ホストアクセスクラスライブラリ

本書について

本書は、HCL Z and I Emulator for Windows Version 3.0 版ホスト・アクセス・クラス・ライブラリー (HACL) を使用する際に必要なプログラミング情報を記載しています。本書では、*Windows*® は、Windows® 7、Windows® 8、Windows® 8.1、Windows® 10、Windows® Server 2008、および Windows® Server 2012 を指します。また、本書では、ワークステーションはサポートされているすべてのパーソナル・コンピューターを指します。パーソナル・コンピューターの 1 つのモデルまたはアーキテクチャーのことしか指していない場合には、そのタイプだけを指定します。

本書の対象読者

本書は、ホスト・アクセス・クラス・ライブラリー (HACL) 機能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および開発者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® に関する情報については、詳細情報の参照先 (ページ 2) にある資料のリストを参照してください。

本書では、読者が、使用する言語およびコンパイラーに 精通していると想定して説明しています。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、詳細情報の参照先 (ページ 2)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。

本書の使用方法

本書は、以下のように構成されています。

- 概要 (ページ 4)では、ホスト・アクセス・クラス・ライブラリーの概要を説明しています。
- ホスト・アクセス・クラス・ライブラリー C++ (ページ 14)では、ホスト・アクセス・クラス・ライブラリーの C++ メソッド および特性を説明しています。
- ホスト・アクセス・クラス・ライブラリーの自動化オブジェクト (ページ 254)では、ホスト・アクセス・クラス・ライブラリーの自動化オブジェクトの メソッドおよび特性を説明しています。
- Java 用ホスト・アクセス・クラス・ライブラリー (ページ 428) では、ホスト・アクセス・クラス・ライブラリー (HACL) Java™ クラスについての詳細情報がどこに記述されているかを説明しています。
- Sendkeys 略号キーワード (ページ 430)では、Sendkeys メソッドの略号キーワードを記載しています。
- ECL プレーン ー形式および内容 (ページ 433)では、HACL 表示スペース・モデル内の 種々のデータ・プレーンの形式および内容を説明しています。

詳細情報の参照先

Z and I Emulator for Windows・ライブラリーには、以下の資料が含まれています。

- *Installation Guide*
- *Quick Beginnings*
- *Emulator User's Reference*

- *Administrator's Guide and Reference*
- *Emulator Programming*
- ホスト・アクセス・クラス・ライブラリー

印刷した資料のほかに、Z and I Emulator for Windows で提供される HTML 資料もあります。

Java 用ホスト・アクセス・クラス・ライブラリー

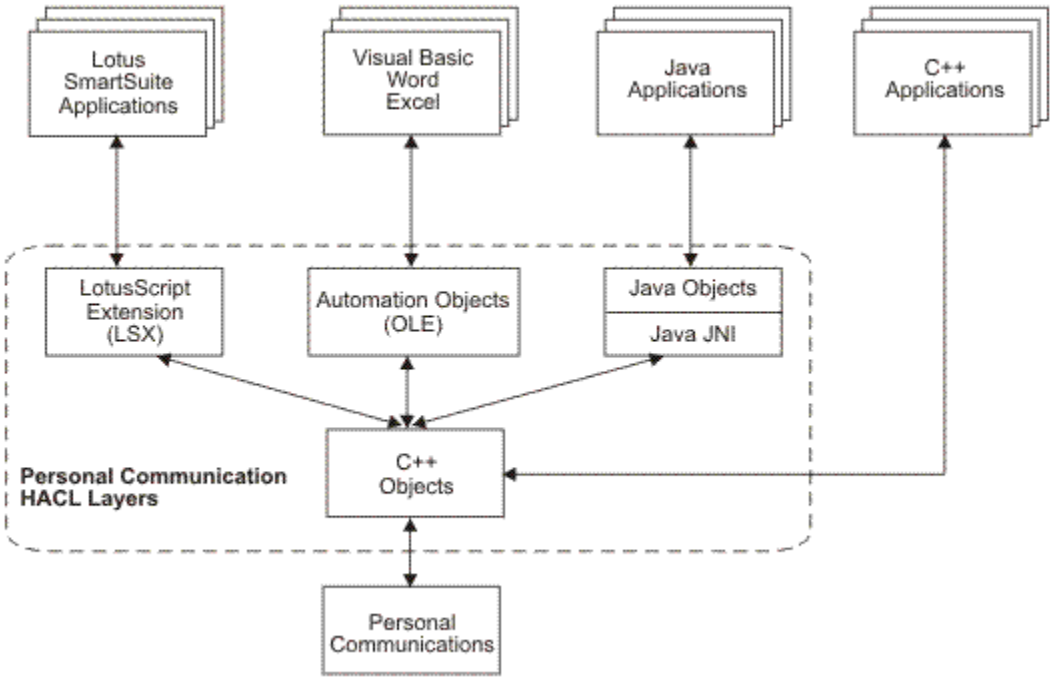
HACL Java HTML ファイルは、組み込みオブジェクトとしてZ and I Emulator for Windowsを使用するための ActiveX/OLE 2.0 準拠アプリケーションの作成方法を説明しています。これらのファイルは、以下のパスにあるZ and I Emulator for Windowsの製品資料に付属している Docs_Admin_Aids zipped フォルダーからアクセスできます: *ZIEWin_3.0_Docs_Admin_Aids.zip\publications\ja\doc\hacl*

