDE DATA メッセージ内に戻すか、または状況情報が入った ACK メッセージの 1 つで応答します。

- WM_DDE_DATA(hData, aSYSCON)
- WM_DDE_ACK(wStatus, aSYSCON)

パーソナル・コミュニケーションズがシステム構成を戻せない場合は、wStatus ワードの下位バイトにエラー・コードが入った、次の DDE ACK メッセージが戻されます。

```
WM_DDE_ACK(wStatus, aSYSCON)
```

戻りコード	説明
9	システム・エラーが発生しました。

システム構成情報の形式

パーソナル・コミュニケーションズは、システム構成を CF_TEXT 形式のテキストとして戻します。次の考えられる値と共に、次のフィールドが戻されます。

フィールド	戻される値	説明
バージョン	N	パーソナル・コミュニケーションズのバージョン
レベル	NN	パーソナル・コミュニケーションズのレベル
予約済み	XXXXXX	予約済み
予約済み	XXXX	予約済み
モニター・タイプ	MONO、CGA、EGA、VGA、XGA	モニターのタイプ
国コード	NNNN	3270 または 5250 で使用される国コード

Get System Formats

3270	*5250*	*VT*
YES	YES	YES

Get System Formats 関数は、パーソナル・コミュニケーションズがサポートする Windows クリップボード形式のリストを戻します。クライアント・アプリケーションは、パーソナル・コミュニケーションズがサポートする形式リストを取り出すために次のメッセージを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
             MAKELONG(cfFormat, aFORMATS) );
```

ここで、

cfFormat 要求するデータ・アイテムの DDE 形式を識別します。使用する値は CF_TEXT です。

aFORMATS 要求するデータ・アイテムとして形式を識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズは、サポートしている Windows クリップボード形式のリストを CF_TEXT 形式で DDE DATA メッセージ内に戻します。

```
WM_DDE_DATA(hData, aFORMATS)
```

パーソナル・コミュニケーションズは、次の Windows クリップボード形式をサポートしています。

- CF_TEXT
- CF_DSPTEXT

パーソナル・コミュニケーションズが形式データ・アイテムを戻せない場合は、wStatus ワードの下位バイトにエラー・コードが入った、次の DDE ACK メッセージが戻されます。

```
WM_DDE_ACK(wStatus, aFORMATS)
```

戻りコード	説明
9	システム・エラーが発生しました。

Get System Status

3270	*5250*	*VT*
YES	YES	YES

Get System Status 関数は、各 3270 または 5250の状況を戻します。クライアント・アプリケーションは、状況データ・アイテムを取り出すために次のメッセージを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
             MAKELONG(cfFormat, aSTATUS) );
```

ここで、

cfFormat 要求するデータ・アイテムの DDE 形式を識別します。使用する値は CF_TEXT です。
aSTATUS 要求するデータ・アイテムとして状況を識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズは、状況データ・アイテムを CF_TEXT 形式で DDE DATA メッセージ内に戻します。

```
WM_DDE_DATA(hData, aSTATUS)
```

パーソナル・コミュニケーションズは、オープンされている各セッションごとに状況情報行を戻します。それぞれの状況情報行には、次の範囲の値を持つ一連のフィールドが入っています。

フィールド	値の範囲	説明
セッション ID	A、B、...、Z	セッションの短縮 ID
ホスト・タイプ	370、400	パーソナル・コミュニケーションズが現在サポートしているホスト・システム
エミュレーション・タイプ	3270、5250	パーソナル・コミュニケーションズがサポートしているエミュレーション・タイプ
セッション状況	Closed、Invisible、Normal、Minimized、Maximized	セッションのウィンドウの現在の設定

パーソナル・コミュニケーションズが状況データ・アイテムを戻せない場合は、wStatus ワードの下位バイトにエラー・コードが入った、次の DDE ACK メッセージが戻されます。

```
WM_DDE_ACK(wStatus, aSTATUS)
```

戻りコード	説明
9	システム・エラーが発生しました。

Get System SysItems

3270	*5250*	*VT*
YES	YES	YES

パーソナル・コミュニケーションズは、DDE システム・トピックをサポートしているので、クライアント・アプリケーション はシステム・トピックに接続し、パーソナル・コミュニケーションズに関する情報と、パーソナル・コミュニケーションズが管理しているセッションの状況についての情報を取得できます。

Get System SysItems 関数は、パーソナル・コミュニケーションズのシステム・トピック内で使用できるデータ・アイテムのリストを戻します。クライアント・アプリケーションは、システム・トピック・データ・アイテムを取得するために次のメッセージを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
             MAKELONG(cfFormat, aSYSITEMS) );
```

ここで、

cfFormat 要求するデータ・アイテムの DDE 形式を識別します。使用する値は CF_TEXT です。
aSYSITEMS 要求するデータ・アイテムとして SysItems を識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズは、システム・トピック・データ・アイテムのリストを CF_TEXT 形式で DDE DATA メッセージ内に戻します。

```
WM_DDE_DATA(hData, aSYSITEMS)
```

パーソナル・コミュニケーションズでサポートされるデータ・アイテムは次のとおりです。

- SysItems
- Topics
- Status
- Formats
- SysCon

パーソナル・コミュニケーションズがシステム・データ・アイテムを戻せない場合は、wStatus ワードの下位バイトにエラー・コードが入った、次の DDE ACK メッセージが戻されます。

```
WM_DDE_ACK(wStatus, aSYSITEMS)
```

戻りコード	説明
9	システム・エラーが発生しました。

Get System Topics

3270	*5250*	*VT*
YES	YES	YES

Get System Topics 関数は、パーソナル・コミュニケーションズが現在サポートしている活動状態の DDE トピックのリストを戻します。クライアント・アプリケーションは、現在活動状態にあるトピックのリストを取り出すために次のメッセージをシステム・トピックへ送信します。

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
             MAKELONG(cfFormat, aTOPICS) );
```

ここで、

cfFormat 要求するデータ・アイテムの DDE 形式を識別します。使用する値は CF_TEXT です。
aTOPICS 要求するデータ・アイテムとしてトピックを識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズは、DDE トピックのリストを CF_TEXT 形式で DDE DATA メッセージ内に戻します。

```
WM_DDE_DATA(hData, aTOPICS)
```

パーソナル・コミュニケーションズでサポートされるトピックは次のとおりです。

- System - システム・トピック
- SessionA - セッション A トピック

  .
  .
  .
- SessionZ - セッション Z トピック

注: サポートされるセッション・トピックの実際の数は、現在オープンされているセッションの数によって決まります。クライアント・プログラムは、常にシステム・トピックのトピック・データ・アイテムを問い合わせて、現在オープンされているセッションのリストを取得する必要があります。

パーソナル・コミュニケーションズがトピックのリストを戻せない場合は、wStatus ワードの下位バイトにエラー・コードが入った、次の DDE ACK メッセージが戻されます。

```
WM_DDE_ACK(wStatus, aTOPICS)
```

戻りコード	説明
9	システム・エラーが発生しました。

Get Trim Rectangle

3270	*5250*	*VT*
YES	YES	YES

Get Trim Rectangle 関数は、表示スペースのうち、現在のトリミング長方形内にある区域をクライアントへ戻します。クライアントはトリミング長方形を受け取るために次のメッセージを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
             MAKELONG(cfFormat, aTRIMRECT) );
```

ここで、

cfFormat トリミング長方形の形式を識別します。これは CF_TEXT です。
aTRIMRECT 要求するデータ・アイテムとしてトリミング長方形を識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズはトリミング長方形を DDE データ・メッセージ内に戻すか、または、次のうちどちらかの ACK メッセージで応答します。

- WM_DDE_DATA(hData, aTRIMRECT)
- WM_DDE_ACK(wStatus, aTRIMRECT)

パーソナル・コミュニケーションズがトリミング長方形を戻せない場合は、次の状況コードのうち 1 つが wStatus ワードの下位バイト内に戻されます。

戻りコード	説明
6	無効な形式が指定されました。
9	システム・エラーが発生しました。

Initiate Session Conversation

3270	*5250*	*VT*
YES	YES	YES

Initiate Session Conversation 関数はクライアント・アプリケーション を、パーソナル・コミュニケーションズの使用可能なセッションへ接続します。セッション会話が確立されると、その会話が終了するまで、そのセッションはそのクライアント専用のものとして予約されます。

クライアント・アプリケーションは、セッションと DDE 会話を開始するために次のメッセージを送信します。

```
SendMessage( -1,
             WM_DDE_INITIATE,
             hClientWnd,
             MAKELONG(aIBM327032, aSessionN) );
```

ここで、

aIBM327032 アプリケーション・アトムを識別します。アトム aIBM3270 を作成するために使用するストリングは、IBM3270 です。PC400 の場合は、アプリケーション・アトムは aIBM5250 で、これを作成するために使用するストリングは IBM5250 です。
aSessionN トピック・アトムを識別します。アトム aSessionN を作成するために使用するストリングは、NULL または Session にセッション ID の A、B、...、Z までのいずれかを付けたものです。

パーソナル・コミュニケーションズの応答

特定のトピックが選択され、しかもパーソナル・コミュニケーションズがクライアント・アプリケーションとの会話をサポートできる場合、パーソナル・コミュニケーションズは以下のように指定して INITIATE トランザクションを確認します。

```
WM_DDE_ACK(aIBM327032, aSessionN)
```

トピックが選択されなかった場合 (aSessionN = NULL)、パーソナル・コミュニケーションズは以下のように、現在使用可能なすべてのトピックを確認することによって応答します。

```
WM_DDE_ACK(aIBM327032, aSystem)
WM_DDE_ACK(aIBM327032, aSessionA)
:
WM_DDE_ACK(aIBM327032, aSessionZ)
```

クライアント・アプリケーションは、戻されたトピック・リストから、通信したい会話を選択し、その他の必要ない会話をすべて終了します。

Initiate Structured Field Conversation

3270	*5250*	*VT*
YES	YES	YES

Initiate Structured Field Conversation 関数は、クライアント・アプリケーションとホスト・アプリケーションを接続します。 これによって、クライアント・アプリケーションとホスト・アプリケーションの間でデータを送受信できます。

クライアントは構造化フィールド会話を受け取るために次のコマンドを送信します。

```
SendMessage( -1,
             WM_DDE_INITIATE,
             hClientWnd,
             MAKELONG(aIBM3270, aLUN_xxxx) );
```

ここで、

aIBM3270 アプリケーション・アトムを識別します。

aLUN_xxxx トピック・アトムを識別します。アトム aLUN_xxxx を作成するために使用するストリングは、LU にセッション ID の A、B、...、Z までのいずれかを付け、次に “_” を付け、さらに任意の長さのユーザー定義ストリングを付けたものです。

PC/3270 の応答

PC/3270 クライアント・アプリケーションとの構造化フィールド会話をサポートできる場合には、次のパラメーターを指定して肯定応答メッセージを戻します。

```
WM_DDE_ACK(aIBM3270, aLUN_xxxx)
```

Initiate System Conversation

3270	*5250*	*VT*
YES	YES	YES

Initiate System Conversation 関数は、クライアント・アプリケーションをシステム会話へ接続します。システム会話へ接続できるクライアントは一度に 1 つだけです。クライアントはシステム会話を開始するために次のコマンドを送信します。

```
SendMessage( -1,
             WM_DDE_INITIATE,
             hClientWnd,
             MAKELONG(aIBM327032, aSystem) );
```

ここで、

aIBM327032 アプリケーション・アトムを識別します。
aSystem トピック・アトムを識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズは、クライアント・アプリケーションとのシステム・トピック会話をサポートできる場合には、次に示すパラメーターを指定して肯定応答メッセージを戻します。

```
WM_DDE_ACK(aIBM327032, aSystem)
```

Put Data to Presentation Space

3270	*5250*	*VT*
YES	YES	YES

Put Data to Presentation Space 関数は、呼び出しパラメーターで指定した位置でホスト表示スペースに書き込む、ASCIIZ データ・ストリングを送信します。クライアントはストリングを送信するために次のメッセージをセッションへ送信します。

```
PostMessage( hServerWnd,
             WM_DDE_POKE,
             hClientWnd,
             MAKELONG(hData, aEPS) );
```

ここで、

hData セッションへ送信するストリングが入っている Windows グローバル・メモリー・オブジェクトへのハンドルを識別します。このグローバル・メモリー・オブジェクトには次の構造体が入っています。

```
typedef struct tagPutString
{
 unsigned unused:13;              // ** unused **
 unsigned fRelease:1;             // Session frees memory
 unsigned freserved:2;            // ** reserved **
 int      cfFormat;               // Always CF_DSPTEXT
 unsigned uPSStart;               // PS Position
 unsigned uEOFflag;               // EOF effective switch
 char     szStringData[1];        // String Data
} PUTSTRING, FAR *lpPUTSTRING;
```

uEOFflag フィールドには、次の値が有効です。

```
WC_EFFECTEOF   0 // The string is truncated at EOF.
WC_UNEFFECTEOF 1 // The string is not truncated at EOF.
```

aEPS 表示スペース・アトムをアイテムとして識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズは、ストリング・データを受け取り、それを表示スペースへ送信し、肯定の ACK メッセージを戻します。

表示スペースがストリング・データを受け入れない場合、パーソナル・コミュニケーションズは wStatus ワードの下位バイトに、次の状況コードのうち 1 つが入った否定の ACK メッセージを戻します。

```
WM_DDE_ACK(wStatus, aEPS)
```

戻りコード	説明
1	PS 位置が無効です。
2	長さが無効です。
3	EOF フラグの値が無効です。
5	宛先 PS への入力が禁止されていました。
6	無効な形式が指定されました。
7	ストリングは切り捨てられました (配置は成功)。
9	システム・エラーが発生しました。

Search for String

3270	*5250*	*VT*
YES	YES	YES

この関数を使用すると、指定したストリングが指定した区域内にあるかどうか、クライアント・アプリケーションで表示スペースを調べることができます。

注: クライアントは、この関数を使用する前に、**Set Presentation Space Service Condition** 関数を使用して PS 開始位置、検索方向、検索したいストリング、および EOF フラグを設定しなければなりません。

クライアントはストリングを検索するために次のメッセージを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
             MAKELONG(cfFormat, aSTRING) );
```

ここで、

cfFormat 検索情報の形式を識別します。これは CF_DSPTEXT です。
aSTRING 検索データ・アイテムを識別します。

パーソナル・コミュニケーションズの応答

指定された区域内でストリングが見つかった場合、パーソナル・コミュニケーションズはそのストリングの開始位置を DDE データ・メッセージ内に戻します。

- WM_DDE_DATA(hData, aSTRING)
- WM_DDE_ACK(wStatus, aSTRING)

パーソナル・コミュニケーションズがストリングの開始位置を戻せない場合は、次の状況コードのうち 1 つが wStatus ワードの下位バイト内に戻されます。

戻りコード	説明
1	PS 位置が無効であるか、またはストリングが長すぎます。
2	ストリングが見つかりません。
6	無効な形式が指定されました。
9	システム・エラーが発生しました。

検索情報の構造体

パーソナル・コミュニケーションズは、検索情報を次の構造で戻します。

```
typedef struct tagSEARCH
{
 unsigned unused:12;        // *** unused ***
 unsigned fResponse:1;      // TRUE = DDE_REQUEST response
 unsigned fRelease:1;       // TRUE = Client frees this data
 unsigned reserved:1;       // *** reserved ***
 unsigned fAckReq:1;        // TRUE = Client returns DDE_ACK
 int      cfFormat;         // Format of Search data CF_DSPTEXT
 unsigned uFieldStart;      // String start offset
}  SEARCH, far *lpSEARCH;
```

Send Keystrokes

3270	*5250*	*VT*
YES	YES	YES

Send Keystrokes 関数は、接続したセッションへキー・ストロークを送信します。クライアントはキー・ストロークを送信するために次のメッセージをセッションへ送信します。

```
PostMessage( hServerWnd,
             WM_DDE_POKE,
             hClientWnd,
             MAKELONG(hData, aKEYS) );
```

ここで、

hData セッションへ送信するキー・ストロークが入っている Windows グローバル・メモリー・オブジェクトへのハンドルを識別します。このグローバル・メモリー・オブジェクトには次の構造体が入っています。

```
typedef struct tagKeystrokes
{
 unsigned unused:13;              // ** unused **
 unsigned fRelease:1;             // Session frees memory
 unsigned freserved:2;            // ** reserved **
 int      cfFormat;               // Always CF_DSPTEXT
 unsigned uTextType;              // Type of keystrokes
 unsigned uRetryCount;            // Retry count 1 .. 16
 unsigned char szKeyData[1];      // Keystrokes
} KEYSTROKES, FAR *lpKEYSTROKES;
```

次のキー・テキスト・タイプがサポートされています。

```
WC_PURETEXT   0 // Pure text, no AID, or included HLLAPI
                // commands
WC_HLLAPITEXT 1 // Text, including HLLAPI tokens
```

注: キー・ストロークが純粋テキストである場合、WC_PURETEXT を指定すると、考えられる最も速い方法でキー・ストロークがホストへ転送されます。 WC_HLLAPITEXT を指定した場合、キー・ストローク・データにはテキストが含まれている HLLAPI コマンドが入っている場合があります。

aKEYS キー・ストロークをアイテムとして識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズは、キー・ストロークを受け取り、それを表示スペースへ送信します。表示スペースがキー・ストロークを受け入れない場合、表示スペースへリセットが送信され、そのキー・ストロークが再度送信されます。この手順は、表示スペースがキー・ストロークを受け入れるか、または再試行カウントに達するまで続けられます。パーソナル・コミュニケーションズがキー・ストロークをホストへ送信できない場合、パーソナル・コミュニケーションズは wStatus ワードの下位バイトに、次の状況コードのうち 1 つが入った否定の ACK メッセージを戻します。それ以外の場合、パーソナル・コミュニケーションズは肯定の ACK メッセージを戻し、キー・ストロークの送信が完了したことを知らせます。

```
WM_DDE_ACK(wStatus, aKEYS)
```

戻りコード	説明
1	再試行カウントが無効でした。
2	キー・ストロークのタイプが無効でした。
6	無効な形式が指定されました。
9	システム・エラーが発生しました。

Session Execute Macro

3270	*5250*	*VT*
YES	YES	YES

ユーザーは、**DDE_EXECUTE** 関数と共にコマンドとマクロを発行できます。 **DDE_EXECUTE** 関数は、コマンド・ストリングをパーソナル・コミュニケーションズへ渡します。 コマンド・ストリングは DDE 仕様に準拠しなければなりません。DDE コマンド構文の詳細については「*Microsoft Windows Software Development Kit Guide to Programming*」を参照してください。

クライアントは、**DDE_EXECUTE** 関数を発行するために次のコマンドを送信します。

```
PostMessage ( hServerWnd,
              WM_DDE_EXECUTE,
              hClientWnd,
              MAKELONG(NULL, hCommands) );
```

ここで、

hCommands
パーソナル・コミュニケーションズのコマンドを含む Windows グローバル・メモリー・オブジェクトへのハンドルを識別します。発行できるコマンドのリストについては、『Session Execute Macro 関数のコマンド発行』を参照してください。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズがコマンド・ストリングを処理できる場合、パーソナル・コミュニケーションズは肯定の状況情報が入った ACK メッセージをクライアントへ戻します。パーソナル・コミュニケーションズがコマンド・ストリングを実行できない場合、パーソナル・コミュニケーションズは wStatus ワードの下位ワードに次のエラー・コードが含まれる ACK メッセージを戻します。

戻りコード	説明
9	システム・エラーが発生しました。

Session Execute Macro 関数のコマンド発行

ユーザーは **Session Execute Macro** 関数で次のコマンドを発行できます。

- WINDOW
- KEYBOARD
- SEND
- RECEIVE
- SENDKEY
- WAIT

各コマンドごとに別々の DDE_EXECUTE メッセージを使用してください。

注意:

- 非英数字またはブランクを含んでいる値は二重引用符で囲んでください (例えば、*"value value"*)。
- ストリング内に二重引用符を含めたい場合は、二重引用符を 2 回タイプします (例えば、this is a double quotation mark:*""*)。
- コマンドの最大長は 255 文字です。

WINDOW コマンド

```
[WINDOW(action[, "name"])]
```

ウィンドウ・アクションを実行します。ここで、

```
action = HIDE|RESTORE|MAXIMIZE|MINIMIZE|
         SHOW|CNGNAME
name = LT name or Switch List Entry name
```

注: *action* に CNGNAME が指定されている場合は *name* を指定してください。*name* が NULL ストリングである場合、デフォルトのタイトルが表示されます。

KEYBOARD コマンド

```
[KEYBOARD(action)]
```

マウスを含め、セッション・キーボードの使用を可能にしたり禁止したりします。

```
action= LOCK|UNLOCK
```

SEND コマンド

```
[SEND("pcfilename","hostfilename","options")]
```

ホストにファイルを送信します。ここで、

```
pcfilename = [path]filename[.ext]
hostfilename =
  For VM system:
    filename filetype[filemode]
  For MVS system:
     ['']filename[(membername)][']
  For CICS system:
  For OS/400 system:
    library name filename member name
```

options は、MVS、VM、CICS、QUIET、OS/400 のファイル転送オプションとエミュレーション固有の転送オプションの任意の組み合わせをスペースで区切ったものです。