第 1 章. 概要

ホスト・アクセス・クラス・ライブラリー (HACL) は、アプリケーション・プログラマーがホスト・アプリケーションに簡単かつ迅速にアクセスできるようにするオブジェクトのセットです。HCL Z and I Emulator for Windows では、複数の異なる HACL レイヤーをサポートすることにより、以下の多様なプログラミング言語と環境をサポートします。C++ オブジェクト、Java™ オブジェクト、Microsoft® COM ベースの自動化テクノロジー (OLE)。どの層も同様な機能性を提供しますが、個々の 環境別の構文および機能の相違に応じて、それぞれの層ごとに若干の相違があります。機能性および柔軟性の最も高い層は C++ 層であり、この層は他のすべての層の基盤を提供します。

このような階層化の概念によって、Java™、Microsoft® Visual Basic®、Visual Basic® for Applications、Lotus® Notes™、Lotus® WordPro、および Visual C++® を含め、多種多様なプログラミング環境で基本的な HACL 関数を使用することができます。次の図は、各 HACL 層を示しています。

図 1. HACL 層



C++ オブジェクト

この C++ クラス・ライブラリーは、ホスト接続をオブジェクト指向によって抽象化したものを完全な形で提供します。それには、ホスト表示スペース (画面) での読み書き、画面上のフィールドのエミュレーション、状況情報についてのオペレーター標識域 (OIA) の読み取り、ビジュアル・エミュレーター・ウィンドウに関する情報のアクセスと更新、ファイルの転送、および重要イベントの非同期通知の実行が含まれます。

C++ オブジェクトの詳細については、ホスト・アクセス・クラス・ライブラリー C++ (ページ 14)を参照してください。

Java オブジェクト

Java™ オブジェクトは、Host-on-Demand バージョン 3 に類似する HACL 関数すべてに対して Java™ ラッピングを提供します。HACL Java™ クラスの詳細については、Java 用ホスト・アクセス・クラス・ライブラリー (ページ 428) を参照してください。

自動化オブジェクト

ホスト・アクセス・クラス・ライブラリーの自動化オブジェクトを使用すると、Z and I Emulator for Windows は、Microsoft® COM ベースのオートメーション技術 (以前は、OLE オートメーションと呼ばれていました) をサポートできます。HACL 自動化オブジェクトは、一連の自動化サーバーであり、これを使用することによって、自動化コントローラー (Microsoft® Visual Basic® など) が、プログラマチックに Z and I Emulator for Windows のデータおよび機能にアクセスすることができます。言い替えると、自動化プロトコルを制御できるアプリケーション (自動化コントローラー) は、Z and I Emulator for Windows の操作の一部 (自動化サーバー) を制御することができます。

注: HCL Z and I Emulator for Windows で提供される自動化オブジェクトは本質的に 32 ビットです。これらは 32 ビット Microsoft Office プログラムでのみ使用できます。

自動化オブジェクト層の詳細については、ホスト・アクセス・クラス・ライブラリーの自動化オブジェクト (ページ 254)を参照してください。

ECL の概念

以下のセクションで、エミュレーター・クラス・ライブラリー (ECL) のいくつかの基本概念について説明します。これらの概念を理解すれば、ライブラリーを有効に使用するのに役立ちます。

接続、ハンドルと名前の

ECL に関する限り、接続とは Z and I Emulator for Windows の単一かつ固有のエミュレーター・ウィンドウのことです。エミュレーター・ウィンドウは、ホストに実際に接続されていることも接続されていないこともあり、また、画面に表示されることも表示されないこともあります。例えば、Z and I Emulator for Windows のウィンドウが切断状態にある場合もあります。各接続は、接続ハンドルまたは接続名によって区別されます。大半の HACL オブジェクトは、特定の接続に関連があります。通常、オブジェクトは、オブジェクトのコンストラクター上のパラメーターとして、接続ハンドルまたは接続名をとります。Visual Basic® などの、コンストラクター上のパラメーターをサポートしない言語では、関連づけを行うためにメンバー関数が提供されます。オブジェクトは、その作成後に他の接続に関連付けることはできません。例えば、接続 'B' に関連付けられた ECLPS (表示スペース) オブジェクトを作成するには、次のようなコードを使用します。