転送オプションの詳細については、「*Administrator's Reference*」を参照してください。

RECEIVE コマンド

```
[RECEIVE("pcfilename","hostfilename","options")]
```

ホストからファイルを受信します。ここで、

```
pcfilename = [path]filename[.ext]
hostfilename =
  For VM system:
    filename filetype[filemode]
  For MVS system:
     ['][filename[(membername)]['][']
  For CICS system:
  For OS/400 system:
    library name filename member name
```

options は、MVS、VM、CICS、QUIET、OS/400 のファイル転送オプションとエミュレーション固有の転送オプションの任意の組み合わせをスペースで区切ったものです。

転送オプションの詳細については、「*Administrator's Reference*」を参照してください。

SENDKEY コマンド

```
[SENDKEY(token,token)]
```

パーソナル・コミュニケーションズにキー・ストロークを送信します。ここで、

```
token = text string|command|macro macroname
```

注意:

- テキスト・ストリングは二重引用符で囲みます。
- マクロは接頭符に "macro" を付けます。
- SENDKEY の引き数ストリングは、255 文字以下でなければなりません。
- 次のコマンドがサポートされています。

表 49. SENDKEY コマンド・リスト

コマンド名	トークン	PC/3270	PC400
カーソル切り替え	alt cursor	YES	YES
文字セット切り替え	alt view	YES	YES
アテンション	sys attn	YES	YES
バックスペース	backspace	YES	YES
後退タブ	backtab	YES	YES
後退タブ・ワード	backtab word	YES	YES
前進	character advance	NO	YES
文字バックスペース	backspace valid	NO	YES
画面クリア	clear	YES	YES
入力音 (クリッカー)	click	YES	YES
カラー選択 青	blue	YES	NO
カラー選択フィールド継承	field color	YES	NO
カラー選択 緑	green	YES	NO
カラー選択 ピンク	pink	YES	NO
カラー選択 赤	red	YES	NO
カラー選択 青緑	turquoise	YES	NO

表 49. *SENDKEY* コマンド・リスト *(続き)*

コマンド名	トークン	PC/3270	PC400
カラー選択 白	white	YES	NO
カラー選択 黄色	yellow	YES	NO
カーソル明滅	cursor blink	YES	YES
カーソル下移動	down	YES	YES
カーソル左移動	left	YES	YES
カーソル右移動	right	YES	YES
カーソル選択	cursor select	YES	YES
カーソル上移動	up	YES	YES
削除	delete char	YES	YES
ワード削除	delete word	YES	YES
装置取り消し (DvCnl)	device cancel	YES	YES
複写 (DUP)	dup	YES	YES
編集 消去	edit-clear	YES	YES
編集 コピー	edit-copy	YES	YES
編集 切り抜き	edit-cut	YES	YES
編集 貼り付け	edit-paste	YES	YES
編集 取り消し	edit-undo	YES	YES
フィールドの終わり	end field	YES	YES
実行 (Enter)	enter	YES	YES
EOF 消去	erase eof	YES	YES
フィールド消去	erase field	YES	NO
入力消去	erase input	YES	YES
高速カーソル下移動	fast down	YES	YES
高速カーソル左移動	fast left	YES	YES
高速カーソル右移動	fast right	YES	YES
高速カーソル上移動	fast up	YES	YES
フィールド終了	field exit	NO	YES
フィールド・マーク	field mark	YES	YES
フィールド +	field +	NO	YES
フィールド -	field -	NO	YES
グラフィック・カーソル	+cr	YES	NO
ヘルプ	help	YES	YES
拡張強調表示 フィールドの継承	field hilight	YES	NO
拡張強調表示 反転	reverse	YES	NO
拡張強調表示 下線	underscore	YES	NO
カーソル・ホーム	home	YES	YES
ホスト印刷	host print	YES	NO
入力	input	YES	YES

表 49. SENDKEY コマンド・リスト (続き)

コマンド名	トークン	PC/3270	PC400
入力 (非表示)	input nd	YES	YES
挿入切り替え	insert	YES	YES
小文字化	to lower	YES	NO
マーク 水平線下移動	mark down	YES	YES
マーク 垂直線左移動	mark left	YES	YES
マーク 垂直線右移動	mark right	YES	YES
マーク 水平線上移動	mark up	YES	YES
マーク 下移動	move down	YES	YES
マーク 左移動	move left	YES	YES
マーク 右移動	move right	YES	YES
マーク 上移動	move up	YES	YES
改行	newline	YES	YES
次ページ	page down	NO	YES
1 秒休止	pause	YES	YES
前ページ	page up	NO	YES
画面印刷	local copy	YES	YES
プログラム・アテンション (PA) キー 1	pa1	YES	NO
プログラム・アテンション (PA) キー 2	pa2	YES	NO
プログラム・アテンション (PA) キー 3	pa3	YES	NO
プログラム・ファンクション・キー 1 . . . プログラム・ファンクション・キー 24	pf1 . . . pf24	Yes . . . X	Yes . . . X
中止	quit	YES	YES
リセット	reset	YES	YES
応答時間モニター	rtm	YES	NO
ロールダウン	roll down	NO	YES
ロールアップ	roll up	NO	YES
後退	rubout	YES	YES
ルーラー	rule	YES	YES
SO/SI 表示	so si	YES	YES
SO/SI 生成	so si generate	NO	YES
システム要求	sys req	YES	YES
タブ	tab field	YES	YES
タブ・ワード	tab word	YES	YES
テスト	test request	NO	YES
マーク解除	unmark	YES	YES

表 49. *SENDKEY* コマンド・リスト *(続き)*

コマンド名	トークン	PC/3270	PC400
大文字化	to upper	YES	NO
大文字/小文字の切り替え	to other	YES	NO
バインド待ち	wait app	YES	YES
システムとの接続待ち	wait sys	YES	YES
状態変化待ち	wait trn	YES	YES
入力可能待ち	wait inp inh	YES	YES
ウィンドウの再配置 1 : : : ウィンドウの再配置 8	view 1 : : : view 8	Yes : : : X	Yes : : : X

例:

1. ログオンするには次のように指定します。

   ```
   [SENDKEY("Logon")]
   ```

2. 読み取り装置リストを得るには次のように指定します。

   ```
   [SENDKEY("RDRL", enter)]
   ```

WAIT コマンド

```
[WAIT("[time out][wait condition]")]
```

タイムアウトが満了となるか、またはクライアントが指定した待ち条件が発生するまで待ちます。このコマンドの場合、クライアントは最低 1 つのオプションを設定しなければなりません。

time out (任意指定)
: クライアントがコマンド・ステートメント内にタイムアウト値を設定する場合、 WAIT ステートメント内では次の単位が使用可能です。

  - msec
  - millisecond
  - milliseconds
  - sec
  - second
  - seconds
  - minute
  - minutes
  - hour
  - hours

wait condition (任意指定)
: 待ち条件オプションについては、クライアントは次のものを選択できます。

  while cursor at (cursor row, cursor column)
  : カーソルが、(カーソル行、カーソル列) の位置にある間、待ち続けます。

while "string"
"string" が、画面上のいずれかの場所にある間、待ち続けます。

while "string" at (cursor row, cursor column)
"string" が、画面上の (カーソル行、カーソル列) の位置にある間、待ち続けます。

until cursor at (cursor row, cursor column)
カーソルが、(カーソル行、カーソル列) へ移動するまで、待ち続けます。

until "string"
"string" が、画面上のいずれかの場所に表示されるまで、待ち続けます。

until "string" at (cursor row, cursor column)
"string" が、(カーソル行、カーソル列) に表示されるまで、待ち続けます。

例:

1. 10 秒待つには、次のように指定します。

   ```
   [WAIT("10 seconds")]
   ```

2. 画面の (2,9) に "ABCDEF" が表示されている間、待つには次のように指定します。

   ```
   [WAIT("while ""ABCDEF"" at (2,9)")]
   ```

3. 画面の (2,9) に "ABCDEF" が表示されるまで、または 8 秒後に表示されるように待つためには、次のように指定します。

   ```
   [WAIT("8 seconds until ""ABCDEF"" at (2,9)")]
   ```

Set Cursor Position

3270	*5250*	*VT*
YES	YES	YES

Set Cursor Position 関数を使用すると、クライアント・アプリケーションでセッション・ウィンドウ内のカーソル位置を設定できます。

```
PostMessage( hServerWnd,
             WM_DDE_POKE,
             hClientWnd,
             MAKELONG(hData, aSETCURSOR) );
```

ここで、

hData 次のような構造体でカーソル位置決め情報が入っている Windows グローバル・メモリー・オブジェクトに対するハンドルを識別します。

```
typedef struct tagSETCURSOR
{
 unsigned unused:13;           // ** unused **
 unsigned fRelease:1;          // TRUE = Session frees memory
 unsigned freserved:2;         // ** reserved **
 int      cfFormat;            // Always CF_DSPTEXT
 unsigned uSetCursorType;      // Cursor Set Type
 unsigned uSetCursor1;         // Cursor Row or PS Offset
 unsigned uSetCursor2;         // Cursor Col
} SETCURSOR, FAR *lpSETCURSOR;
```

パーソナル・コミュニケーションズは、カーソル位置を設定する次の 2 つの方法をサポートします。

- PS オフセット (uSetCursorType = 0)
- 行/桁の番号 (uSetCursorType = 1)

アプリケーションでどちらかの方法を指定するには、まず uSetCursorType フィールドを該当する値に設定し、その後、次のように他の 2 つのフィールド uSetCursor1 と uSetCursor2 を該当する値に設定します。

- uSetCursorType = 0 オフセット
  - uSetCursor1: 0 ... (PSsize - 1)
- uSetCursorType = 1 行/列
  - uSetCursor1: 0 ... (PSrows - 1)
  - uSetCursor2: 0 ... (PScols - 1)

aSETCURSOR カーソル位置をアイテムとして識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズはカーソル情報を受け取り、PS 内の指定された位置へカーソルを移動させます。カーソルの位置設定が成功した場合、パーソナル・コミュニケーションズはクライアント・アプリケーションへ肯定の ACK メッセージを戻します。それ以外の場合は、wStatus ワードの下位バイトに次のエラー・コードのうち 1 つが入っている否定の ACK メッセージが戻されます。

```
WM_DDE_ACK(wStatus, aSETCURSOR)
```

戻りコード	説明
1	カーソル設定タイプが無効です。 0 か 1 でなければなりません。
2	カーソル PS オフセットが無効です。 0 ... (PSsize - 1) でなければなりません。
3	カーソル行の値が無効です。 0 ... (PSrows - 1) でなければなりません。
4	カーソル列の位置が無効です。 0 ... (PScols - 1) でなければなりません。
6	無効な形式が指定されました。
9	システム・エラーが発生しました。

Set Mouse Intercept Condition

3270	*5250*	*VT*
YES	YES	YES

この関数は、代行受信するマウス入力を指定します。クライアントは、代行受信するマウス・イベントを設定するために次のコマンドを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_POKE,
             hClientWnd,
             MAKELONG(hData, aMOUSE) );
```

ここで、

hData マウス入力の代行受信の条件を指定している Windows グローバル・メモリー・オブジェクトのハンドル、CF_TEXT または CF_DSPTEXT を識別します。

aMOUSE マウス・アトムをアイテムとして識別します。

形式が CF_TEXT の場合は、クライアント・プログラムは次の構造体でこの条件を送信します。

```
typedef struct tagSETMOUSE_CF_TEXT
{
 unsigned      unused:12,           //
 unsigned      fRelease:1,          //
 unsigned      fReserved:3;         //
 int           cfFormat;            // Always CF_TEXT
 unsigned char Condition[1]         //
} SETMOUSE_CF_TEXT, FAR *lpSETMOUSE_CF_TEXT;
```

次の表はパラメーターの値を示しています。

パラメーター名	意味	値
条件	マウス代行受信の条件	\0 で終わるストリングは、次のように定義した定数で構成されます (任意の順序)。 **L** 左ボタンの代行受信が可能 **l** 左ボタンの代行受信が不可 **R** 右ボタンの代行受信が可能 **r** 右ボタンの代行受信が不可 **M** 中央ボタンの代行受信が可能 **m** 中央ボタンの代行受信が不可 **S** シングルクリックの代行受信が可能 **s** シングルクリックの代行受信が不可 **D** ダブルクリックの代行受信が可能 **d** ダブルクリックの代行受信が不可 **T** 指示されたストリングを取り出す **t** 指示されたストリングを取り出さない

形式が CF_DSPTEXT の場合は、クライアント・プログラムは次の構造体でこの条件を送信します。

```
typedef struct tagSETMOUSE_CF_DSPTEXT
{
 unsigned      unused:12,          //
 unsigned      fRelease:1,         //
 unsigned      fReserved:3;        //
 int           cfFormat;           // Always CF_DSPTEXT
 BOOL          bLeftButton;        //
 BOOL          bRightButton;       //
 BOOL          bMiddleButton;      //
 BOOL          bSingleClick;       //
 BOOL          bDoubleClick;       //
 BOOL          bRetrieveString;    //
} SETMOUSE_CF_DSPTEXT, FAR *lpSETMOUSE_CF_DSPTEXT;
```

次の表はパラメーターの値を示しています。

パラメーター名	意味	値	
bLeftButton	マウスの左ボタンの代行受信を行うかどうか	**True**	左ボタンの代行受信が可能
		False	左ボタンの代行受信が不可
bRightButton	マウスの右ボタンの代行受信を行うかどうか	**True**	右ボタンの代行受信が可能
		False	右ボタンの代行受信が不可
bMiddleButton	マウスの中央ボタンの代行受信を行うかどうか	**True**	中央ボタンの代行受信が可能
		False	中央ボタンの代行受信が不可
bSingleClick	シングルクリックの代行受信を行うかどうか	**True**	シングルクリックの代行受信が可能
		False	シングルクリックの代行受信が不可
bDoubleClick	ダブルクリックの代行受信を行うかどうか	**True**	ダブルクリックの代行受信が可能
		False	ダブルクリックの代行受信が不可
bRetrieveString	指示されているストリングを取り出すかどうか	**True**	指示されたストリングを取り出す
		False	指示されたストリングを取り出さない

パーソナル・コミュニケーションズの応答

Set Mouse Intercept Condition 要求を受信するときに、パーソナル・コミュニケーションズが指定された状況に代行受信条件を設定できる場合、ACK メッセージを戻します。それ以外の場合、wStatus フィールドの下位バイトに次の戻りコードの 1 つが入っている否定の ACK メッセージを、クライアントに戻します。

```
WM_DDE_ACK(wStatus, aMOUSE)
```

戻りコード	説明
2	Condition パラメーター内の 1 文字が無効です。
6	無効な形式が指定されました。
9	システム・エラーが発生しました。

Set Presentation Space Service Condition

3270	*5250*	*VT*
YES	YES	YES

Set Presentation Space Service Condition 関数は、次の関数を使用するための条件を設定します。

- **Get Partial Presentation Space**
- **Find Field**
- **Search for String**

クライアント・アプリケーションはこの関数を呼び出すことによって次のような条件を設定します。

- PS 開始位置
- PS の長さ
- EOF フラグ
- 検索方向
- ASCIIZ ストリング

クライアントは、上記の関数を呼び出す前に、**Set Presentation Space Service Condition** 関数を指定しなければなりません。この関数によって設定した条件は、次に **Set Presentation Space Service Condition** 関数を呼び出すまでその効力を持ち続けます。クライアントは、条件を設定するために次のメッセージを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_POKE,
             hClientWnd,
             MAKELONG(hData, aEPSCOND) );
```

ここで、

hData 次の内容を含む Windows グローバル・メモリー・オブジェクトへのハンドルを識別します。

```
typedef struct tagPSSERVCOND
{
 unsigned   unused:13,             //
 unsigned   fRelease:1,            //
 unsigned   fReserved:2;           //
 int        cfFormat;              // Always CF_DSPTEXT
 unsigned   uPSStart;              // PS Position
 unsigned   uPSLength;             // Length of PS
 unsigned   uSearchDir;            // Direction for search
 unsigned   uEOFflag;              // EOF effective switch
 char       szTargetString[1];     // Target String
} PSSERVCOND, FAR *lpPSSERVCOND;
```

uSearchDir フィールドには次の値が有効です。

```
WC_SRCHFRWD    0 // Search forward.
WC_SRCHBKWD    1 // Search backward.
```

uEOFflag フィールドには次の値が有効です。

```
WC_UNEFFECTEOF 0 // The string is not truncated at EOF.
WC_EFFECTEOF   1 // The string is truncated at EOF.
```

aEPSCOND **Set Presentation Space Service Condition** 関数用のアイテムを識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズが **Set Presentation Space Service Condition** 関数を実行できる場合、パーソナル・コミュニケーションズは次に ACK メッセージを戻します。

```
WM_DDE_ACK(wStatus, aEPSCOND)
```

パーソナル・コミュニケーションズが **Set Presentation Space Service Condition** 関数を実行できない場合、パーソナル・コミュニケーションズは wStatus の下位バイトに、次の戻りコードのうち 1 つが入っている否定の ACK メッセージを戻します。

戻りコード	説明
1	PS 位置が無効です。
2	長さが無効です。
3	EOF フラグの値が無効です。
4	検索方向の値が無効です。
6	無効な形式が指定されました。
9	システム・エラーが発生しました。

Set Session Advise Condition

3270	*5250*	*VT*
YES	YES	YES

この関数は、**Start Session Advise** 関数の DDE_ADVISE のための条件を設定します。クライアントは検索ストリングと画面の領域を指定できます。アドバイス条件が満たされると、サーバーは **Start Session Advise** 関数によって指定されたオプションに従ってその条件をクライアントへ通知します。

注: クライアントは **Start Session Advise** 関数を呼び出す前に、**Set Session Advise Condition** 関数を指定しなければなりません。 **Start Session Advise** 関数を始動した後にアドバイス条件を設定した場合、そのアドバイス条件は無視され、クライアントは否定の ACK メッセージを受け取ります。アドバイスの始動の詳細については、 445 ページの『Start Session Advise』を参照してください。

クライアントは、アドバイス条件を設定するために次のメッセージを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_POKE,
             hClientWnd,
             MAKELONG(hData, aPSCOND) );
```

ここで、

hData 次の内容を含む Windows グローバル・メモリー・オブジェクトへのハンドルを識別します。

```
typedef struct tagSEARCHDATA
{
 unsigned   unused:13,             //
 unsigned   fRelease:1,            //
 unsigned   fReserved:2;           //
 int        cfFormat;              // Always CF_DSPTEXT
 WORD       uPSStart;              // PS Position of string
 WORD       uPSLength              // Length of String
 BOOL       bCaseSensitive;        // Case Sensitive TRUE=YES
 char       SearchString[1];       // Search String
} SEARCHDATA, FAR *lpSEARCHDATA;
```

aPSCOND **Set Session Advise Condition** 関数用のアイテムを識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズが **Set Session Advise Condition** 関数を実行できる場合、パーソナル・コミュニケーションズは次の ACK メッセージを戻します。

```
WM_DDE_ACK(wStatus, aPSCOND)
```

パーソナル・コミュニケーションズが **Set Session Advise Condition** 関数を実行できない場合、パーソナル・コミュニケーションズは wStatus の下位バイトに次の戻りコードのうち 1 つが入っている否定の ACK メッセージを戻します。

戻りコード	説明
1	アドバイスはすでに活動状態です。
2	アドバイス条件はすでに活動状態です。
3	PS 位置が無効です。
4	ストリングの長さが無効です。
6	無効な形式が指定されました。
9	システム・エラーが発生しました。

Set Structured Field Service Condition

3270	*5250*	*VT*
YES	YES	YES

Set Structured Field Service Condition 関数は、クライアント・アプリケーションが提供する Query Reply データを渡します。

注: クライアントは **Start Read SF** 関数または **Write SF** 関数を呼び出す前に、**Set Structured Field Service Condition** 関数を呼び出さなければなりません。

クライアントは、条件を設定するために次のメッセージを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_POKE,
             hClientWnd,
             MAKELONG(hData, aSFCOND) );
```

ここで、

hData 次の内容を含む Windows グローバル・メモリー・オブジェクトへのハンドルを識別します。

```
typedef struct tagSFSERVCOND
{
 unsigned   unused:12,          //
 unsigned   fRelease:1,         //
 unsigned   fReserved:3;        //
 int        cfFormat;           // Always CF_DSPTEXT
 WORD       uBufferSize;        // Buffer size of Read SF
 WORD       uQRLength;          // Length of Query Reply data
 char       szQueryReply[1];    // Query Reply data
} SFSERVCOND, FAR *lpSFSERVCOND;
```

aSFCOND **Set Structured Field Service Condition** 関数用のアイテムを識別します。

PC/3270 の応答

PC/3270 は、Query Reply ID、Type (DOID ではない) およびその長さを検査します。これらが有効であれば、PC/3270 は次に ACK メッセージを戻します。

```
WM_DDE_ACK(wStatus, aSFCOND)
```

PC/3270 が **Set Structured Field Service Condition** 関数を実行できない場合、PC/3270 は wStatus の下位バイトに次の戻りコードのうち 1 つが入っている否定の ACK メッセージを戻します。

戻りコード	説明
1	PS SF ID が無効です。
2	長さが無効です。
3	このセッションにはすでに 1 つの DDM 基本タイプが接続されています。
4	Structured Field Service Condition がすでに設定されています。
6	無効な形式が指定されました。
9	システム・エラーが発生しました。

Start Close Intercept

3270	*5250*	*VT*
YES	YES	YES

Start Close Intercept 関数を使用すると、ユーザーがエミュレーター・セッション・ウィンドウからクローズ・オプションを選択したときに生成されるクローズ要求を、クライアント・アプリケーションによって代行受信できます。この関数は Stop Close Intercept 関数が要求されるまで、クローズ要求を代行受信してそれを破棄します。この関数を使用した後、クライアントはクローズ要求が発生したこと (CLOSE) を通知する DATA メッセージを受け取ります。

クライアントはセッション・アドバイスを開始するために次のコマンドを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_ADVISE,
             hClientWnd,
             MAKELONG(hOptions, aCLOSE) );
```

ここで、

hOptions 次の構造を持つ Windows グローバル・メモリー・オブジェクトへのハンドルです。

```
typedef struct tagOPTIONS
{
 unsigned reserved:14;        // *** reserved ***
 unsigned fDeferUpd:1;        // Send notification only
 unsigned fAckReq:1;          // Client will ACK all notices
 WORD     cfFormat;           // Clipboard format to use
}  OPTIONS, FAR *lpOPTIONS;
```

fDeferUpd の値が 1 である場合、NULL に設定された hData とともに DDE データ・メッセージがクライアント・アプリケーションへ送信されます。次に、クライアントはデータ・アイテムを要求するために DDE REQUEST を発行しなければなりません。

fAckReq の値が 1 である場合、サーバーはそれ以降のクローズ要求をクライアントへ通知しません。この状態は、サーバーが前の通知に対する応答で、クライアントから ACK メッセージを受け取るまで続きます。

cfFormat フィールドはクローズ要求を送信する形式を指定します。(CF_DSPTEXT 形式でなければなりません。)

aCLOSE クローズ・インターセプトをアイテムとして識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズは、代行受信を開始できる場合には Start Close Intercept を受け取って ACK メッセージを戻します。それ以外の場合、wStatus フィールドの下位バイトに次の戻りコードの 1 つが入っている否定の ACK メッセージを、クライアントに戻します。

```
WM_DDE_ACK(wStatus, aCLOSE)
```

戻りコード	説明
1	Close Intercept はすでに始動しています。
6	無効な形式が指定されました。
9	システム・エラーが発生しました。

代行受信が始動されると、クライアントはクローズ要求が代行受信されたことを知らせる次の DATA メッセージを受け取ります。

```
WM_DDE_DATA(hData, aCLOSE)
```

ここで、

hData 次の内容を含む Windows グローバル・メモリー・オブジェクトへのハンドルを識別します。

```
typedef struct tagCLOSEREQ
}
 unsigned   unused:12,       // *** unused ***
 unsigned   fResponse:1,     // TRUE = DD_REQUEST response
 unsigned   fRelease:1,      // TRUE = Client releases memory
 unsigned   reserved:1,      // *** reserved ***
 unsigned   fAckReq:1,       // TRUE = DDE_ACK is required
 int        cfFormat;        // Always CF_DSPTEXT
 WORD       uCloseReqCount;  // Counter of the Close Requests
} CLOSEREQ, FAR *lpCLOSEREQ;
```

この DATA メッセージは、Stop Close Intercept メッセージがパーソナル・コミュニケーションズへ送信されるまで続きます。

Start Keystroke Intercept

3270	*5250*	*VT*
YES	YES	YES

Start Keystroke Intercept 関数を使用すると、端末オペレーターによってセッションへ送信された任意のキー・ストロークをクライアント・アプリケーションでフィルターにかけることができます。この関数を呼び出した後では、キー・ストロークは代行受信され、クライアントがそれ (KEYS) を受け取ります。

クライアントは代行受信を開始するために次のコマンドを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_ADVISE,
             hClientWnd,
             MAKELONG(hOptions, aKEYS) );
```

ここで、

hOptions 次の構造を持つ Windows グローバル・メモリー・オブジェクトへのハンドルです。

```
typedef struct tagOPTIONS
}
 unsigned reserved:14;        // Reserved
 unsigned fDeferUpd:1;        // Send notification only
 unsigned fAckReq:1;          // Client will ACK all notices
 WORD     cfFormat;           // Clipboard format to use
}  OPTIONS, FAR *lpOPTIONS;
```

fDeferUpd の値が 1 である場合、NULL に設定された hData とともに DDE データ・メッセージがクライアント・アプリケーションへ送信されます。次にクライアントはデータ・アイテムを要求するために DDE REQUEST を発行しなければなりません。

fAckReq の値が 1 である場合、サーバーはそれ以降のキー・ストロークをクライアントへ通知しません。この状態は、サーバーが前のキー・ストロークの通知に対する応答でクライアントから ACK メッセージを受け取るまで続きます。

cfFormat フィールドは、端末オペレーターによってキー・ストロークが送信されるときにキー・ストロークを送信する形式を指定します。(CF_DSPTEXT 形式でなければなりません。)

aKEYS キー・ストロークをアイテムとして識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズは、代行受信を開始できる場合には **Start Keystroke Intercept** を受け取って ACK メッセージを戻します。それ以外の場合、wStatus フィールドの下位バイトに次の戻りコードの 1 つが入っている否定の ACK メッセージを、クライアントに戻します。

```
WM_DDE_ACK(wStatus, aKEYS)
```

戻りコード	説明
1	Keystroke Intercept はすでに始動しています。
6	無効な形式が指定されました。
9	システム・エラーが発生しました。

代行受信が始動されると、クライアントはキー・ストロークが代行受信されたことを知らせる次の DATA メッセージを受け取ります。

```
WM_DDE_DATA(hData, aKEYS)
```

この DATA メッセージは、Stop Keystroke Intercept メッセージがパーソナル・コミュニケーションズへ送信されるまで続きます。データ・アイテムの形式は、クライアントが DDE_REQUEST を経由してそのデータ・アイテムを要求した場合と同じになります。

Start Mouse Input Intercept

3270	*5250*	*VT*
YES	YES	YES

Start Mouse Input Intercept 関数を使用すると、端末オペレーターがエミュレーター・セッション・ウィンドウ上でマウス・ボタンを押したとき、クライアント・アプリケーションがマウス入力を代行受信できます。この関数を呼び出した後、クライアントは、マウス入力が発生した PS 位置を含む DATA メッセージを受信します。

クライアントはマウス入力の代行受信を開始するために、次のコマンドを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_ADVISE,
             hClientWnd,
             MAKELONG(hOptions, aMOUSE) );
```

ここで、

hOptions 次の構造を持つ Windows グローバル・メモリー・オブジェクトへのハンドルです。

```
typedef struct tagOPTIONS
{
 unsigned reserved:14;        // Reserved
 unsigned fDeferUpd:1;        // Send notification only
 unsigned fAckReq:1;          // Client will ACK all notices
 WORD     cfFormat;           // Clipboard format to use
}  OPTIONS, FAR *lpOPTIONS;
```

fDeferUpd の値が 1 である場合、NULL に設定された hData とともに DDE データ・メッセージがクライアント・アプリケーションへ送信されます。次に、クライアントはデータ・アイテムを要求するために DDE REQUEST を発行しなければなりません。

fAckReq の値が 1 である場合、サーバーはそれ以降の構造化フィールド・データをクライアントへ通知しません。この状態は、サーバーが前の通知に対する応答で、クライアントから ACK メッセージを受け取るまで続きます。

cfFormat フィールドは、更新されたデータ・アイテムを送信するための形式を指定します。

aMOUSE MOUSE をアイテムとして識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズは、代行受信を開始できる場合には **Start Mouse Input Intercept** を受け取って ACK メッセージを戻します。それ以外の場合、wStatus フィールドの下位バイトに次の戻りコードの 1 つが入っている否定の ACK メッセージを、クライアントに戻します。

```
WM_DDE_ACK(wStatus, aMOUSE)
```

戻りコード	説明
1	Mouse Input Intercept はすでに始動しています。
6	無効な形式が指定されました。
9	システム・エラーが発生しました。

Mouse Input Intercept が始動すると、クライアントは構造化フィールドの DATA メッセージを受信します。

```
WM_DDE_DATA(hData, aMOUSE)
```

ここで、

hData 形式が CF_TEXT の場合は、パーソナル・コミュニケーションズは以下の形式でマウス入力情報を戻します。

```
typedef struct tagMOUSE_CF_TEXT
{
 unsigned      unused:12,       // **** Unused ****
 unsigned      fRespons:1,      // TRUE = DDE_REQUEST response
 unsigned      fRelease:1,      // TRUE = Client frees this data
 unsigned      reserved:1,      // **** Reserved ****
 unsigned      fAckReq:1;       // TRUE = Client returns DDE_ACK
 int           cfFormat;        // CF_TEXT
 unsigned char PSPos[4];        // PS position
 unsigned char Tab1[1];         // TAB character
 unsigned char PSRowPos[4];     // PS row position
 unsigned char Tab2[1];         // TAB character
 unsigned char PSColPos[4];     // PS columns position
 unsigned char Tab3[1];         // TAB character
 unsigned char PSSize[4];       // Size of the PS
 unsigned char Tab4[1];         // TAB character
 unsigned char PSRows[4];       // PS number of rows
 unsigned char Tab5[1];         // TAB character
 unsigned char PSCols[4];       // PS number of columns
 unsigned char Tab6[1];         // TAB character
 unsigned char ButtonType[1];   // Pressed button type
 unsigned char Tab7[1];         // TAB character
 unsigned char ClickType[1];    // Click type
 unsigned char Tab8[1];         // TAB character
 unsigned char ClickString[1];  // Retrieved string
} MOUSE_CF_TEXT, FAR *lpMOUSE_CF_TEXT;
```

次の表はパラメーターの値を示しています。

パラメーター名	意味	値
PSPos	マウスがクリックされた位置の PS オフセット	0 ... (PSSize - 1)
PSRowPos	マウスがクリックされた位置の行番号	0 ... (PSRows - 1)
PSColPos	マウスがクリックされた位置の桁番号	0 ... (PSCols - 1)
PSSize	表示スペースのサイズ	
PSRows	表示スペースの行数	
PSCols	表示スペースの桁数	

パラメーター名	意味	値
ButtonType	クリックされたマウス・ボタンのタイプ	**L** 左ボタン **M** 中央ボタン **R** 右ボタン
ClickType	クリックのタイプ	**S** シングルクリック **D** ダブルクリック
ClickString	マウスが指示した取り出しストリング	'\0' で終了する文字ストリング
Tab1～8	区切り文字としてのタブ文字	'\t'

hData 形式が CF_DSPTEXT の場合は、パーソナル・コミュニケーションズは以下の形式でマウス入力情報を戻します。

```
typedef struct tagMOUSE_CF_DSPTEXT
{
 unsigned      unused:12,          // **** Unused ****
 unsigned      fRespons:1,         // TRUE = DDE_REQUEST response
 unsigned      fRelease:1,         // TRUE = client frees the storage
 unsigned      reserved:1,         // **** Reserved ****
 unsigned      fAckReq:1;          // TRUE = client returns DDE_ACK
 int           cfFormat;           // CF_DSPTEXT
 unsigned      uPSPos;             // PS position
 unsigned      uPSRowPos;          // PS row position
 unsigned      uPSColPos;          // PS column position
 unsigned      uPSSize;            // Size of the presentation space
 unsigned      uPSRows;            // PS number of rows
 unsigned      uPSCols;            // PS number of columns
 unsigned      uButtonType;        // Pressed button type
 unsigned      uClickType;         // Click type
 unsigned char szClickString[1];  // Retrieved string
} MOUSE_CF_DSPTEXT, FAR *lpMOUSE_CF_DSPTEXT;
```

次の表はパラメーターの値を示しています。

パラメーター名	意味	値
uPSPos	マウスがクリックされた位置の PS オフセット	0 ... (uPSSize - 1)
uPSRowPos	マウスがクリックされた位置の行番号	0 ... (uPSRows - 1)
uPSColPos	マウスがクリックされた位置の桁番号	0 ... (uPSCols - 1)
uPSSize	表示スペースのサイズ	
uPSRows	表示スペースの行数	
uPSCols	表示スペースの桁数	
uButtonType	クリックされたマウス・ボタンのタイプ	**0x0001** 左ボタン **0x0002** 中央ボタン **0x0003** 右ボタン

パラメーター名	意味	値
uClickType	クリックのタイプ	**0x0001** シングルクリック **0x0002** ダブルクリック
szClickString	マウスが指示した取り出しストリング	\0 で終了する文字ストリング

この DATA メッセージは、Stop Mouse Input Intercept メッセージがパーソナル・コミュニケーションズへ送信されるまで続きます。

Start Read SF

3270	*5250*	*VT*
YES	YES	YES

Start Read SF 関数を使用すると、ホスト・アプリケーションからの構造化フィールド・データをクライアント・アプリケーションによって読み取れます。この関数を使用した後、クライアントはクローズ要求が発生したことを通知する DATA メッセージを受け取ります。

注: この関数を使用する前に、クライアントは Query Reply データをサーバーへ渡すために **Set Structured Field Service Condition** 関数を呼び出さなければなりません。

クライアントは Read SF を開始するために次のコマンドを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_ADVISE,
             hClientWnd,
             MAKELONG(hOptions, aSF) );
```

ここで、

hOptions 次の構造を持つ Windows グローバル・メモリー・オブジェクトへのハンドルです。

```
typedef struct tagOPTIONS
{
 unsigned reserved:14;        // Reserved
 unsigned fDeferUpd:1;        // Send notification only
 unsigned fAckReq:1;          // Client will ACK all notices
 WORD     cfFormat;           // Clipboard format to use
}  OPTIONS, FAR *lpOPTIONS;
```

fDeferUpd の値が 1 である場合、NULL に設定された hData とともに DDE データ・メッセージがクライアント・アプリケーションへ送信されます。次に、クライアントはデータ・アイテムを要求するために DDE REQUEST を発行しなければなりません。

fAckReq の値が 1 である場合、サーバーはそれ以降の構造化フィールド・データをクライアントへ通知しません。この状態は、サーバーが前の通知に対する応答で、クライアントから ACK メッセージを受け取るまで続きます。

cfFormat フィールドは、構造化フィールド・データを送信するための形式を指定します。(CF_DSPTEXT 形式でなければなりません。)

aSF 構造化フィールドをアイテムとして識別します。

PC/3270 の応答

PC/3270 は、**Start Read SF** を始動できる場合には、Start Read SF を受け取って ACK メッセージを渡します。それ以外の場合、wStatus フィールドの下位バイトに次の戻りコードの 1 つが入っている否定の ACK メッセージを、クライアントに戻します。

```
WM_DDE_ACK(wStatus, aSF)
```

戻りコード	説明
1	Read SF はすでに始動しています。
3	前に Set Structured Field Service Condition 関数が呼び出されていません。
6	無効な形式が指定されました。
9	システム・エラーが発生しました。

Read SF を始動すると、クライアントは次の構造化フィールドの DATA メッセージを受け取ります。

```
WM_DDE_DATA(hData, aSF)
```

ここで、

hData 次の内容を含む Windows グローバル・メモリー・オブジェクトへのハンドルを識別します。

```
typedef struct tagSFDATA
{
 unsigned   unused:12,            //
 unsigned   fRelease:1,           //
 unsigned   fReserved:3;          //
 int        cfFormat;             // Always CF_DSPTEXT
 WORD       uSFLength;            // Length of SF data
 char       szSFData[1];          // SF data
} SFDATA, FAR *lpSFDATA;
```

この DATA メッセージは、Stop Read SF メッセージが PC/3270 へ送信されるまで続きます。

Start Session Advise

3270	*5250*	*VT*
YES	YES	YES

Start Session Advise 関数は、パーソナル・コミュニケーションズのセッションとクライアントの間にリンクを確立します。これにより、クライアントはデータ・アイテムが更新されるときに表示スペース (PS)、オペレーター情報域 (OIA)、またはトリミング長方形 (TRIMRECT) の更新データを受け取ります。

注: 表示スペースの更新時、クライアント・アプリケーションに条件付き通知が必要な場合は、表示スペース用のアドバイス関数を呼び出す前にアドバイス条件を設定してください。 434 ページの『Set Session Advise Condition』を参照してください。

クライアントはセッション・アドバイスを開始するために次のコマンドを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_ADVISE,
             hClientWnd,
             MAKELONG(hOptions, aItem) );
```

ここで、

hOptions
Windows グローバル・メモリー・オブジェクトへのハンドルです。構造体は次のとおりです。

```
typedef struct tagOPTIONS
{
 unsigned reserved:14;        // Reserved
 unsigned fDeferUpd:1;        // Send notification only
 unsigned fAckReq:1;          // Client will ACK all notices
 WORD     cfFormat;           // Clipboard format to use
}  OPTIONS, FAR *lpOPTIONS;
```

fDeferUpd の値が 1 である場合、NULL に設定された hData とともに DDE データ・メッセージがクライアント・アプリケーションへ送信されます。次に、クライアントはデータ・アイテムを要求するために DDE REQUEST を発行しなければなりません。

fAckReq の値が 1 である場合、サーバーはそれ以降のデータ・アイテムの変更をクライアントへ通知しません。この状態は、サーバーが前の更新通知に対する応答で、クライアントから ACK メッセージを受信するまで続きます。

アイテムが更新されると、cfFormat フィールドは、そのデータ・アイテムを送信するための形式を指定します。

aItem
要求されている情報のアイテムを指定します。この場合、値は PS、OIA、または TRIMRECT です。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズは、アドバイスを始動できる場合には、**Start Session Advise** を受け取って ACK メッセージを戻します。それ以外の場合、wStatus フィールドの下位バイトに次の戻りコードの 1 つが入っている否定の ACK メッセージを、クライアントに戻します。

```
WM_DDE_ACK(wStatus, aItem)
```

戻りコード	説明
1	アドバイスはデータ・アイテムに対してすでに活動状態です。
6	アドバイス・パラメーターが無効です。
9	システム・エラーが発生しました。

アドバイスが始動すると、クライアントはデータ・アイテム (PS、OIA、または TRIMRECT) が更新されたことを知らせる次の DATA メッセージを受け取ります。

```
WM_DDE_DATA(hData, aItem)
```

この DATA メッセージは、Stop Session Advise メッセージがパーソナル・コミュニケーションズへ送信されるまで続きます。データ・アイテムの形式は、クライアントが DDE_REQUEST を経由してそのデータ・アイテムを要求した場合と同じになります。

Stop Close Intercept

3270	*5250*	*VT*
YES	YES	YES

Stop Close Intercept 関数は、クローズ要求を代行受信するクライアント・アプリケーションの機能を終了させます。クライアントは、**Stop Close Intercept** 関数を実行するために次のコマンドを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_UNADVISE,
             hClientWnd,
             MAKELONG(NULL, aCLOSE) );
```

ここで、

aCLOSE クローズ・インターセプトをアイテムとして識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズが DDE_UNADVISE を実行できる場合、パーソナル・コミュニケーションズは肯定の状況情報が入った、次の ACK メッセージをクライアントへ戻します。

```
WM_DDE_ACK(wStatus, aCLOSE)
```

パーソナル・コミュニケーションズが DDE_UNADVISE を実行できない場合、パーソナル・コミュニケーションズは、否定の状況情報、および wStatus ワードの下位バイトに次の戻りコードのうち 1 つが入っている ACK メッセージを戻します。

戻りコード	説明
1	アドバイスはまだ始動していません。
9	システム・エラーが発生しました。

Stop Keystroke Intercept

3270	*5250*	*VT*
YES	YES	YES

Stop Keystroke Intercept 関数は、キー・ストロークを代行受信するクライアント・アプリケーションの機能を終了させます。クライアントは、**Stop Keystroke Intercept** 関数を実行するために次のコマンドを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_UNADVISE,
             hClientWnd,
             MAKELONG(NULL, aKEYS) );
```

ここで、

aKEYS キー・ストロークをアイテムとして識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズが DDE_UNADVISE を実行できる場合、パーソナル・コミュニケーションズは肯定の状況情報が入った、次の ACK メッセージをクライアントへ戻します。

```
WM_DDE_ACK(wStatus, aKEYS)
```

パーソナル・コミュニケーションズが DDE_UNADVISE を実行できない場合、パーソナル・コミュニケーションズは、否定の状況情報、および wStatus ワードの下位バイトに次の戻りコードのうち 1 つが入っている ACK メッセージを戻します。

戻りコード	説明
1	アドバイスはまだ始動していません。
9	システム・エラーが発生しました。

Stop Mouse Input Intercept

3270	*5250*	*VT*
YES	YES	YES

Stop Mouse Input Intercept 関数は、マウス入力を代行受信するクライアント・アプリケーションの機能を終了させます。

クライアントは、**Stop Mouse Input Intercept** 関数を実行するために次のコマンドを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_UNADVISE,
             hClientWnd,
             MAKELONG(NULL, aMOUSE) );
```

ここで、

aMOUSE マウスをアイテムとして識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズが DDE_UNADVISE を実行できる場合、パーソナル・コミュニケーションズは肯定の状況情報が入った、次の ACK メッセージをクライアントへ戻します。

```
WM_DDE_ACK(wStatus, aMOUSE)
```

パーソナル・コミュニケーションズが DDE_UNADVISE を実行できない場合、パーソナル・コミュニケーションズは否定の状況情報、および wStatus ワードの下位バイトに次の戻りコードのうち 1 つが入っている ACK メッセージを戻します。

戻りコード	説明
1	アドバイスはまだ始動していません。
9	システム・エラーが発生しました。

Stop Read SF

3270	*5250*	*VT*
YES	YES	YES

Stop Read SF 関数は、構造化フィールド・データを読み取るクライアント・アプリケーションの機能を終了させます。

クライアントは、**Stop Read SF** 関数を実行するために次のコマンドを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_UNADVISE,
             hClientWnd,
             MAKELONG(NULL, aSF) );
```

ここで、

aSF 構造化フィールドをアイテムとして識別します。

PC/3270 の応答

PC/3270 が DDE_UNADVISE を実行できる場合、PC/3270 は肯定の状況情報が入った、次の ACK メッセージをクライアントへ戻します。

```
WM_DDE_ACK(wStatus, aSF)
```

PC/3270が DDE_UNADVISE を実行できない場合、PC/3270は、否定の状況情報、および wStatus ワードの下位バイトに次の戻りコードのうち 1 つが入っている ACK メッセージを戻します。

戻りコード	説明
1	アドバイスはまだ始動していません。
9	システム・エラーが発生しました。

Stop Session Advise

3270	*5250*	*VT*
YES	YES	YES

Stop Session Advise 関数は、パーソナル・コミュニケーションズとクライアントの間のリンクを切断します。クライアントは **Stop Session Advise** 関数を実行するために次のコマンドを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_UNADVISE,
             hClientWnd,
             MAKELONG(NULL, aItem) );
```

ここで、

aItem 要求されている情報のアイテムを指定します。この場合、値は PS、OIA、TRIMRECT、または NULL です。

aItem が NULL である場合、クライアントはその会話に対して活動状態にあるすべての通知の終了を要求しました。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズが DDE_UNADVISE を実行できる場合、パーソナル・コミュニケーションズは肯定の状況情報が入った、次の ACK メッセージをクライアントへ戻します。

```
WM_DDE_ACK(wStatus, aItem)
```

パーソナル・コミュニケーションズが DDE_UNADVISE を実行できない場合、パーソナル・コミュニケーションズは、否定の状況情報、および wStatus ワードの下位バイトに次の戻りコードのうち 1 つが入っている ACK メッセージを戻します。

戻りコード	説明
1	アドバイスはまだ始動していません。
9	システム・エラーが発生しました。

Terminate Session Conversation

3270	*5250*	*VT*
YES	YES	YES

Terminate Session Conversation 関数は、クライアントが事前に会話を始動しているパーソナル・コミュニケーションズのセッションからクライアントを切断します。

クライアントはセッション会話を終了させるために次のコマンドを送信します。

```
SendMessage( hServerWnd,
             WM_DDE_TERMINATE,
             hClientWnd,
             MAKELONG(NULL, NULL) );
```

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズは、次の終了メッセージで終了コマンドを確認します。

```
WM_DDE_TERMINATE
```

Terminate Structured Field Conversation

3270	*5250*	*VT*
YES	YES	YES

Terminate Structured Field Conversation 関数は、クライアントを構造化フィールド会話から切断します。

クライアントは構造化フィールド会話を終了させるために次のコマンドを送信します。

```
SendMessage( hServerWnd,
             WM_DDE_TERMINATE,
             hClientWnd,
             MAKELONG(NULL, NULL) );
```

PC/3270 の応答

PC/3270 は、次の終了メッセージで終了コマンドを確認します。

```
WM_DDE_TERMINATE
```

Terminate System Conversation

3270	*5250*	*VT*
YES	YES	YES

この関数は、クライアントをシステム会話から切断します。

クライアントはシステム会話を終了させるために次のコマンドを送信します。

```
SendMessage( hServerWnd,
             WM_DDE_TERMINATE,
             hClientWnd,
             MAKELONG(NULL, NULL) );
```

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズは、次の終了メッセージで終了コマンドを確認します。

```
WM_DDE_TERMINATE
```

ユーザーがパーソナル・コミュニケーションズ・セッションをクローズすると、パーソナル・コミュニケーションズが割り振った任意のグローバル・メモリー・ブロックが Windows によって解放されます。これらのグローバル・メモリー・オブジェクトをクライアントが長時間保持している場合は、クライアントに問題が生じる恐れがあります。クライアント・アプリケーションがグローバル・メモリー・アイテム内に情報を長時間保持する必要がある場合は、グローバル・メモリー・アイテムを、クライアント・アプリケーションが所有するグローバル・メモリー・アイテム内にコピーすることをお勧めします。

Write SF

3270	*5250*	*VT*
YES	YES	YES

Write SF 関数を使用すると、クライアント・アプリケーションが構造化フィールド・データをホスト・アプリケーションへ書き込むことができます。

注: クライアントは、**Write SF** 関数を呼び出す前に **Set Structured Field Service Condition** 関数を呼び出さなければなりません。

クライアントは、構造化フィールド・データを書き込むために次のメッセージを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_POKE,
             hClientWnd,
             MAKELONG(hData, aSF) );
```

ここで、

hData 次の内容を含む Windows グローバル・メモリー・オブジェクトへのハンドルを識別します。

```
typedef struct tagWRITESF
{
 unsigned   unused:12,             //
 unsigned   fRelease:1,            //
 unsigned   fReserved:3;           //
 int        cfFormat;              // Always CF_DSPTEXT
 WORD       uSFLength;             // Length of SF data
 char       Work[8];               // Work area
 char       szSFData[1];           // SF data
} WRITESF, FAR *lpWRITESF;
```

aSF 構造化フィールドをアイテムとして識別します。

PC/3270 の応答

PC/3270 は構造化フィールド・データを受け取って、そのデータをホスト・アプリケーションへ送信します。データ伝送が正常終了した場合、PC/3270 は次の ACK メッセージを戻します。

```
WM_DDE_ACK(wStatus, aSF)
```

それ以外の場合、PC/3270 は wStatus の下位バイトに、次の戻りコードのうち 1 つが入った否定の ACK メッセージを戻します。

戻りコード	説明
2	長さが無効です。
6	無効な形式が指定されました。
9	システム・エラーが発生しました。

16 ビット環境での DDE メニュー・アイテム API

パーソナル・コミュニケーションズは、セッション・メニュー・バーに対する動的メニュー・アイテムの属性の追加、削除、および変更をサポートしています。その場合、このメニュー・アイテムに対して最大 16 個までのサブメニュー・アイテム用のスペースを持つメニューが作成されます。

パーソナル・コミュニケーションズは 2 種類の DDE 会話をサポートしています。1 つは DDE メニュー・クライアント・アプリケーションとして機能するもの、もう 1 つは DDE メニュー・サーバーとして機能するものです。

16 ビット環境での DDE メニュー・クライアント

メニュー・アイテムを追加、削除、および変更するためには、セッションと DDE メニュー・サーバー・アプリケーションとの間で次の DDE 会話が行われなければなりません。

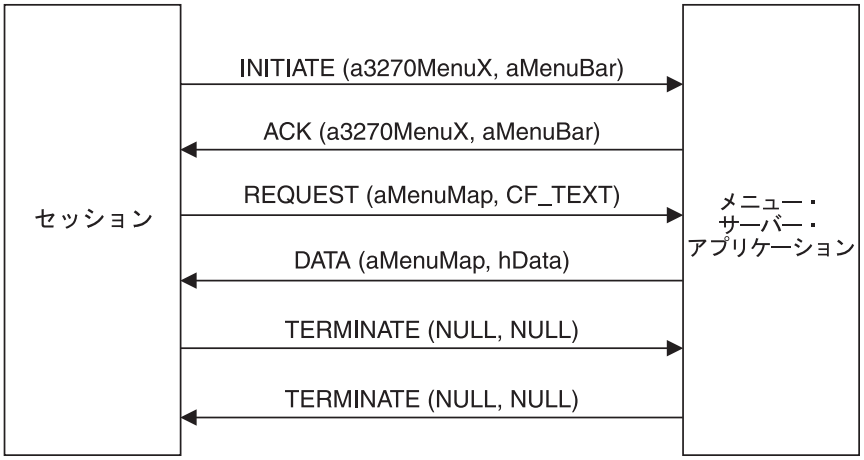

図 8. DDE メニュー・サーバーの会話

次のデータ階層は、セッション・メニュー・バーへ動的メニュー・アイテムとサブメニューを追加するときに、パーソナル・コミュニケーションズが予期するメニュー・マップを詳しく説明したものです。

```
POPUP  "MyMenu"
BEGIN
  MENUITEM "Send Files to Host",         SEND
  MENUITEM "Receive Files from Host",    RECEIVE
  MENUITEM SEPARATOR
  MENUITEM "Convert Files",              CONVERT
END
```

ユーザーが新しいメニューからメニュー・アイテムを選択すると、パーソナル・コミュニケーションズはアプリケーションとして、3270MenuN または 5250MenuN を持ち、トピックとして itemN トークンを持った DDE Initiate を送信します。 DDE アプリケーションから ACK を受け取った場合、パーソナル・コミュニケーションズはセッションがユーザー入力を受け入れるのを禁止します。その後、メニュー・クライアント・アプリケーションはダイアログなどを表示することができます。メニュー・サーバー・アプリケーションは、そのメニュー・アイテムの処理を完了した時点で DDE Terminate を送信して、パーソナル・コミュニケーションズにプロセスが完了したことを知らせます。その後、パーソナル・コミュニケーションズはユーザー用のウィンドウを再び使用可能にします。

32 ビット環境での DDE メニュー・サーバー

メニュー・アイテムを追加、削除、および変更するためには、セッションと DDE メニューのクライアント・アプリケーションとの間で 454 ページの図 9 が行われなければなりません。

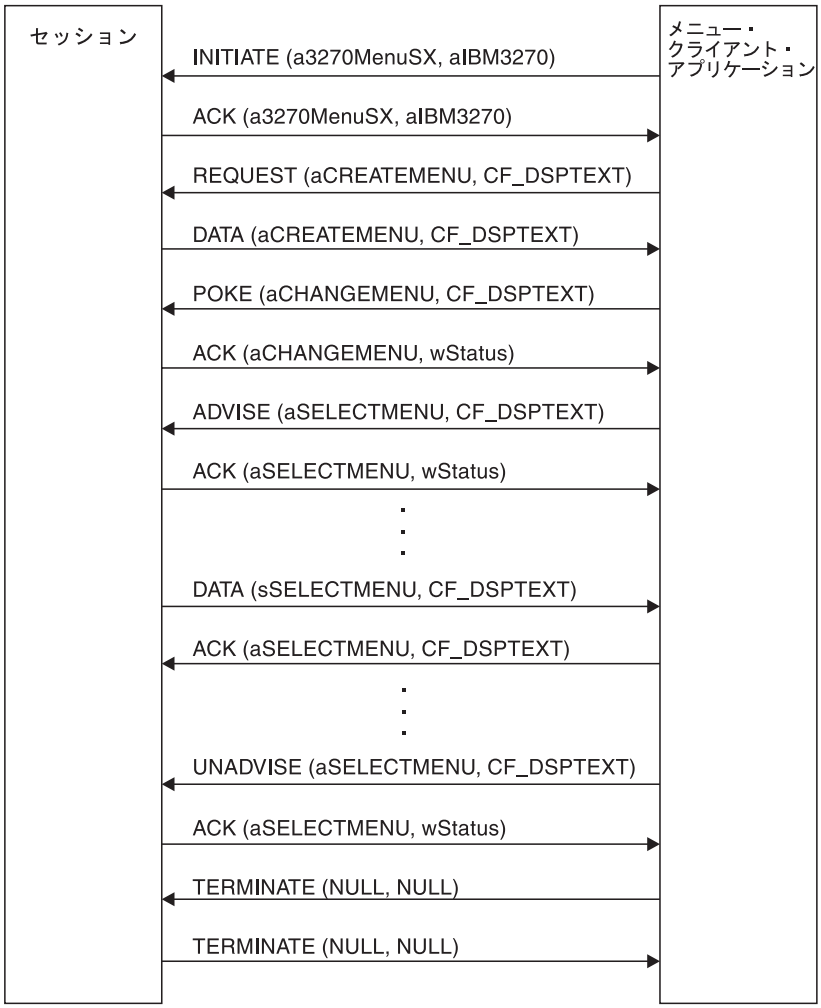

図 9. DDE メニュー・クライアントの会話

ユーザーが新しいメニューからメニュー・アイテムを選択すると、パーソナル・コミュニケーションズはアイテムとして aSELECTMENU を持った DDE DATA を送信します。パーソナル・コミュニケーションズが DDE DATA をクライアント・アプリケーションへ送信した時点で、パーソナル・コミュニケーションズはセッションがユーザー入力を受け入れることを禁止します。その後、メニュー・クライアント・アプリケーションはダイアログなどを表示することができます。メニュー・クライアント・アプリケーションは、そのメニュー・アイテムの処理を完了すると DDE ACK を送信し、パーソナル・コミュニケーションズにプロセスが完了したことを知らせます。その後、パーソナル・コミュニケーションズはユーザー用のウィンドウを再び使用可能にします。

16 ビット環境での DDE メニュー関数

このセクションでは、パーソナル・コミュニケーションズで使用できる DDE メニュー・アイテム API 関数について説明します。PC/3270 Windows モード および PC400 では、以下にリストするすべての関数を提供しています。

Change Menu Item

3270	*5250*	*VT*
YES	YES	YES

Change Menu Item 関数には、メニュー・アイテムの追加、削除、挿入、変更、および除去を行います。クライアントは、メニューを変更するためにセッションに対して次のメッセージを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_POKE,
             hClientWnd,
             MAKELONG(hData, aCHANGEMENU) );
```

ここで、

hData メニューの変更要求を含む Windows グローバル・メモリー・オブジェクトへのハンドルを識別します。このグローバル・メモリー・オブジェクトには次の構造体が入っています。

```
typedef struct tagChangeMenu
{
 unsigned unused:13;     // ** unused **
 unsigned fRelease:1;    // Session frees memory
 unsigned freserved:2;   // ** reserved **
 int      cfFormat;      // Always CF_DSPTEXT
 HANDLE   hMenu;         // Handle of the menu item
 WORD     wPosition;     // The position of the menu
                         // item
 WORD     wIDNew;        // The menu ID of the new
                         // menu item
 WORD     wOperation;    // Specifies the operation
 WORD     wFlags;        // Specifies the options
 unsigned char szItemName[1]; // String of the item
} CHANGEMENU, FAR *lpCHANGEMENU;
```

次の操作がサポートされています。

```
MF_APPEND  // Appends a menu item to the end of a menu.
MF_CHANGE  // Modifies a menu item in a menu.
MF_DELETE  // Deletes a menu item from a menu, destroying
           // the menu item.
MF_INSERT  // Inserts a menu item into a menu.
MF_REMOVE  // Removes a menu item from a menu but does not
           // destroy the menu item.
```

wOperation フィールドに MF_APPEND を指定した場合は、次のフィールドも記入しなければなりません。

hMenu 追加変更したいメニューを識別します。ポップアップ・メニューへ新しいアイテムを追加するには、 **Create Menu Item** 関数を実行したときにパーソナル・コミュニケーションズから戻すハンドルを指定します。新しいメニュー・アイテムを最上位レベルのメニュー・バーへ追加するには NULL を指定します。

wIDNew 新しいメニュー・アイテムのコマンド ID を指定します。新しいメニュー・アイテムを最上位レベルのメニュー・バーへ追加する場合には、**Create Menu Item** 関数を実行したときにパーソナル・コミュニケーションズから戻すメニュー・アイテムのハンドルを指定します。

wFlags 次のオプションを設定できます。

```
MF_CHECKED       // Places a check mark next to
                 // the item.
MF_DISABLED      // Disables the menu item so
                 // that it cannot be selected,
                 // but does not gray it.
MF_ENABLED       // Enables the menu item so that
                 // it can be selected and
                 // restores from its grayed
                 // state.
MF_GRAYED        // Disables the menu item so
                 // that it cannot be selected,
                 // and grays it.
MF_MENUBARBREAK  // Same as MF_MENUBREAK except
                 // that for pop-up menus,
                 // separates the new column from
                 // the old column with a
                 // vertical line.
MF_MENUBREAK     // Places the item on a new line
                 // for menu bar items.
                 // For pop-up menus, places the
                 // item in a new column, with
                 // no dividing line between the
                 // columns.
MF_SEPARATOR     // Draws a horizontal dividing
                 // line.  Can only be used in a
                 // pop-up menu.  This line cannot
                 // be grayed, disabled, or
                 // highlighted.  The wIDNew and
                 // szItemName fields are
                 // ignored.
MF_UNCHECKED     // Does not place a check mark
                 // next to the item (default).
```

szItemName 新しいメニュー・アイテムの内容を指定します。 NULL で終了する文字ストリングが入っています。

wOperation フィールドに MF_CHANGE を指定した場合は、次のフィールドにも記入しなければなりません。

hMenu 変更したいメニューを識別します。ポップアップ・メニューの アイテムを変更するには、 **Create Menu Item** 関数を実行したときにパーソナル・コミュニケーションズから戻すハンドルを指定します。アイテムを最上位レベルのメニュー・バーへ追加するには NULL を指定します。

nPosition 変更したいアイテム・メニューを指定します。 wPosition パラメーターの解釈は、wFlags パラメーターの設定によって異なります。

MF_BYPOSITION
既存のメニュー・アイテムの位置を指定します。メニュー内の最初のアイテムは位置ゼロです。

MF_BYCOMMAND
既存のメニュー・アイテムのコマンド ID を指定します。

wIDNew メニュー・アイテムのコマンド ID を指定します。最上位レベルのメニュー・バーのアイテムを変更する場合には、**Create Menu Item** 関数を実行したときにパーソナル・コミュニケーションズから戻すメニュー・アイテムのハンドルを指定します。

wFlags 次のオプションを設定できます。

```
MF_BYCOMMAND     // Specifies that the nPosition
                 // parameter gives the menu
                 // item control ID number.
                 // This is the default if
                 // neither MF_BYCOMMAND nor
                 // MF_BYPOSITION is set.
MF_BYPOSITION    // Specifies that the nPosition
                 // parameter gives the position
                 // of the menu item to be
                 // changed rather than an ID
                 // number.
MF_CHECKED       // Places a check mark next to
                 // the item.
MF_DISABLED      // Disables the menu item so
                 // that it cannot be selected,
                 // but does not gray it.
MF_ENABLED       // Enables the menu item so
                 // that it can be selected and
                 // restores from its grayed
                 // state.
MF_GRAYED        // Disables the menu item so
                 // that it cannot be selected,
                 // and grays it.
MF_MENUBARBREAK  // Same as MF_MENUBREAK except
                 // that for pop-up menus,
                 // separates the new column
                 // from the old column with a
                 // vertical line.
MF_MENUBREAK     // Places the item on a new
                 // line for menu bar items.
                 // For pop-up menus, places the
                 // item in a new column, with
                 // no dividing line between
                 // the columns.
MF_SEPARATOR     // Draws a horizontal dividing
                 // line.  Can only be used in
                 // a pop-up menu.  This line
                 // cannot be grayed, disabled,
                 // or highlighted.  The wIDNew
                 // and szItemName fields are
                 // ignored.
MF_UNCHECKED     // Does not place a check mark
                 // next to the item (default).
```

szItemName メニュー・アイテムの内容を指定します。NULL で終了する文字ストリングが入っています。

wOperation フィールドに MF_DELETE を指定した場合は、次のフィールドにも記入しなければなりません。

hMenu 削除したいメニューを識別します。ポップアップ・メニューからアイテムを削除するには、 **Create Menu Item** 関数を実行したときにパーソナル・コミュニケーションズから戻すハンドルを指定します。最上位レベルのメニュー・バーからアイテムを削除するには NULL を指定します。

nPosition
削除したいアイテム・メニューを指定します。nPosition パラメーターの解釈は、wFlags パラメーターの設定によって異なります。

MF_BYPOSITION
既存のメニュー・アイテムの位置を指定します。メニュー内の最初のアイテムは位置ゼロです。

MF_BYCOMMAND
既存のメニュー・アイテムのコマンド ID を指定します。

wFlags
次のオプションを設定できます。

```
MF_BYCOMMAND     // Specifies that the nPosition
                 // parameter gives the menu
                 // item control ID number.
                 // This is the default if
                 // neither MF_BYCOMMAND nor
                 // MF_BYPOSITION is set.
MF_BYPOSITION    // Specifies that the nPosition
                 // parameter gives the position
                 // of the menu item to be
                 // deleted rather than an ID
                 // number.
```

wOperation フィールドに MF_INSERT を指定した場合は、次のフィールドも記入しなければなりません。

hMenu
挿入したいメニューを識別します。アイテムをポップアップ・メニューへ挿入するには、 **Create Menu Item** 関数を実行したときにパーソナル・コミュニケーションズから戻すハンドルを指定します。アイテムを最上位レベルのメニュー・バーへ追加するには NULL を指定します。

nPosition
新しいメニュー・アイテムを挿入する前にメニュー・アイテムを指定します。 nPosition パラメーターの解釈は、wFlags パラメーターの設定によって異なります。

MF_BYPOSITION
既存のメニュー・アイテムの位置を指定します。メニュー内の最初のアイテムは位置ゼロです。

MF_BYCOMMAND
既存のメニュー・アイテムのコマンド ID を指定します。

wIDNew
メニュー・アイテムのコマンド ID を指定します。あるいは、最上位レベルのメニュー・バーのアイテムを変更する場合には、**Create Menu Item** 関数を実行したときにパーソナル・コミュニケーションズから戻すメニュー・アイテムのハンドルを指定します。

wFlags 次のオプションを設定できます。

```
MF_BYCOMMAND     // Specifies that the nPosition
                 // parameter gives the menu
                 // item control ID number. This
                 // is the default if neither
                 // MF_BYCOMMAND nor MF_BYPOSITION
                 // is set.
MF_BYPOSITION    // Specifies that the nPosition
                 // parameter gives the position
                 // of the menu item to be
                 // changed rather than an ID
                 // number.
MF_CHECKED       // Places a check mark next to
                 // the item.
MF_DISABLED      // Disables the menu item so
                 // that it cannot be selected,
                 // but does not gray it.
MF_ENABLED       // Enables the menu item so
                 // that it can be selected and
                 // restores from its grayed
                 // state.
MF_GRAYED        // Disables the menu item so
                 // that it cannot be selected,
                 // and grays it.
MF_MENUBARBREAK  // Same as MF_MENUBREAK except
                 // that for pop-up menus,
                 // separates the new column
                 // from the old column with a
                 // vertical line.
MF_MENUBREAK     // Places the item on a new
                 // line for menu bar items.
                 // For pop-up menus, places the
                 // item in a new column, with
                 // no dividing line between the
                 // columns.
MF_SEPARATOR     // Draws a horizontal dividing
                 // line.  Can only be used in
                 // a pop-up menu.  This line
                 // cannot be grayed, disabled,
                 // or highlighted. The wIDNew
                 // and szItemName fields are
                 // ignored.
MF_UNCHECKED     // Does not place a check mark
                 // next to the item (default).
```

szItemName メニュー・アイテムの内容を指定します。NULL で終了する文字ストリングが入っています。

wOperation フィールドに MF_REMOVE を指定した場合は、次のフィールドも記入しなければなりません。

hMenu 除去したいメニューを識別します。アイテムをポップアップ・メニューから除去するには、 **Create Menu Item** 関数を実行したときにパーソナル・コミュニケーションズから戻すハンドルを指定します。最上位レベルのメニュー・バーからアイテムを除去するには NULL を指定します。

nPosition　除去したいアイテム・メニューを指定します。nPosition パラメーターの解釈は、wFlags パラメーターの設定によって異なります。

MF_BYPOSITION
既存のメニュー・アイテムの位置を指定します。メニュー内の最初のアイテムは位置ゼロです。

MF_BYCOMMAND
既存のメニュー・アイテムのコマンド ID を指定します。

wFlags　次のオプションを設定できます。

```
MF_BYCOMMAND     // Specifies that the nPosition
                 // parameter gives the menu
                 // item control ID number.
                 // This is the default if
                 // neither MF_BYCOMMAND nor
                 // MF_BYPOSITION is set.
MF_BYPOSITION    // Specifies that the nPosition
                 // parameter gives the
                 // position of the menu item to
                 // be removed rather than an ID
                 // number.
```

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズはメニューの変更要求を受け取って処理します。要求を受け入れられない場合、パーソナル・コミュニケーションズは wStatus ワードの下位バイトに、次の状況コードのうち 1 つが入った否定の ACK メッセージを戻します。それ以外の場合、パーソナル・コミュニケーションズはキー・ストロークが送信されたことを知らせる 肯定の ACK メッセージを戻します。

```
WM_DDE_ACK(wStatus, aCHANGEMENU)
```

戻りコード	説明
1	無効なパラメーターが指定されました。
6	無効な形式が指定されました。
9	システム・エラーが発生しました。

Create Menu Item

3270	*5250*	*VT*
YES	YES	YES

Create Menu Item 関数は、メニュー・アイテムをメニュー・バーへ追加するようパーソナル・コミュニケーションズに要求します。それと同時にポップアップ・メニューが作成されますが、ポップアップ・メニューは最初は空なので、Create Menu Item 関数を使用してメニュー・アイテムに記入します。また、最上位レベルのメニュー・バーへ追加される新しいメニュー・アイテムのストリングも、Change Menu Item 関数を使用して指定します。

クライアントはメニュー・アイテムを作成するために次のメッセージを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
             MAKELONG(cfFormat, aCREATEMENU) );
```

ここで、

cfFormat 新しいメニュー・アイテムの ID の形式を識別します。有効値は CF_DSPTEXT です。

aCREATEMENU 作成メニュー・アイテムを識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズは、メニュー・アイテムを作成できる場合には、新しく作成したメニュー・アイテムのハンドルを DDE データ・メッセージ内に戻します。

```
WM_DDE_DATA(hData, aCREATEMENU)
```

または

```
WM_DDE_ACK(wStatus, aCREATEMENU)
```

ここで、

hData メニュー・アイテムのハンドルを含む Windows グローバル・メモリー・オブジェクトへのハンドルを識別します。このグローバル・メモリー・オブジェクトには次の構造体が入っています。

```
typedef struct tagcreatemenu
}
 unsigned   unused:12,       // *** unused ***
 unsigned   fresponse:1,     // true = dd_request response
 unsigned   frelease:1,      // true = client releases memory
 unsigned   reserved:1,      // *** reserved ***
 unsigned   fackreq:1,       // true = dde_ack is required
 int      cfformat;          // always cf_dsptext
 handle   hmemuitem;         // handle of the menu item
} CREATEMENU, FAR *lpCREATEMENU;
```

パーソナル・コミュニケーションズがメニュー・アイテムを作成できない場合は、次の状況コードのうち 1 つが wStatus ワードの下位バイト内に戻されます。

戻りコード	説明
6	無効な形式が指定されました。
9	システム・エラーが発生しました。

Initiate Menu Conversation

3270	*5250*	*VT*
YES	YES	YES

Initiate Menu Conversation 関数は、クライアント・アプリケーションを、使用可能なパーソナル・コミュニケーションズのセッションへ接続します。メニュー会話が確立されると、会話が終了するまで、そのセッションのメニューはそのクライアント専用のものとして予約されます。

クライアント・アプリケーションは、メニューとの DDE 会話を開始するために次のメッセージを送信します。

```
SendMessage( -1,
             WM_DDE_INITIATE,
             hClientWnd,
             MAKELONG(aIBM327032, SN) );
```

ここで、

aIBM327032 アプリケーション・アトムを識別します。アトム aIBM3270 を作成するために使用するストリングは IBM3270 です。 PC400 の場合は、アプリケーション・アトムは aIBM5250 で、これを作成するために使用するストリングは IBM5250 です。

SN トピック・アトムを識別します。アトム a3270MenuSN を作成するために使用するストリングは 3270MenuS でセッション ID の A、B、...、Z までのいずれかを付けたものです。PC400 の場合は、トピック・アトムは a5250MenuSN で、これを作成するために使用するストリングは 5250MenuS にセッション ID の A、B、...、Z のいずれかを付けたものです。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズがクライアント・アプリケーションとの会話をサポートできる場合、パーソナル・コミュニケーションズは次のもので INITIATE トランザクションを確認します。

```
WM_DDE_ACK(aIBM327032, SN)
```

Start Menu Advise

3270	*5250*	*VT*
YES	YES	YES

Start Menu Advise 関数を使用すると、クライアント・アプリケーションで追加したメニュー・アイテムが選択されたときに、クライアント・アプリケーションはユーザー定義ルーチンを処理できます。この関数を使用した後、クライアントは選択されたメニュー・アイテムを知らせる DATA メッセージを受け取ります。

クライアントはメニュー・アドバイスを開始するために次のコマンドを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_ADVISE,
             hClientWnd,
             MAKELONG(hOptions, aSELECTMENU) );
```

ここで、

hOptions 次の構造を持つ Windows グローバル・メモリー・オブジェクトへのハンドルです。

```
typedef struct tagOPTIONS
{
 unsigned reserved:14;       // Reserved
 unsigned fDeferUpd:1;       // Send notification only
                             // (Must be 0)
 unsigned fAckReq:1;         // Client will ACK all notices
                             // (Must be 1)
 WORD     cfFormat;          // Always CF_DSPTEXT
}  OPTIONS, FAR *lpOPTIONS;
```

aSELECTMENU メニュー・アドバイスをアイテムとして識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズは、**Start Menu Advise** 関数を始動できる場合は、Start Menu Advise を受け取って ACK メッセージを戻します。

```
WM_DDE_ACK(wStatus, aSELECTMENU)
```

それ以外の場合、wStatus フィールドの下位バイトに次の戻りコードの 1 つが入っている否定の ACK メッセージを、クライアントに戻します。

戻りコード	説明
1	メニュー・アドバイスはすでに始動しています。
6	無効な形式が指定されました。
9	システム・エラーが発生しました。

クライアント・アプリケーションに追加したメニュー・アイテムが選択されると、クライアントは選択されたメニュー・アイテムを知らせる次の DATA メッセージを受け取ります。

```
WM_DDE_DATA(hData, aSELECTMENU)
```

ここで、

hData 次の内容を含む Windows グローバル・メモリー・オブジェクトへのハンドルを識別します。

```
typedef struct tagSELECTMENU
{
 unsigned   Unused:12,       // *** unused ***
 unsigned   fResponse:1,     // TRUE = DD_REQUEST response
 unsigned   fRelease:1,      // TRUE = Client releases memory
 unsigned   reserved:1,      // *** reserved ***
 unsigned   fAckReq:1,       // TRUE = DDE_ACK is required
 int        cfFormat;        // Always CF_DSPTEXT
 WORD       uIDSelected;     // Command ID of the
                             // selected menu item
} SELECTMENU, FAR *lpSELECTMENU;
```

この DATA メッセージは、Stop Menu Advise メッセージがパーソナル・コミュニケーションズへ送信されるまで続きます。

Stop Menu Advise

3270	*5250*	*VT*
YES	YES	YES

Stop Menu Advise 関数は、クライアント・アプリケーションで追加したメニュー・アイテムが選択されたときにユーザー定義ルーチンを処理するクライアント・アプリケーションの機能を終了させます。クライアントは Stop Menu Advise 関数を実行するために次のコマンドを送信します。

```
PostMessage( hServerWnd,
             WM_DDE_UNADVISE,
             hClientWnd,
             MAKELONG(NULL, aSELECTMENU) );
```

ここで、

aSELECTMENU　メニュー・アドバイスをアイテムとして識別します。

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズが DDE_UNADVISE を実行できる場合、パーソナル・コミュニケーションズは肯定の状況情報が入った、次の ACK メッセージをクライアントへ戻します。

```
WM_DDE_ACK(wStatus, aCLOSE)
```

パーソナル・コミュニケーションズが DDE_UNADVISE を実行できない場合、パーソナル・コミュニケーションズは、否定の状況情報、および wStatus ワードの下位バイトに次の戻りコードのうち 1 つが入っている ACK メッセージを戻します。

戻りコード	説明
1	アドバイスはまだ始動していません。
9	システム・エラーが発生しました。

Terminate Menu Conversation

3270	*5250*	*VT*
YES	YES	YES

Terminate Menu Conversation 関数は、クライアントが事前に会話を始動しているパーソナル・コミュニケーションズのセッションからそのクライアントを切断します。

クライアントはセッション会話を終了させるために次のコマンドを送信します。

```
SendMessage( hServerWnd,
             WM_DDE_TERMINATE,
             hClientWnd,
             MAKELONG(NULL, NULL) );
```

パーソナル・コミュニケーションズの応答

パーソナル・コミュニケーションズは、次の終了メッセージで終了コマンドを確認します。

```
WM_DDE_TERMINATE
```

16 ビット環境での DDE 関数の要約

表 50 は、パーソナル・コミュニケーションズで使用できる DDE 関数を示したものです。この表では DDE 関数の名前、クライアントからパーソナル・コミュニケーションズへ送信するコマンド、およびクライアントのコマンド内の変数に使用できる値を示しています。

表 50. 16 ビット環境での DDE 関数の要約

関数名	クライアントのコマンド
Change Menu Item	`PostMessage( hServerWnd, WM_DDE_POKE, hClientWnd, MAKELONG(hData, aCHANGEMENU) );`
	hData = グローバル・メモリー・オブジェクトのハンドル
Create Menu Item	`PostMessage( hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELONG(cfFormat, aCREATEMENU) );`
	cfFormat = CF_DSPTEXT
Find Field	`PostMessage( hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELONG(cfFormat, aFIELD) );`
	cfFormat = CF_DSPTEXT
Get Keystrokes	`PostMessage( hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELONG(cfFormat, aKEYS) );`
	cfFormat = CF_DSPTEXT
Get Mouse Input	`PostMessage( hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELONG(cfFormat, aMOUSE) );`
	cfFormat = CF_TEXT | CF_DSPTEXT
Get Number of Close Requests	`PostMessage( hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELONG(cfFormat, aCLOSE) );`
	cfFormat = CF_DSPTEXT
Get Operator Information Area	`PostMessage( hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELONG(cfFormat, aOIA) );`
	cfFormat = CF_DSPTEXT

表 50. 16 ビット環境での DDE 関数の要約 (続き)

関数名	クライアントのコマンド
Get Partial Presentation Space	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELONG(cfFormat, aEPS));
	cfFormat = CF_TEXT | CF_DSPTEXT
Get Presentation Space	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELONG(cfFormat, aPS));
	cfFormat = CF_TEXT | CF_DSPTEXT
Get Session Status	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELONG(cfFormat, aSSTAT));
	cfFormat = CF_TEXT
Get System Configuration	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELONG(cfFormat, aSYSCON));
	cfFormat = CF_TEXT
Get System Formats	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELONG(cfFormat, aFORMATS));
	cfFormat = CF_TEXT
Get System Status	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELONG(cfFormat, aSTATUS));
	cfFormat = CF_TEXT
Get System SysItems	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELONG(cfFormat, aSYSITEMS));
	cfFormat = CF_TEXT
Get System Topics	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELONG(cfFormat, aTOPICS));
	cfFormat = CF_TEXT
Get Trim Rectangle	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELONG(cfFormat, aTRIMRECT));
	cfFormat = CF_TEXT
Initiate Menu Conversation	PostMessage(hServerWnd, WM_DDE_INITIATE, hClientWnd, MAKELONG(aIBM327032, SN));
	N = A から Z のセッション文字

表 50. 16 ビット環境での DDE 関数の要約 (続き)

関数名	クライアントのコマンド

Initiate Session Conversation	`SendMessage( -1,`
	`WM_DDE_INITIATE,`
	`hClientWnd,`
	`MAKELONG(aIBM327032, aSessionN) );`

	N = A から Z のセッション文字。

Initiate Structured Field Conversation	`SendMessage( -1,`
	`WM_DDE_INITIATE,`
	`hClientWnd,`
	`MAKELONG(aIBM327032, aLUN_xxxx)`

	N = A から Z のセッション文字。 xxxx = ユーザー定義ストリング。

Initiate System Conversation	`SendMessage( -1,`
	`WM_DDE_INITIATE,`
	`hClientWnd,`
	`MAKELONG(aIBM327032, aSystem) );`

Put Data to Presentation Space	`PostMessage( hServerWnd,`
	`WM_DDE_POKE,`
	`hClientWnd,`
	`MAKELONG(hData, aEPS) );`

	hData = グローバル・メモリー・オブジェクトのハンドル

Search for String	`PostMessage( hServerWnd,`
	`WM_DDE_REQUEST,`
	`hClientWnd,`
	`MAKELONG(cfFormat, aSTRING) );`

	cfFormat = CF_DSPTEXT

Send Keystrokes	`PostMessage( hServerWnd,`
	`WM_DDE_POKE,`
	`hClientWnd,`
	`MAKELONG(hData, aKEYS) );`

	hData = グローバル・メモリー・オブジェクトのハンドル

Session Execute Macro	`PostMessage( hServerWnd,`
	`WM_DDE_EXECUTE,`
	`hClientWnd,`
	`MAKELONG(NULL, hCommands) );`

	hCommands = グローバル・メモリー・オブジェクトのハンドル

Set Cursor Position	`PostMessage( hServerWnd,`
	`WM_DDE_POKE,`
	`hClientWnd,`
	`MAKELONG(hData, aSETCURSOR) );`

	hData = グローバル・メモリー・オブジェクトのハンドル

Set Mouse Intercept Condition	`PostMessage( hServerWnd,`
	`WM_DDE_POKE,`
	`hClientWnd,`
	`MAKELONG(hData, aMOUSE) );`

	hData = グローバル・メモリー・オブジェクトのハンドル

Set Presentation Space Service Condition	`PostMessage( hServerWnd,`
	`WM_DDE_POKE,`
	`hClientWnd,`
	`MAKELONG(hData, aEPSCOND) );`

	hData = グローバル・メモリー・オブジェクトのハンドル

表 50. 16 ビット環境での DDE 関数の要約 (続き)

関数名	クライアントのコマンド
Set Session Advise Condition	PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, MAKELONG(hData, aPSCOND));
	hData = グローバル・メモリー・オブジェクトのハンドル
Set Structured Field Service Condition	PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, MAKELONG(hData, aSFCOND));
	hData = グローバル・メモリー・オブジェクトのハンドル
Start Close Intercept	SendMessage(hServerWnd, WM_DDE_ADVISE, hClientWnd, MAKELONG(hOptions, aCLOSE));
	hOptions = グローバル・メモリー・オブジェクトのハンドル
Start Keystroke Intercept	SendMessage(hServerWnd, WM_DDE_ADVISE, hClientWnd, MAKELONG(hOptions, aKEYS));
	hOptions = グローバル・メモリー・オブジェクトのハンドル
Start Menu Advise	PostMessage(hServerWnd, WM_DDE_ADVISE, hClientWnd, MAKELONG(hOptions, aSELECTMENU));
	hOptions = グローバル・メモリー・オブジェクトのハンドル
Start Mouse Input Intercept	PostMessage(hServerWnd, WM_DDE_ADVISE, hClientWnd, MAKELONG(hOptions, aMOUSE));
	hOptions = グローバル・メモリー・オブジェクトのハンドル
Start Read SF	PostMessage(hServerWnd, WM_DDE_ADVISE, hClientWnd, MAKELONG(hOptions, aSF));
	hOptions = グローバル・メモリー・オブジェクトのハンドル
Start Session Advise	PostMessage(hServerWnd, WM_DDE_ADVISE, hClientWnd, MAKELONG(hOptions, aItem));
	hOptions = グローバル・メモリー・オブジェクトのハンドル aItem = OIA | PS | TRIMRECT
Stop Close Intercept	PostMessage(hServerWnd, WM_DDE_UNADVISE, hClientWnd, MAKELONG(NULL, aCLOSE));

表 50. 16 ビット環境での DDE 関数の要約 (続き)

関数名	クライアントのコマンド
Stop Keystroke Intercept	`PostMessage( hServerWnd, WM_DDE_UNADVISE, hClientWnd, MAKELONG(NULL, aKEYS) );`
Start Mouse Input Intercept	`PostMessage( hServerWnd, WM_DDE_ADVISE, hClientWnd, MAKELONG(hOptions, aMOUSE) );`
	hOptions = グローバル・メモリー・オブジェクトのハンドル
Stop Menu Advise	`PostMessage( hServerWnd, WM_DDE_UNADVISE, hClientWnd, MAKELONG(NULL, aSELECTMENU) );`
Stop Read SF	`PostMessage( hServerWnd, WM_DDE_UNADVISE, hClientWnd, MAKELONG(NULL, aSF) );`
Stop Session Advise	`PostMessage( hServerWnd, WM_DDE_UNADVISE, hClientWnd, MAKELONG(NULL, aItem) );`
	aItem = OIA | PS | TRIMRECT | NULL
Terminate Session Conversation	`SendMessage( hServerWnd, WM_DDE_TERMINATE, hClientWnd, MAKELONG(NULL, NULL) );`
Terminate Menu Conversation	`SendMessage( hServerWnd, WM_DDE_TERMINATE, hClientWnd, MAKELONG(NULL, NULL) );`
Terminate Structured Field Conversation	`SendMessage( hServerWnd, WM_DDE_TERMINATE, hClientWnd, MAKELONG(NULL, NULL) );`
Terminate System Conversation	`SendMessage( hServerWnd, WM_DDE_TERMINATE, hClientWnd, MAKELONG(NULL, NULL) );`
Write SF	`PostMessage( hServerWnd, WM_DDE_POKE, hClientWnd, MAKELONG(hData, aSF) );`
	hData = グローバル・メモリー・オブジェクトのハンドル

付録 F. REXX EHLLAPI 関数

この付録は、EHLLAPI を使って REXX 言語のアプリケーション・プログラムを作成するプログラマーのために書かれたものです。プログラマーは REXX コマンド言語に精通している必要があります。ここでは REXX EHLLAPI 関数の概説が示されていますが、それらの関数はアルファベット順に説明されており、それぞれに詳細な記述が付記されています。 REXX およびサンプル・プログラムを使ったアプリケーション作成については、関数の説明の後に記載します。

On Windows プラットフォームでは、REXX アプリケーションは Windows 用の IBM オブジェクト REXX を使用する必要があります。

REXX EHLLAPI での関数呼び出しと戻り値の概説

REXX EHLLAPI は、REXX 関数または REXX サブルーチンのいずれかで呼び出されます。 REXX 関数は 1 つの値を返しますが、それは変数または次のような戻りコードに割り当てられます。

rc=**HLLAPI(** *function-string* [**,***parameters*] **)**

REXX サブルーチンは戻り値を `Result` と呼ばれる特定の REXX 変数の中に配置します。下記にそれを示しています。

call HLLAPI *function-string* [**,***parameters*]

インストール

REXX EHLLAPI 実行ファイル (SAAHLAPI.DLL) は、 IBM パーソナル・コミュニケーションズとともにインストールされます。

EHLLAPI は外部機能として用意されており、使用するときにだけメモリーにロードされます。REXX EHLLAPI 関数を使用できるようにするために、REXX アプリケーション・プログラムには下記のステートメントを含める必要があります。

```
if rxfuncquery('hllapi') then call rxfuncadd 'hllapi','saahlapi','hllapisrv'
```

REXX EHLLAPI は、パラメーター・セットと共に使用される単一の 関数名を提供します。その関数名は上記の rxfuncadd 呼び出しの最初のパラメーターです。デフォルト値は **HLLAPI** です。

規則

各 REXX EHLLAPI 関数について次の点を記述します。

- 関数名
- 前提呼び出し
- 指定構文
- 指定パラメーター
- 戻りパラメーター
- 追加情報

関数名 関数の名前と簡単な解説を示します。

前提呼び出し 関数を使用する前に、アプリケーション・プログラムから呼び出しておく必要のある関数をリストします。なし という語は、前提呼び出しが必要ないことを示しています。すべての REXX EHLLAPI 関数に共通の前提呼び出しを『関数の前提呼び出しの概要』に示します。

指定構文 関数呼び出しの構文を図示しています。詳細については、 471 ページの『REXX EHLLAPI での関数呼び出しと戻り値の概説』を参照してください。

指定パラメーター REXX EHLLAPI 関数を呼び出すために、ユーザー・プログラムが定義しなければならないパラメーターをリストします。

どの関数呼び出しでも最初のパラメーターとして関数名を指定しますが、プログラムが定義しなければならないパラメーターの数は、呼び出す関数によって違います。指定パラメーターの形式については、各関数の説明を参照してください。

戻りパラメーター 値 (戻りコードか実際のデータのどちらか) を単一の変数に代入します。REXX EHLLAPI を関数として使用する場合、これらの値は受信変数に代入されます。 REXX EHLLAPI をサブルーチンとして使用する場合、その値は *result* と呼ばれる特殊変数に代入されます。戻り値の形式については、各関数の説明を参照してください。

追加情報 関数に関する必要な技術情報を示します。

関数の前提呼び出しの概要

表 51 では、REXX EHLLAPI 関数ごとに前提呼び出しを示しています。前提呼び出しはアプリケーション・プログラム使用時に必要となります。

表 51. 関数の前提呼び出し

関数	前提呼び出し
Change_Switch_Name	Connect_PM
Change_Window_Name	Connect_PM
Connect	なし
Connect_PM	なし
Convert_Pos	なし
Copy_Field_To_Str	Connect
Copy_OIA	Connect
Copy_PS	Connect
Copy_PS_To_Str	Connect
Copy_Str_To_Field	Connect
Copy_Str_To_PS	Connect
Disconnect	Connect
Disconnect_PM	Connect_PM
Find_Field_Len	Connect
Find_Field_Pos	Connect
Get_Key	Start_Keystroke_Intercept
Get_Window_Status	Connect_PM
Intercept_Status	Start_Keystroke_Intercept

表 51. 関数の前提呼び出し (続き)

関数	前提呼び出し
Lock_PMSVC	Connect_PM
Lock_PS	Connect
Pause	なし
Query_Close_Intercept	Start_Close_Intercept
Query_Cursor_Pos	Connect
Query_Emulator_Status	なし
Query_Field_Attr	Connect
Query_Host_Update	Start_Host_Notify
Query_Session_List	なし
Query_Session_Status	なし
Query_Sessions	なし
Query_System	なし
Query_Window_Coord	Connect_PM
Query_Workstation_Profile	なし
Receive_File	なし
Release	Connect
Reserve	Connect
Reset_System	なし
Search_Field	Connect
Search_PS	Connect
Send_File	なし
Sendkey	Connect
Set_Cursor_Pos	Connect
Set_Session_Parms	なし
Set_Window_Status	Connect_PM
Start_Close_Intercept	なし
Start_Communication	なし
Start_Host_Notify	なし
Start_Keystroke_Intercept	なし
Start_Session	なし
Stop_Close_Intercept	Start_Close_Intercept
Stop_Communication	なし
Stop_Host_Notify	Start_Host_Notify
Stop_Keystroke_Intercept	Start_Keystroke_Intercept
Stop_Session	なし
Wait	Connect

EHLLAPI と REXX EHLLAPI 関数の概要

表 52 は、それぞれの EHLLAPI と対応する REXX EHLLAPI 関数を示しています。

表 52. EHLLAPI および REXX EHLLAPI 関数

EHLLAPI	REXX EHLLAPI
CHANGE SWITCH LIST LT NAME (105)	Change_Switch_Name
CHANGE PS WINDOW NAME (106)	Change_Window_Name
CONNECT PRESENTATION SPACE (1)	Connect
CONNECT PM WINDOW SERVICES (101)	Connect_PM
CONVERT POSITION or CONVERT ROWCOL (99)	Convert_Pos
COPY FIELD TO STRING (34)	Copy_Field_To_String
COPY OIA (13)	Copy_OIA
COPY PRESENTATION SPACE (5)	Copy_PS
COPY PRESENTATION SPACE TO STRING (8)	Copy_PS_To_Str
COPY STRING TO FIELD (33)	Copy_Str_To_Field
COPY STRING TO PRESENTATION SPACE (15)	Copy_Str_To_PS
DISCONNECT PRESENTATION SPACE (2)	Disconnect
DISCONNECT PM WINDOW SERVICES (102)	Disconnect_PM
FIND FIELD LENGTH (32)	Find_Field_Len
FIND FIELD POSITION (31)	Find_Field_Pos
GET KEY (51)	Get_Key
PAUSE (18)	Pause
PM WINDOW STATUS (104)	Get_Window_Status and Set_Window_Status
POST INTERCEPT STATUS (52)	Intercept_Status
LOCK PMSVC API (61)	Lock_PMSVC
LOCK PRESENTATION SPACE API (60)	Lock_PS
QUERY CLOSE INTERCEPT (42)	Query_Close_Intercept
QUERY CURSOR LOCATION (7)	Query_Cursor_Pos
QUERY FIELD ATTRIBUTE (14)	Query_Field_Attr
QUERY HOST UPDATE (24)	Query_Host_Update
QUERY PM WINDOW COORDINATES (103)	Query_Window_Coord
QUERY SESSION STATUS (22)	Query_Session_Status
QUERY SESSIONS (10)	Query_Sessions
QUERY SYSTEM (20)	Query_System
RECEIVE FILE (91)	Receive_File
RELEASE (12)	Release
RESERVE (11)	Reserve

表 52. *EHLLAPI および REXX EHLLAPI 関数 (続き)*

EHLLAPI	**REXX EHLLAPI**
RESET SYSTEM (21)	Reset_System
SEARCH FIELD (30)	Search_Field
SEARCH PRESENTATION SPACE (6)	Search_PS
SEND FILE (90)	Send_File
SEND KEY (3)	Sendkey
SET CURSOR (40)	Set_Cursor_Pos
SET SESSION PARAMETERS (9)	Set_Session_Parms
START CLOSE INTERCEPT (41)	Start_Close_Intercept
START HOST NOTIFICATION (23)	Start_Host_Notify
START KEYSTROKE INTERCEPT (50)	Start_Keystroke_Intercept
STOP CLOSE INTERCEPT (43)	Stop_Close_Intercept
STOP HOST NOTIFICATION (25)	Stop_Host_Notify
STOP KEYSTROKE INTERCEPT (53)	Stop_Keystroke_Intercept
WAIT (4)	Wait
pcsStartSession	Start_Session
pcsStopSession	Stop_Session
pcsConnectSession	Start_Communication
pcsDisconnectSession	Stop_Communication
pcsQuerySessionList	Query_Session_List
pcsQueryEmulatorStatus	Query_Emulator_Status
pcsQueryWorkstationProfile	Query_Workstation_Profile

Change_Switch_Name

Change_Switch_Name 関数は、ウィンドウ・タイトル・バーにリストされるセッション名を変更またはリセットします。

前提呼び出し

Connect_PM

結果

Change_Switch_Name の構文は次のとおりです。

HLLAPI('Change_switch_name', session_id**,** *type* **[,** *new_name* **])**

指定パラメーター

関数名: **Change_Switch_Name**

session_id: タスク・リスト上で名前を変更するセッションの 1 文字短縮名。

type: 次のいずれかを示します。

- **Set** は、セッション *session_id* のセッション名 (タイトル) を *new_name* に変更します。
- **Reset** は、元のセッション名 (タイトル) を復元します。

注: Set または **Reset** の先頭文字だけでも有効です。

戻りパラメーター

戻りコード	説明
0	**Change_Switch_Name** 関数が正常に終了しました。
1	プログラムは現在、ホスト・セッションと接続されていません。
2	パラメーター指定でエラーが発生しました。
9	システム・エラーが発生しました。
12	セッションが停止しました。

Change_Window_Name

Change_Window_Name 関数は、ホスト・ウィンドウのタイトル・バーにリストされているセッション名を変更またはリセットします。

前提呼び出し

Connect_PM

結果

Change_Window_Name 関数の構文は次のとおりです。

HLLAPI('Change_window_name', *session_id*, *type* [, *new_name*])

指定パラメーター

関数名: **Change_Window_Name**

session_id: タイトル・バー上で名前を変更するセッションの 1 文字短縮名。

type: 次のいずれかを示します。

- **Set** は、セッション *session_id* のウィンドウ名 (タイトル) を *new_name* に変更します。
- **Reset** は、元のウィンドウ名 (タイトル) を復元します。

注: Set または **Reset** の先頭文字だけでも有効です。

戻りパラメーター

戻りコード	説明
0	**Change_Window_Name** 関数が正常に終了しました。
1	プログラムは現在、ホスト・セッションと接続されていません。
2	パラメーター指定でエラーが発生しました。
9	システム・エラーが発生しました。
12	セッションが停止しました。

Connect

Connect 関数は、REXX アプリケーション・プログラムをホストの表示スペースに接続します。

前提呼び出し

この関数には前提呼び出しはありません。

結果

Connect 関数の構文は次のとおりです。

HLLAPI('Connect', *session_id* **)**

指定パラメーター

関数名: **Connect**
session_id: 接続するセッション・ウィンドウの 1 文字短縮名。

戻りパラメーター

戻りコード	説明
0	**Connect** 関数が正常に終了しました。
1	無効なホスト表示スペース ID が指定されました。指定されたセッションが存在しないか、または、論理プリンター・セッションです。さらにこの戻りコードは、DDE/EHLLAPI 用の API Setting がオンに設定されていないことを示している場合があります。
4	接続は正常に行われましたが、セッションは使用中です。
5	接続は正常に行われましたが、セッションはロックされています (入力禁止)。
9	システム・エラーが発生しました。
11	セッションがすでに別のシステム関数で使用されています。

Connect_PM

Connect_PM 関数は、REXX アプリケーション・プログラムを表示スペース・ウィンドウに接続します。

前提呼び出し

この関数には前提呼び出しはありません。

結果

Connect_PM 関数の構文は次のとおりです。

HLLAPI('Connect_PM', *session_id* **)**

指定パラメーター

関数名: **Connect_PM**
session_id: 接続したいセッション・ウィンドウの 1 文字短縮名。

戻りパラメーター

戻りコード	説明
0	**Connect_PM** 関数が正常に終了しました。
1	無効なホスト表示スペース ID が指定されました。さらにこの戻りコードは、DDE/EHLLAPI 用の API Setting がオンに設定されていないことを示している場合があります。
9	システム・エラーが発生しました。
10	この関数はエミュレーション・プログラムではサポートされていません。
11	セッションがすでに別のシステム関数で使用されています。

Convert_Pos

Convert_Pos 関数は、ホストの表示スペースにおける位置を示す値をディスプレイの行と桁座標に変換するか、ディスプレイの行と桁座標を (特定のセッション ID で指定された) ホストの表示スペースにおける位置を示す値に変換します。

注: 行と桁の変換を指定する場合、*桁* は 2 番目のパラメーターです。有効な行と桁の値は、このセッションの構成時に指定した値です。例えば、24 行と 80 桁のセッションでは、位取りは 1 ～ 1920 の値となります。

前提呼び出し

この関数には前提呼び出しはありません。

結果

Convert_Pos 関数の構文は次のとおりです。

HLLAPI('Convert_pos', *session_id*, *column* | *position* [, *row*])

指定パラメーター

関数名: **Convert_Pos**
session_id: セッションの 1 文字短縮名。
column: 変換する桁。行と一緒に指定する必要があります。
row: 変換する行。
position: 変換する位取り。

戻りパラメーター

次の値は、行と桁の変換を要求する場合に有効です。

戻りコード	説明
0	指定された行または桁が表示スペースの外にあります。
n	指定された行と桁の位取り。次に例を示します。 `HLLAPI ('Convert_pos', 'a', 10, 2) = '170'` これは、桁 10、行 2 を 24x80 の表示スペースの 170 に変換します。

下記の値は、位取り変換が要求されている場合に有効です。

戻りコード	説明
0	指定された位取りが表示スペースの外にあります。
c r	**c** は桁番号、 **r** は行番号を示します。次に例を示します。 `HLLAPI ('Convert_pos', 'a', 170) = '10 2'` これは、位取り 170 を 24x80 表示スペースの 桁 10、行 2 に変換します。

Copy_Field_To_Str

Copy_Field_To_Str 関数は、あるターゲット・フィールドからデータ・ストリングに文字を転送します。 **Find_Field_Pos** および **Find_Field_Len** 関数を使用して、ターゲットとその値の長さを判別することができます。

前提呼び出し

Connect

結果

Copy_Field_To_Str 関数の構文は次のとおりです。

HLLAPI('Copy_field_to_str', *pos*, *length*)

指定パラメーター

関数名: **Copy_Field_To_Str**
pos: コピーするターゲット・フィールド。
length: 長さ (バイト数)、コピー先データ・ストリング長。

戻りパラメーター

戻りコード	説明
''	ヌル。*pos* でフィールド・データが検索されなかった、または *pos* が無効です。
データ	戻されるデータ・ストリングの内容は、**Set_Session_Parms** 関数の拡張属性バイト (EAB) 値によって決まります。 EAB をオフに設定した場合は、表示スペースのテキストだけが戻されます。 EAB をオンに設定すると、表示される各バイトに対して 2 バイトが戻されます。最初のバイトには EAB 値が、そして 2 番目のバイトにはテキスト・データが含まれます。 **COPY FIELD TO STRING** 関数と共に、 (9) 関数の EAD オプションを使用すると、2 バイト EAD を戻すことができます。 EAB オプションなしで EAD を使用した場合、各文字のあとに EAD が戻されます。EAB オプションを指定すると、EAB のあとに EAD が戻されます。

Copy_OIA

この関数は、接続セッションからのオペレーター情報域 (OIA) の内容を戻します。

前提呼び出し

Connect

結果

Copy_OIA 関数の構文は次のとおりです。

HLLAPI('Copy_OIA')

指定パラメーター

この関数用に提供されているパラメーターはありません。

戻りパラメーター

戻りコード	説明
''	ヌル。接続されていないか、その他のエラーが生じました。
データ	OIA データの 104 バイトのコピー。 57 ページの『Copy OIA (13)』を参照。

Copy_PS

Copy_PS 関数は、現行接続セッションの表示スペースの全内容を戻します。

前提呼び出し

Connect

結果

Copy_PS 関数の構文は次のとおりです。

HLLAPI('Copy_PS')

指定パラメーター

この関数用に提供されているパラメーターはありません。

戻りパラメーター

戻りコード	説明
''	ヌル。接続されていないか、その他のエラーが生じました。
データ	戻されたデータ・ストリングの内容は **Set_Session_Parms** 関数の EAB 値によって決まります。 EAB をオフに設定した場合は、表示スペースのテキストだけが戻されます。 EAB をオンに設定すると、表示される各バイトに対して 2 バイトが戻されます。最初のバイトには EAB 値が、そして 2 番目のバイトにはテキスト・データが含まれます。 これらのバイトがスペース文字として戻されることがありますが、それは、コピーの開始位置が 2 バイト文字の 2 番目のバイトであるか、または終了位置が 2 バイト文字の最初のバイトである場合です。

追加情報

COPY_PS では、EHLLAPI に対する 2 つの呼び出しが生じます。最初の呼び出しは表示スペースのサイズを判別するための **QUERY_SESSIONS**、 2 番目の呼び出しはバッファーがオーバーフローしないよう **COPY_PS** の代わりに使用する **COPY_PS_TO_STRING** です。

COPY_PS_TO_STRING は、 **QUERY_SESSIONS** と **COPY_PS_TO_STRING** の呼び出しの間に表示スペースのサイズが変更される可能性がわずかながらあるために使用します。

注: 表示スペースのサイズが増えた場合には、 **QUERY_SESSIONS** で戻されるバイト数だけがコピーされます。そのサイズが小さくなったときは、現行のサイズを超える文字が無視されます。

Copy_PS_To_Str

Copy_PS_To_Str 関数は、現行接続セッションからのデータをデータ・ストリングにコピーします。

前提呼び出し

Connect

結果

Copy_PS_To_Str 関数の構文は次のとおりです。

HLLAPI('Copy_PS_to_str', *pos*, *length* **)**

指定パラメーター

関数名: **Copy_PS_To_Str**
pos: コピーするターゲット・フィールド。
length: 長さ (バイト数)、コピー先データ・ストリング長。

注: EAB をオンに設定した場合は、ターゲット・フィールドの値は 2 倍にしないでください。EAD をオンに設定すると、REXX EHLLAPI は自動的にそれを行います。

戻りパラメーター

戻りコード	説明
''	ヌル。接続されていないか、その他のエラーが生じました。
データ	戻されたデータ・ストリングの内容は **Set_Session_Parms** 関数の EAB 値によって決まります。 EAB をオフに設定した場合は、表示スペースのテキストだけが戻されます。 EAB をオンに設定すると、表示される各バイトに対して 2 バイトが戻されます。最初のバイトには EAB 値が、そして 2 番目のバイトにはテキスト・データが含まれます。

Copy_Str_To_Field

Copy_Str_To_Field 関数は、接続セッションのターゲット・フィールド・ロケーションで指定されたフィールドに文字ストリングをコピーします。

前提呼び出し

Connect

結果

Copy_Str_To_Field 関数の構文は次のとおりです。

HLLAPI('Copy_str_to_field', *string*, *pos* **)**

指定パラメーター

関数名: **Copy_Str_To_Field**
string: ターゲット・フィールドに転送するデータを含むストリング。
pos: コピーするターゲット・フィールド。

戻りパラメーター

戻りコード	説明
0	**Copy_Str_To_Field** 関数が正常に終了しました。
1	プログラムは現在、ホスト・セッションと接続されていません。
2	パラメーター・エラーです。
5	コピー先フィールドが保護または入力禁止になっているか、あるいは、無効なデータ (フィールド属性など) がコピー先フィールドに送られました。
6	コピーは完了しましたが、データは切り捨てられました。
7	*pos* パラメーターが無効でした。
9	システム・エラーが発生しました。
24	画面のフィールド (不定様式) がありません。

Copy_Str_To_PS

Copy_Str_To_PS 関数は、*pos* パラメーターにより指定されたホストの表示スペースの中に文字ストリングをコピーします。

前提呼び出し

Connect

結果

Copy_Str_To_PS 関数の構文は次のとおりです。

HLLAPI('Copy_str_to_PS', *string*, *pos* **)**

指定パラメーター

関数名: **Copy_Str_To_PS**
string: 表示スペースに転送するデータを含むストリング。
pos: コピーする表示スペース

戻りパラメーター

戻りコード	説明
0	**Copy_Str_To_PS** 関数が正常に終了しました。
1	プログラムは現在、ホスト・セッションと接続されていません。
2	パラメーター・エラーです。
5	コピー先フィールドが保護または入力禁止になっているか、あるいは、無効なデータ (フィールド属性など) がコピー先フィールドに送られました。
6	コピーは完了しましたが、データは切り捨てられました。
7	*pos* パラメーターが無効でした。
9	システム・エラーが発生しました。

Disconnect

Disconnect 関数は、ユーザーのアプリケーション・プログラムを現在接続されているセッションから切り離します。

前提呼び出し

Connect

結果

Disconnect 関数の構文は次のとおりです。

HLLAPI('Disconnect')

指定パラメーター

この関数用に提供されているパラメーターはありません。

戻りパラメーター

戻りコード	説明
0	**Disconnect** 関数が正常に終了しました。
1	プログラムは現在、ホストの表示スペースと接続されていません。
9	システム・エラーが発生しました。

Disconnect_PM

Disconnect_PM 関数は、セッション・ウィンドウから切り離します。

前提呼び出し

Connect_PM

結果

Disconnect_PM 関数の構文は次のとおりです。

HLLAPI('Disconnect_PM', *session_id* **)**

指定パラメーター

関数名: **Disconnect_PM**
session_id: 接続したいセッションの 1 文字短縮名。

戻りパラメーター

戻りコード	説明
0	**Disconnect_PM** 関数が正常に終了しました。
1	プログラムは現在、Windows Service と接続されていません。
9	システム・エラーが発生しました。

Find_Field_Len

Find_Field_Len 関数は、 *search_option* パラメーターで指定された属性と共にターゲット・フィールドの長さを戻します。

前提呼び出し

Connect

結果

Find_Field_Len 関数の構文は次のとおりです。

HLLAPI('Find_field_len', *search_option*, *pos* **)**

指定パラメーター

関数名: **Find_Field_Len**
search_option: 次の表を参照してください。
pos: コピーするターゲット・フィールド。

次の *search_option* 値が有効です。

値	説明
'␣␣' または 'T␣'	現行のフィールド (カーソルが置かれているフィールド)。
'N␣'	次の保護または無保護フィールド。
'P␣'	前の保護または無保護フィールド。
'NP'	次の保護フィールド。
'NU'	次の無保護フィールド。
'PP'	前の保護フィールド。
'PU'	前の無保護フィールド。

戻りパラメーター

戻りコード	説明
0	指定されたフィールドは検索されませんでした。
データ	指定したフィールドの長さ。

Find_Field_Pos

Find_Field_Pos 関数は、 *search_option* パラメーターで指定された属性と一緒にターゲット・フィールドの位置を戻します。

前提呼び出し

Connect

結果

Find_Field_Pos 関数の構文は次のとおりです。

HLLAPI('Find_field_pos', *search_option*, *pos* **)**

指定パラメーター

関数名: **Find_Field_Pos**
search_option: 次の表を参照してください。
pos: コピーするターゲット・フィールド。

次の *search_option* 値が有効です。

値	説明
'␣␣' または 'T␣'	現行のフィールド (カーソルが位置付けられているフィールド)。
'N␣'	次のフィールド。保護または無保護。
'P␣'	前のフィールド。保護、無保護のいずれでも可。
'NP'	次の保護フィールド。
'NU'	次の無保護フィールド。
'PP'	前の保護フィールド。
'PU'	前の無保護フィールド。

戻りパラメーター

戻りコード	説明
0	指定されたフィールドは検索されませんでした。
データ	指定したフィールドの位置。

Get_Key

Get_Key 関数は、*session_id* がブランクのときに、指定された *session_id* からの、つまり現行接続セッションからのキー・ストロークをアプリケーション・プログラムが代行受信できるようにします。その場合、プログラムはキー・ストロークが使用可能になるまで待機します。

前提呼び出し

Start_Keystroke_Intercept

結果

Get_Key 関数の構文は次のとおりです。

HLLAPI('Get_key', *session_id* **)**

指定パラメーター

関数名: **Get_Key**
session_id: セッションの 1 文字短縮名。

戻りパラメーター

戻りコード	説明
''	ヌル。エラーまたは *session_id* に接続されませんでした。
データ・ストリング	データ・ストリングの内容は、3270 または 5250 セッションのオペレーターにより押されるキーによって異なります。 155 ページの『キーボードの略号』を参照。

注: @ (エスケープ) 文字は、**Set_Session_Parms** 関数に ESC= を指定することにより設定されます。

追加情報

キー・ストロークの代行受信が (**Start_Keystroke_Intercept** 関数を介して) 活動中の場合、次のタスクが実行されるまではどのキー・ストロークも接続されたセッションに送信されません。

1. **Get_Key** 関数を指定して、代行受信バッファーのキー・ストロークを除去する。
2. **Intercept_Status** 関数を指定して、キー・ストロークを受諾または拒否するようにする。 *Accept* を指定すると、キー・ストロークは **Sendkey** 関数により接続されたセッションに送信されます。 *Reject* を指定すると、キー・ストロークは破棄されます。

Get_Window_Status

Get_Window_Status 関数は、現行のウィンドウ状況を、ASCII 文字 (16 進) で戻します。

前提呼び出し

Connect_PM

結果

Get_Window_Status 関数の構文は次のとおりです。

HLLAPI('Get_window_status', *session_id* **)**

指定パラメーター

関数名: **Get_Window_Status**
session_id: セッションの 1 文字短縮名。

戻りパラメーター

戻りコード	説明
''	ヌル。ウィンドウに接続されていません。詳細については **Connect_PM** 関数を参照してください。
0008	ウィンドウが可視状態です。
0010	ウィンドウは不可視状態です。
0080	ウィンドウは活動状態です。
0100	ウィンドウは非活動状態です。
0400	ウィンドウは最小化されています。
0800	ウィンドウは最大化されています。
注: 上の状況が 1 つ以上当てはまる場合、戻りコードは 1 つにまとめられます。例えば、ウィンドウが可視状態 (0008)、非活動状態 (0100)、最大化 (0800) 状態の場合、戻りコードは 0908 になります。	

Intercept_Status

Intercept_Status 関数は、 **Get_Key** 関数で入力されたキー・ストロークが受諾されたか拒否されたかをセッションに通知します。

前提呼び出し

Start_Keystroke_Intercept

結果

Intercept_Status 関数の構文は次のとおりです。

HLLAPI('Intercept_status', *session_id*, *status*)

指定パラメーター

関数名: **Intercept_Status**
session_id: セッションの 1 文字短縮名。
status: 次の表を参照してください。

値	説明
'A'	キー・ストロークが受諾されました。
'R'	キー・ストロークが拒否されました。ビープ音のシグナル。

戻りパラメーター

戻りコード	説明
0	**Interrupt_Status** 関数が正常に終了しました。
1	表示スペースが無効でした。
8	前の **Start_Keystroke_Intercept** 関数が活動中ではありません。
9	システム・エラーが発生しました。

Lock_PMSVC

この関数は表示スペースのウィンドウをロックまたはアンロックします。

前提呼び出し

Connect_PM

結果

Lock_PMSVC 関数の構文は次のとおりです。

HLLAPI('Lock_PMSVC', *session_id*, *status*, *queue_option* **)**

指定パラメーター

関数名: **Lock_PMSVC**
session_id: セッションの 1 文字短縮名。
status: 次の表を参照してください。

値	説明
'L'	表示スペースのウィンドウをロックします。
'U'	表示スペースのウィンドウをアンロックします。

queue_option: 次の表を参照してください。

値	説明
'R'	すぐに戻る。
'Q'	表示スペースのウィンドウがすでにロックされている場合にキューに入る (ロック・モードの場合に限る)。

戻りパラメーター

戻りコード	説明
0	**Lock_PMSVC** 関数が正常に終了しました。
1	ホストの無効な表示スペースが指定されたかまたは接続されていません。
2	パラメーター指定でエラーが発生しました。
9	システム・エラーが発生しました。
43	API がすでに別の EHLLAPI アプリケーションにロックされています (LOCK モードの場合)。または API がロックされていません (UNLOCK モードの場合)。

Lock_PS

Lock_PS は、表示スペースをロックまたはアンロックします。

前提呼び出し

Connect

結果

Lock_PS 関数の構文は次のとおりです。

HLLAPI('Lock_PS', *session_id*, *status*, *queue_option* **)**

指定パラメーター

関数名: **Lock_PS**
session_id: セッションの 1 文字短縮名。
status: 次の表を参照してください。

値	説明
'L'	表示スペースをロックします。
'U'	表示スペースをアンロックします。

queue_option: 次の表を参照してください。

値	説明
'R'	すぐに戻る。
'Q'	表示スペースがすでにロックされている場合にキューに入る (ロック・モードの場合に限る)。

戻りパラメーター

戻りコード	説明
0	**Lock_PS** 関数が正常に終了しました。
1	表示スペースが無効でした。
2	パラメーター指定でエラーが発生しました。
9	システム・エラーが発生しました。
43	API がすでに別の EHLLAPI アプリケーションにロックされています (LOCK モードの場合)。または API がロックされていません (UNLOCK モードの場合)。

Pause

Pause 関数は、 *n* ½ 秒単位で一時停止を発生させます。

Set_Session_Parms 関数が IPAUSE に設定され、 **Start_Host_Notify** 関数がすでに呼び出されている場合、ホスト画面を更新すると一時停止は終了します。*sessname* が指定され、IPAUSE が設定されている場合は、特定のセッションを更新したときだけ一時停止は割り込まれます。それ以外の場合は、どの接続セッションを更新しても、一時停止は割り込まれます (IPAUSE が設定されている場合)。

前提呼び出し

この関数には前提呼び出しはありません。

結果

Pause 関数の構文は次のとおりです。

HLLAPI('Pause', *n* [**,** *sessname*] **)**

指定パラメーター

関数名: **Pause**
n: 一時停止の時間。
sessname: 任意指定パラメーター。次の表を参照してください。

値	説明
'X#'	X は短縮セッション ID (short-session_id) 名です。# は示されているとおりにコード化します。

戻りパラメーター

戻りコード	説明
0	待機時間が終了しました。
9	システム・エラーが発生しました。
26	ホスト・セッションの表示スペースまたは OIA がすでに更新されています。詳細については、 501 ページの『Query_Host_Update』を参照してください。

Query_Close_Intercept

Query_Close_Intercept 関数は、セッションからクローズ要求が出されたかどうかを判別します。

前提呼び出し

Start_Close_Intercept

結果

Query_Close_Intercept 関数の構文は次のとおりです。

HLLAPI('Query_close_intercept', *session_id* **)**

指定パラメーター

関数名: **Query_Close_Intercept**
session_id: ホスト・セッションの 1 文字短縮名。

戻りパラメーター

戻りコード	説明
0	クローズ割り込みは生じませんでした。
1	プログラムは現在、ホスト・セッションと接続されていません。
2	パラメーター指定でエラーが発生しました。
8	前の **Start_Close_Intercept** 関数がホストの表示スペースに対して呼び出されていません。
9	システム・エラーが発生しました。
12	セッションが停止しました。
26	最後の **Query_Close_Intercept** 関数呼び出し以来、クローズ割り込みが生じました。

Query_Cursor_Pos

Query_Cursor_Pos 関数は、現行接続セッションのカーソル位置を戻します。

前提呼び出し

Connect

結果

Query_Cursor_Pos 関数の構文は次のとおりです。

HLLAPI('Query_cursor_pos')

指定パラメーター

この関数用に提供されているパラメーターはありません。

戻りパラメーター

戻りコード	説明
0	プログラムは現在、ホスト・セッションと接続されていません。
データ	カーソル位置。

Query_Emulator_Status

Query_Emulator_Status 関数は、指定されたホスト・セッションの状況を戻します。

前提呼び出し

この関数には前提呼び出しはありません。

結果

Query_Emulator_Status 関数の構文は次のとおりです。

HLLAPI('Query_emulator_status', *session_id* **)**

指定パラメーター

関数名: **Query_Emulator_Status**
session_id: 照会されるセッションのセッション文字 (A-Z) を指定します。

戻りパラメーター

戻りコード	説明
' '	ヌル。エラーが発生しました。
1	セッションが開始されています。
2	セッションが開始され、ホストへの接続が要求されました。
3	セッションが開始され、接続が要求され、API がセッション用に使用可能になっています。

Query_Field_Attr

Query_Field_Attr 関数は、現行接続セッション内のフィールド属性を 16 進で戻します。

前提呼び出し

Connect

結果

Query_Field_Attr 関数の構文は次のとおりです。

HLLAPI('Query_field_attr', *pos* **)**

指定パラメーター

関数名: **Query_Field_Attr**
pos: コピーするターゲット・フィールド。

戻りパラメーター

戻りコード	説明
1	プログラムは現在、ホスト・セッションと接続されていません。
データ	属性バイト (X'C0' 以上の印刷可能な 16 進文字)。

Query_Host_Update

Query_Host_Update 関数は、セッションの OIA または表示スペースがすでに更新されたかどうかを判別します。

前提呼び出し

Start_Host_Notify

結果

Query_Host_Update 関数の構文は次のとおりです。

HLLAPI('Query_host_update', *session_id* **)**

指定パラメーター

関数名: **Query_Host_Update**
session_id: セッションの 1 文字短縮名。

戻りパラメーター

戻りコード	説明
0	最後の呼び出し以来、更新はされていません。
1	ホストの無効な表示スペースが指定されました。
8	前の **Start_Host_Notify** 関数がホストの表示スペース ID に対して呼び出されていません。
9	システム・エラーが発生しました。
21	OIA が更新されました。
22	表示スペースが更新されました。
23	OIA およびホストの表示スペースが更新されました。
44	印刷は、プリンター・セッションで完了しています。

Query_Session_List

Query_Session_List 関数は、現行の各ホスト・セッションの 2 バイト項目を戻します。

前提呼び出し

この関数には前提呼び出しはありません。

結果

Query_Session_List 関数の構文は次のとおりです。

HLLAPI('Query_session_list',)

指定パラメーター

この関数用に提供されているパラメーターはありません。

戻りパラメーター

戻りコード	説明	
' '	ヌル。接続されていません。	
0	セッションが開始されていません。	
データ	**位置**	**定義**
	1	短縮セッション ID。
	2	次のいずれかの値: **1** セッションが開始されています。 **0** セッションが開始され、ホストへの接続が要求されました。 **3** セッションが開始され、接続が要求され、API がセッション用に使用可能になっています。

Query_Session_Status

Query_Session_Status 関数は、ホスト・セッションから、または *session_id* がブランクの場合は現行接続セッションから状況に関するさまざまな情報を戻します。

前提呼び出し

この関数には前提呼び出しはありません。

結果

Query_Session_Status 関数の構文は次のとおりです。

HLLAPI('Query_session_status', *session_id* **)**

指定パラメーター

関数名: **Query_Session_Status**
session_id: セッションの 1 文字短縮名。

戻りパラメーター

戻りコード	説明	
''	ヌル。接続されていません。	
データ	**位置**	**定義**
	1	短縮セッション ID。
	2 ～ 9	セッションのロング・ネーム。
	10	セッション・タイプ。下記のとおりです。 • D=3270 ホスト • E=3270 プリンター • F=5250 ホスト • G=5250 プリンター • H=ASCII
	11	セッションの特性。次に説明するセッション特性バイトを 2 進数で表します。 **0** EAB **1** PSS **2-7** 予約済み。 ビット 0 (EAB) = 0 の場合、セッションは基本属性を持っており、ビット 0 (EAB) = 1 の場合は拡張属性を持っています。ビット 1 (PSS) = 0 の場合、セッションはプログラム式シンボルをサポートしていません。ビット 1 (PSS) = 1 の場合、セッションはプログラム式シンボルをサポートしています。

戻りコード	説明	
	12 〜 13	ホストの表示スペース内の行数。これは 2 進数であり、表示形式ではありません。セッション・タイプが E または G の場合、この値は 0 です。
	14 〜 15	ホストの表示スペース内の桁数。これは 2 進数であり、表示形式ではありません。セッション・タイプが E または G の場合、この値は 0 です。
	16 〜 17	ホストのコード・ページ番号。2 進数で表しています。
	18	予約済み。

注: ストリングからその 10 進数値を得るために、最後の 3 つのフィールド (行、桁、コード・ページ) を解析した後、c2d(reverse(x)) を使用してそのバイトを反転してください。

Query_Sessions

Query_Sessions 関数は、エラーが発生したときに、それぞれの構成済みセッションに関する 12 バイトの説明、またはヌル (’’) を戻します。

前提呼び出し

この関数には前提呼び出しはありません。

結果

Query_Sessions 関数の構文は下記のとおりです。

HLLAPI(’Query_sessions’)

指定パラメーター

この関数用に提供されているパラメーターはありません。

戻りパラメーター

戻りコード	**説明**	
’’	ヌル。エラーが発生しました。	
データ	**位置**	**定義**
	1	短縮セッション ID。
	2 〜 9	セッションのロング・ネーム。
	10	接続タイプ H=host
	11 〜 12	表示スペースのサイズ。これは 2 進数であり、表示形式ではありません。セッション・タイプが印刷セッションの場合、この値は 0 です (126 ページの『Query Sessions (10)』を参照)。

注: ストリングからその 10 進数値を得るために、最後のフィールド (pssize) を解析した後、c2d(reverse(x)) を使用してください。

Query_System

Query_System 関数は、エラーが発生したときに、35 バイトのシステム構成ストリングまたはヌル (’’) を戻します。

前提呼び出し

この関数には前提呼び出しはありません。

結果

Query_System 関数の構文は次のとおりです。

HLLAPI(’Query_system’)

指定パラメーター

この関数用に提供されているパラメーターはありません。

戻りパラメーター

戻りコード	説明	
’ ’	ヌル。エラーが発生しました。	
データ	**位置**	**定義**
	1	EHLLAPI バージョン番号
	2 ～ 3	EHLLAPI レベル番号
	4 ～ 9	予約済み
	10 ～ 12	予約済み。
	13	ハードウェア・ベース。U = 判別不可
	14	プログラム・タイプ (P は IBM パーソナル・コミュニケーションズ)。
	15 ～ 16	予約済み。
	17 ～ 18	PCOMM バージョン/レベル (2 バイトの ASCII 値)
	19	予約済み。
	20 ～ 23	予約済み
	24 ～ 27	予約済み
	28 ～ 29	予約済み
	30 ～ 31	2 バイトの 2 進数で表される国別コード
	32	次に使用するモニター・タイプを 1 バイトの印刷可能な ASCII コードで表します。 • V = VGA • H = XGA • U = 不明
	33 ～ 35	予約済み。

Query_Window_Coord

Query_Window_Coord 関数は、ホスト・セッションのウィンドウ、または *session_id* がブランクの場合は現行接続セッションに対して、プレゼンテーション・マネージャーのプレゼンテーション・ウィンドウ座標を要求します。

前提呼び出し

Connect_PM

結果

Query_Window_Coord 関数の構文は次のとおりです。

HLLAPI('Query_window_coord', *session_id* **)**

指定パラメーター

関数名: **Query_Window_Coord**
session_id: セッション・ウィンドウの 1 文字短縮名。

戻りパラメーター

戻りコード	説明
''	ヌル。接続されていません。
データ	このデータ・ストリングは下記の形式で 4 つの 10 進数を戻します。 `xLeft yBottom xRight yTop`

Query_Workstation_Profile

Query_Workstation_Profile 関数は、指定されたホスト・セッションを開始する際に使用されたプロファイル名を戻します。

前提呼び出し

この関数には前提呼び出しはありません。

結果

Query_Workstation_Profile 関数の構文は次のとおりです。

HLLAPI('Query_workstation_profile', *session_id* **)**

指定パラメーター

関数名: **Query_Workstation_Profile**
session_id: 照会されるセッションのセッション文字 (A-Z) を指定します。

戻りパラメーター

戻りコード	説明
' '	ヌル。エラーが発生しました。
データ	セッションを開始するために使用されたワークステーション・プロファイルの名前。

Receive_File

Receive_File 関数は、ホスト・セッションからワークステーション・セッションにファイルを転送します。

注: ファイル転送が進行している間は、受信プログラムを終了しないでください。終了すると、エラー・メッセージが出されます。

前提呼び出し

この関数には前提呼び出しはありません。

結果

Receive_File 関数の構文は次のとおりです。

HLLAPI('Receive_file', *string* **)**

指定パラメーター

関数名: **Receive_File**

string: RECEIVE コマンドで指定されたのと同じパラメーター。パラメーターの詳細については、136 ページの『Receive File (91)』を参照してください。

戻りパラメーター

戻りコード	説明
2	パラメーター・エラーが発生しました。つまり、EHLLAPI バッファーに対して短過ぎるかまたは長すぎる (0 バイトか 128 バイトより大きい) データ・ストリング長が指定されました。ファイル転送が失敗しました。
3	ファイル転送が完了しました。
4	ファイル転送が完了し、そしてレコードを分割しました。
9	システム・エラーが発生しました。
27	ファイル転送の取り消しにより終了したか、または **Set_Session_Parms** 関数でタイムアウト時間が指定されているのであれば、そのタイムアウトが終了したかのいずれかです。
101	ファイル転送 (CICS への/CICS からの転送) は正常に終了しました。
300+x	EHLLAPI により報告される Win32 エラー・コードが 300 より大きい。Win32 のエラー・コードを判別するには、戻りコードから 300 を減算し、「*IBM OS/2 コントロール・プログラム プログラミング解説書*

他の戻りコードも受信される場合があります。これらの戻りコードは、ホスト転送プログラムにより生成されるメッセージ番号に関連しています。CICS ホスト転送プログラムに転送する場合、その戻りコードから 100 を減算するとメッセージの数値部分を得ることができます。例えば、戻りコードが 101 の場合には、ホストによりメッセージ番号 INW0001 が出されたことを示します。他のホスト転送プログラムの場合には、そのメッセージの数字部分のみ使用してください。例えば、戻りコードが 34 の場合には、ホストによりメッセージ番号 TRANS34 が出されたことを示します。特定のメッセージの意味については、ご使用のホスト転送プログラムの資

料を参照してください。

Release

Release 関数は、接続されたディスプレイ・セッションのキーボードをブロック解除します。

前提呼び出し

Connect

結果

Release 関数の構文は次のとおりです。

HLLAPI('Release')

指定パラメーター

この関数用に提供されているパラメーターはありません。

戻りパラメーター

戻りコード	説明
0	**Release** 関数は正常終了しました。
1	プログラムは現在、ホスト・セッションと接続されていません。
9	システム・エラーが発生しました。

追加情報

キーボードがロック状態 (**Reserve** 関数) のときにキーボードを切断すると、そのキーボードは自動的に解放されます。

Reserve

Reserve 関数は、**Release** または **Disconnect** 関数のいずれかが実行されるまで、現行接続セッションをユーザー入力からブロックします。

前提呼び出し

Connect

結果

Reserve 関数の構文は次のとおりです。

HLLAPI('Reserve')

指定パラメーター

この関数用に提供されているパラメーターはありません。

戻りパラメーター

戻りコード	説明
0	**Reserve** 関数は正常終了しました。
1	プログラムは現在、ホスト・セッションと接続されていません。
5	表示スペースが使用禁止です。
9	システム・エラーが発生しました。

Reset_System

Reset_System 関数は、(**Set_Session_Parms** 関数により設定される) セッション・パラメーターをデフォルト値にリセットし、すべての接続されたリソースを切断します。

前提呼び出し

この関数には前提呼び出しはありません。

結果

Reset_System 関数の構文は次のとおりです。

HLLAPI('Reset_system')

指定パラメーター

この関数用に提供されているパラメーターはありません。

戻りパラメーター

戻りコード	説明
0	**Reset_System** 関数が正常に終了しました。
9	システム・エラーが発生しました。

Search_Field

Search_Field 関数は、現在接続されている表示スペースで、指定されたストリングが特定のターゲット・フィールド以降出てくるかどうかを調べます。SRCHALL (デフォルト値) オプションが **Set_Session_Parms** 関数に指定されていると、*pos* パラメーターはオーバーライドされます。

前提呼び出し

Connect

結果

Search_Field 関数の構文は次のとおりです。

HLLAPI('Search_field', *string*, *pos* **)**

指定パラメーター

関数名:	**Search_Field**
string:	検索するストリング。
pos:	表示スペースにおけるフィールドの位置。

戻りパラメーター

戻りコード	説明
0	ストリング が検索されないか、またはセッションが接続されていませんでした。
データ	接続された表示スペース内のストリング の位置。

DBCS のみ: この検索の開始位置が 2 バイト文字の 2 番目に指定されている場合、SRCHRFWD 検索は次の文字から、SRCHBKWD 検索は前の文字から始まります。指定されたストリングの最後の文字が 2 バイト文字の先頭の場合、それは含まれません。

検索の間、表示スペース中の SO/SI は無視されます。 2 バイトの制御文字を検索するには、検索ストリングを SO (X'0E') と SI (X'0I') の間に配置してください。例えば、データ・ストリング内の X'0E000C0F' は、 2 バイト文字 FF (X'000C') として扱われます。

Search_PS

Search_PS 関数は、ホストの表示スペースで特定のストリングを検索します。

前提呼び出し

Connect

結果

Search_PS 関数の構文は次のとおりです。

HLLAPI('Search_PS', *string*, *pos* **)**

指定パラメーター

関数名: **Search_PS**
string: 検索するストリング。
pos: 検索を開始する PS 位置。

戻りパラメーター

戻りコード	説明
0	ストリング が検索されないか、またはセッションが接続されていませんでした。
データ	接続された表示スペース内のストリング の位置。

DBCS のみ: この検索の開始位置が 2 バイト文字の 2 番目に指定されている場合、SRCHRFWD 検索は次の文字から、SRCHBKWD 検索は前の文字から始まります。指定されたストリングの最後の文字が 2 バイト文字の先頭の場合、それは含まれません。

検索の間、表示スペース中の SO/SI は無視されます。 2 バイトの制御文字を検索するには、検索ストリングを SO (X'0E') と SI (X'0I') の間に配置してください。例えば、データ・ストリング内の X'0E000C0F' は、 2 バイト文字 FF (X'000C') として扱われます。

Send_File

Send_File 関数は、EHLLAPI が実行されているパーソナル・コンピューターのセッションからホスト・セッションへファイルを転送します。

注: ファイル転送の進行中は、送信プログラムを終了しないでください。終了すると、エラー・メッセージが出されます。

前提呼び出し

この関数には前提呼び出しはありません。

結果

Send_File 関数の構文は次のとおりです。

HLLAPI('Send_file', *string* **)**

指定パラメーター

関数名: **Send_File**

string SEND コマンドで指定したのと同じパラメーター。パラメーターの詳細については、150 ページの『Send File (90)』を参照してください。

戻りパラメーター

戻りコード	説明
2	パラメーター・エラーが発生しました。つまり、EHLLAPI バッファーに対して長過ぎるかまたは短過ぎるデータ・ストリング長が指定されました。ファイル転送が失敗しました。
3	ファイル転送が完了しました。
4	ファイル転送が完了し、そしてレコードを分割しました。
5	ワークステーションのファイル名が無効かまたは見つかりませんでした。ファイル転送が取り消されました。
9	システム・エラーが発生しました。
27	ファイル転送の取り消しにより終了したか、または **Set_Session_Parms** 関数でタイムアウト時間が指定されているのであれば、そのタイムアウトが終了したかのいずれかです。
101	ファイル転送 (CICS への/CICS からの転送) は正常に終了しました。
300+x	EHLLAPI により報告される Win32 エラー・コードが 300 より大きい。Win32 のエラー・コードを判別するには、戻りコードから 300 を減算し、「*IBM OS/2 コントロール・プログラム プログラミング解説書*」を参照してください。

他の戻りコードも受信される場合があります。これらの戻りコードは、ホスト転送プログラムにより生成されるメッセージ番号に関連しています。CICS ホスト転送プログラムに転送する場合、その戻りコードから 100 を減算するとメッセージの数値部分を得ることができます。例えば、戻りコードが 101 の場合には、ホストによりメッセージ番号 INW0001 が出されたことを示します。他のホスト転送プログラムの場合には、そのメッセージの数字部分のみ使用してください。例えば、戻りコードが 34 の場合には、ホストによりメッセージ番号 TRANS34 が出されたことを示

します。特定のメッセージの意味については、ご使用のホスト転送プログラムの資料を参照してください。

Sendkey

Sendkey 関数は、現在接続しているホストの表示スペースに単一または連続したキー・ストロークを転送します。 *string* パラメーターはキー・ストロークのセットを定義し、それらはホストの表示スペースに送信されます。最大 255 のキーが一度に送信可能です。

前提呼び出し

Connect

結果

Sendkey 関数の構文は次のとおりです。

HLLAPI('Sendkey', *string* **)**

指定パラメーター

関数名: **Sendkey**
string: キー・ストローク列。 155 ページの『キーボードの略号』を参照。

戻りパラメーター

戻りコード	説明
0	キー・ストロークが送信され、状況は正常でした。
1	プログラムは現在、ホスト・セッションと接続されていません。
4	ホスト・セッションは使用中であったため、どのキー・ストロークも送信できませんでした。
5	ターゲット・セッションに対する入力が使用禁止にされています。キー・ストロークは拒否されたか、無効なキー・ストロークの略号が送信されました。どのキー・ストロークも送信できませんでした。
6	不完全なキー・ストローク略号。
9	システム・エラーが発生しました。

Set_Cursor_Pos

Set_Cursor_Pos 関数は、現在接続しているホストの表示スペース内の指定されたターゲット・フィールドにカーソルを置きます。

前提呼び出し

Connect

結果

Set_Cursor_Pos 関数の構文は次のとおりです。

HLLAPI('Set_cursor_pos', *pos* **)**

指定パラメーター

関数名: **Set_Cursor_Pos**
pos: コピーするターゲット・フィールド。

戻りパラメーター

戻りコード	説明
0	カーソルは指定された位置に正常に位置付けられました。
1	プログラムは現在、ホスト・セッションと接続されていません。

Set_Session_Parms

Set_Session_Parms 関数は、現行セッションのパラメーターを設定します。

前提呼び出し

この関数には前提呼び出しはありません。

結果

Set_Session_Parms 関数の構文は次のとおりです。

HLLAPI('Set_session_parms', *string* **)**

指定パラメーター

関数名: **Set_Session_Parms**
string: 変更するセッション・オプションを含むストリング。

戻りパラメーター

戻りコード	説明
0	セッションのパラメーターが設定されました。
2	1 つ以上のパラメーターが無効でした。
9	システム・エラーが発生しました。

追加情報

STREOT および EOT オプションは、**Set_Session_Parms** 関数ではサポートされていません。

Set_Window_Status

Set_Window_Status 関数は、セッションのウィンドウ状況を変更します。

前提呼び出し

Connect_PM

結果

Set_Window_Status 関数の構文は次のとおりです。

HLLAPI('Set_window_status', *session_id*, *option* [, *num1* | *option1*, *num2*])

指定パラメーター

関数名: **Set_Window_Status**
session_id: セッションの 1 文字短縮名。
option: 次の表を参照してください。

オプション	説明
'V'	ウィンドウを可視表示します。
'I'	ウィンドウを不可視状態にします。
'A'	ウィンドウを活動状態にします。
'D'	ウィンドウを非活動状態にします。
'R'	ウィンドウを最大化状態または最小化状態から復元します。
'Z'	ウィンドウ配置を *option1* の先頭文字に基づいて変更します。 **Top** エミュレーション・ウィンドウを前景に移動する。 **Bottom** エミュレーション・ウィンドウを背景に移動する。
'X'	ウィンドウを最大化にします。
'N'	ウィンドウを最小化にします (アイコンに変えられる)。
'M'	ここで、 *num1* および *num2* は、新規ウィンドウ位置の左下隅の位置の 10 進表示です。
'S'	ここで、*num1* および *num2* は、新規ウィンドウの幅と高さの 10 進数表示です。

num1 および *num2* パラメーターは、 *Move* ('M') および *Size* ('S') オプションにのみ使用され、*option1* パラメーターは *Zorder* ('Z') オプションに使用されます。

戻りパラメーター

戻りコード	説明
0	**Set_Window_Status** 関数が正常に終了しました。
1	プログラムは現在、ホスト・セッションと接続されていません。
9	システム・エラーが発生しました。
12	セッションが停止しました。

Start_Close_Intercept

Start_Close_Intercept 関数は、ホスト・セッションのクローズ要求を代行受信します。

前提呼び出し

この関数には前提呼び出しはありません。

結果

Start_Close_Intercept 関数の構文は次のとおりです。

HLLAPI('Start_close_intercept', *session_id* **)**

指定パラメーター

関数名: **Start_Close_Intercept**
session_id: セッションの 1 文字短縮名。

戻りパラメーター

戻りコード	説明
0	**Start_Close_Intercept** 関数が正常に終了しました。
1	ホストの無効な表示スペースが指定されました。
2	パラメーター・エラーが生じました。
9	システム・エラーが発生しました。
10	エミュレーション・プログラムはこの関数をサポートしていません。

Start_Communication

Start_Communication 関数は、指定された *session_id* のホスト・セッションと通信を開始します。この呼び出しは、エミュレーター・ウィンドウの「通信」->「接続」を実行するのと同等です。

前提呼び出し

この関数には前提呼び出しはありません。

結果

Start_Communication 関数の構文は次のとおりです。

HLLAPI('Start_communication', *session_id* **)**

指定パラメーター

関数名: **Start_Communication**
session_id: セッションの 1 文字短縮名。

戻りパラメーター

戻りコード	説明
0	**Start_Communication** 接続要求が正常終了しました。
1	無効なセッション ID が指定されました。
2	指定されたセッションが開始されていませんでした。

Start_Host_Notify

Start_Host_Notify 関数は、任意のホスト表示スペースまたはオペレーター情報域が更新されたかどうかを判別します。

前提呼び出し

この関数には前提呼び出しはありません。

結果

Start_Host_Notify 関数の構文は 次のとおりです。

HLLAPI('Start_host_notify', *session_id*, *option* **)**

指定パラメーター

関数名:	**Start_Host_Notify**
session_id:	セッションの 1 文字短縮名。
E	プリンター・セッション中の完了通知を要求します。
option:	次の表を参照してください。

値	説明
'P'	表示スペースの更新だけを調べます。
'O'	OIA の更新だけを調べます。
'B'	表示スペースと OIA 両方の更新を調べます。

戻りパラメーター

戻りコード	説明
0	**Start_Host_Notify** 関数が正常に終了しました。
1	ホストの無効な表示スペースが指定されました。
2	パラメーター指定でエラーが発生しました。
9	システム・エラーが発生しました。

Start_Keystroke_Intercept

Start_Keystroke_Intercept 関数は、*session_id* パラメーターで指定したセッションに送られるすべてのキー・ストロークをフィルターに掛けます。

前提呼び出し

この関数には前提呼び出しはありません。

結果

Start_Keystroke_Intercept 関数の構文は次のとおりです。

HLLAPI('Start_keystroke_intercept', *session_id*, *option* **)**

指定パラメーター

関数名: **Start_Keystroke_Intercept**
session_id: セッションの 1 文字短縮名。
option: 次の表を参照してください。

オプション	説明
'D'	AID キーだけ。
'L'	すべてのキー・ストローク。

戻りパラメーター

戻りコード	説明
0	**Start_Keystroke_Intercept** 関数が正常に終了しました。
1	表示スペースが無効でした。
2	無効なオプションが指定されました。
4	リソースが使用不可能でした。要求した表示スペースは他の API アプリケーションにより使用中でした。
9	システム・エラーが発生しました。

Start_Session

Start_Session 関数は、指定されたワークステーション・プロファイルと、任意指定の *session_id* および開始オプションを使用して、ホスト・セッションを開始します。

前提呼び出し

この関数には前提呼び出しはありません。

結果

Start_Session 関数の構文は次のとおりです。

```
HLLAPI( 'Start_session', profile_name, option[,
 session_id] )
```

指定パラメーター

関数名: **Start_Session**

session_id: 任意指定パラメーター。開始するセッションに関連付けられているセッション文字 (A-Z) を指定します。ヌルの場合には、使用可能な次のセッション文字が使用されます。

profile_name: 開始するワークステーション・プロファイルのファイル名。パスを組み込むこともできますが、オプションです。

option: 次の表を参照してください。

オプション	説明
'V'	ウィンドウを可視状態にしてセッションを開始します。
'I'	ウィンドウを不可視状態にしてセッションを開始します。
'X'	ウィンドウを最大化してセッションを開始します。
'N'	ウィンドウを最小化してセッションを開始します。

戻りパラメーター

戻りコード	説明
0	**Start_Session** 関数が正常終了しました。
1	無効なセッション ID が指定されました。
2	指定されたセッション ID はすでに使用中です。
3	ワークステーション・プロファイル名が無効です。
4	無効な操作が指定されました。
9	システム・エラーが発生しました。

Stop_Close_Intercept

Stop_Close_Intercept 関数は、アプリケーションから **Start_Close_Intercept** 関数をオフに設定できるようにします。**Stop_Close_Intercept** 関数を実行した後は、*session_id* で指定したセッションに対するクローズ要求が受け入れられます。

前提呼び出し

Start_Close_Intercept

結果

Stop_Close_Intercept 関数は次のとおりです。

HLLAPI('Stop_close_intercept', *session_id* **)**

指定パラメーター

関数名: **Stop_Close_Intercept**
session_id: セッションの 1 文字短縮名。

戻りパラメーター

戻りコード	説明
0	**Stop_Close_Intercept** 関数が正常に終了しました。
1	ホストの無効な表示スペースが指定されました。
8	前の **Start_Close_Intercept** 関数が発行されていません。
9	システム・エラーが発生しました。
12	セッションは停止していました。

Stop_Communication

Stop_Communication 関数は、指定された *session_id* のホスト・セッションとの通信を停止します。この呼び出しは、エミュレーター・ウィンドウで「通信」->「切断」を実行するのと同等です。

前提呼び出し

この関数には前提呼び出しはありません。

結果

Stop_Communication 関数の構文は次のとおりです。

HLLAPI('Stop_communication', *session_id* **)**

指定パラメーター

関数名: **Stop_Communication**
session_id: セッションの 1 文字短縮名。

戻りパラメーター

戻りコード	説明
0	**Stop_Communication** 切断の要求は正常終了しました。
1	無効なセッション ID が指定されました。
2	指定されたセッションが開始されていませんでした。

Stop_Host_Notify

Stop_Host_Notify 関数は、 **Start_Host_Notification** 関数が、ホスト・セッションの ID がすでに更新されたかどうかを判別するのを中止します。

前提呼び出し

Start_Host_Notify

結果

Stop_Host_Notify 関数の構文は次のとおりです。

HLLAPI('Stop_host_notify', *session_id* **)**

指定パラメーター

関数名: **Stop_Host_Notify**
session_id: セッションの 1 文字短縮名。

戻りパラメーター

戻りコード	説明
0	**Stop_Host_Notify** 関数が正常に終了しました。
1	ホストの無効な表示スペースが指定されました。
8	**Start_Host_Notify** 関数は発行されませんでした。
9	システム・エラーが発生しました。

Stop_Keystroke_Intercept

Stop_Keystroke_Intercept 関数は、 *session_id* に対するキー・ストロークをアプリケーション・プログラムが代行受信する機能を終了します。
Start_Keystroke_Intercept 関数は、以前の **Start_Keystroke_Intercept** 関数を取り消します。

前提呼び出し

Start_Keystroke_Intercept

結果

Stop_Keystroke_Intercept 関数の構文は次のとおりです。

HLLAPI('Stop_keystroke_intercept', *session_id* **)**

指定パラメーター

関数名: **Stop_Keystroke_Intercept**
session_id: セッションの 1 文字短縮名。

戻りパラメーター

戻りコード	説明
0	**Stop_Keystroke_Intercept** 関数が正常に終了しました。
1	ホスト表示スペースが無効でした。
8	**Start_Keystroke_Intercept** 関数はこの表示スペースに対して呼び出されていません。
9	システム・エラーが発生しました。

Stop_Session

Stop_Session 関数は、指定されたホスト・セッションを停止します。

前提呼び出し

この関数には前提呼び出しはありません。

結果

Stop_Session 関数の構文は次のとおりです。

HLLAPI('Stop_session', *session_id*, *save_option*)

指定パラメーター

関数名: **Stop_Session**
session_id: 停止するセッションのセッション文字 (A-Z) を指定します。
save_option: 次の表を参照してください。

オプション	説明
'D'	プロファイルで指定されたように、デフォルトのプロファイル保管オプションを使用。
'S'	終了時にプロファイルを保管。
'N'	終了時にプロファイルを保管しない。

戻りパラメーター

戻りコード	説明
0	**Stop_Session** 関数が正常終了しました。
1	無効なセッション ID が指定されました。
2	指定されたセッション ID は開始されていません。

Wait

Wait 関数は、現行接続セッションの状況を調べます。コントローラーまたはホスト・システムが使用中の場合、この関数は状態が整うまで一定時間 EHLLAPI を待機させます。その待機時間は、**Set_Session_Parms** 関数の TWAIT、NWAIT、または LWAIT オプションで設定します。

前提呼び出し

Connect

結果

Wait 関数の構文は次のとおりです。

HLLAPI('Wait')

指定パラメーター

この関数用に提供されているパラメーターはありません。

戻りパラメーター

戻りコード	説明
0	キーボードはアンロックされ、入力の用意が整いました。
1	アプリケーション・プログラムは有効なセッションに接続されませんでした。
4	使用中のタイムアウト (XCLOCK または XSYSTEM 状態で)。
5	キーボードがロックされています。
9	システム・エラーが発生しました。

プログラミング上の注意

EHLLAPI 機能の中には、REXX EHLLAPI 環境に適用されないものもあります。EHLLAPI 機能の構造化フィールドは、REXX EHLLAPI ではサポートされていません。

サンプル・プログラム

REXX EHLLAPI 機能の使用法を示すためのサンプル・プログラムが、CD-ROM に収められています。

最初のサンプル・プログラム (QTIME.CMD) は、VM システム時刻に基づいてシステム・クロックを設定するものです (VM ホスト・セッションに接続する場合にのみこのプログラムを実行してください)。

2 番目のサンプル・プログラム (CMMACRO.CMD) は、ホスト・システム上のキー・ストロークを記録し、それを再生するものです。これにより反復タスクは容易になります。

注: それぞれのサンプル・プログラムは、終了する前に、使用リソースを切断および開放します。

付録 G. 特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものであり、本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒106-0032
東京都港区六本木 3-2-31
IBM World Trade Asia Corporation
Licensing

以下の保証は、国または地域の法律に沿わない場合は、適用されません。 IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。 IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM Corporation
Department T01
Building 062
P.O. Box 12195

RTP, NC 27709-2195
U.S.A.

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、プログラムのご使用条件、またはそれと同等の条項に基づいて、IBM より提供されます。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。 IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

IBM の将来の方向または意向に関する記述については、予告なしに変更または撤回される場合があり、単に目標を示しているものです。

著作権使用許諾:

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。お客様は、IBM のアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。

商標

以下は、IBM Corporation の商標です。

CICS
eServer
i5/OS
IBM
IBM Global Network
iSeries
MVS
OS/2
OS/400
Presentation Manager
PS/2
System i5
VisualAge
zSeries

Microsoft、Windows、および Windows ロゴは、Microsoft Corporation の米国およびその他の国における商標です。

Java、JavaBeans、およびすべての Java 関連の商標およびロゴは、Sun Microsystems, Inc. の米国およびその他の国における商標です。

他の会社名、製品名およびサービス名等はそれぞれ各社の商標です。

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

[ア行]

[カ行]

[サ行]

[タ行]

[ナ行]

[ハ行]

[マ行]

[ヤ行]

[ラ行]

[数字]

A

B

C

D

E

F

G

I

L

N

O

P

Q

R

S

T

U

W

X

プログラム番号: 5639-I70

Printed in Japan

SC88-5629-09

日本アイ・ビー・エム株式会社
〒106-8711　東京都港区六本木3-2-12