C++

```
ECLPS *PSObject;
PSObject = new ECLPS('B');
```

Visual Basic®

```
Dim PSObject as Object
Set PSObject = CreateObject("ZIEWin.autECLPS")
PSObject.SetConnectionByName("B")
```

HACL 接続名は、A から Z、または a から z の単一の文字から成ります。接続名の最大数は 52 であり、現在、Z and I Emulator for Windows での同時接続は 52 までに制限されています。接続名は、EHLLAPI 短縮セッション ID と、Z and I Emulator for Windowsのウィンドウ・タイトルおよび OIA に示されるセッション ID と同じです。

HACL ハンドルは、単一の接続を表す固有の 32 ビット番号です。接続名とは異なり接続ハンドルは 52 までの値に限定されておらず、値そのものには、アプリケーションに対する意味はありません。1 つの接続ハンドルを使用して、複数のスレッドおよびプロセスにおいて 同一の接続を参照することができます。

今後の拡張に備えて、アプリケーションでは可能な限り接続ハンドルを使用するようにしてください。たいていの HACL オブジェクトは、接続の識別の必要があるときに ハンドルまたは名前を受け入れます。基本 HACL クラスには、ハンドルから名前へ、また名前からハンドルへ変換するのに使用できる関数があります。これらの関数は、どの HACL オブジェクトからでも使用できます。

注: 接続のプロパティーは、動的なものです。例えば、別のホストに接続を構成し直すと、GetConnType から 戻される接続タイプが変わることがあります。一般に、アプリケーションでは、接続のプロパティーは固定のものと見なしてはなりません。

セッション

ECL に関して言えば、セッション・オブジェクト (ECLSession) とは、他のすべての接続固有オブジェクト用の単なるコンテナーです。これは、特定の接続用の完全なセットの HACL オブジェクトを、アプリケーション で作成するためのショートカットを提供します。*session* という用語は、Z and I Emulator for Windows セッションの概念と混同してはなりません。Z and I Emulator for Windows のセッションとは、画面上の物理エミュレーション・ウィンドウを指します。

ECLSession オブジェクトを作成または破棄しても、Z and I Emulator for Windows のセッション (ウィンドウ) は影響を受けません。アプリケーションでは、1 つまたは複数の 接続を参照する ECLSession オブジェクトをいくつでも作成することができます。

ECL コンテナー・オブジェクト

HACL クラスによっては、他のオブジェクトのコンテナーとして働くものもあります。例えば、ECLSession オブジェクトは、ECLPS、ECLOIA、ECLWinMetrics、および ECLXfer オブジェクトのインスタンスを収容します。コンテナーは、含まれる オブジェクトを指すポインターを戻すメソッドを提供します。例えば、ECLSession オブジェクトには GetOIA メソッドがありますが、 これは OIA オブジェクトを指すポインターを戻します。収容されたオブジェクトは、そのコンテナーのクラスの共用メンバーとして実装されるのではなく、メソッドを通してのみアクセスされます。

パフォーマンス上の理由または他の理由から、コンテナー・オブジェクトの作成時に 収容オブジェクトは作成されることも作成されないこともあります。クラスの実装において、収容オブジェクトを指すポインターをアプリケーションが最初に 要求するまで、そのオブジェクトの作成を延期することもできます。アプリケーションでは、収容されるオブジェクトはそのコンテナーと 同時に作成されると見なしてはなりません。例えば、ECLSession オブジェ

クトの作成時に ECLPS オブジェクト のインスタンスは作成されないことがあります。そのような場合、GetPS メソッドが最初に呼び出される まで、ECLSession クラスは ECLPS オブジェクトの作成を延期することができます。

コンテナー・クラスが破棄されると、そこに収容されているすべてのインスタンスも破棄されます。アプリケーションに戻されたすべてのポインターは無効になるので、使用してはなりません。

注: HACL 層によっては (自動化オブジェクトなど)、包含方式を隠したり、これを明示ポインターを使用しない命名方式に 再キャストしたりすることがあります。

ECL リスト・オブジェクト

いくつかの HACL クラスにはリスト反復機能があります。例えば、ECLConnList クラスは接続のリストを管理します。ECL リスト・クラスは、リスト内容の変更を反映するよう非同期で更新されることはありません。アプリケーションは、リストの内容を更新するために明示的に Refresh メソッドを呼び出さなければなりません。それによって、反復中にリストが変更されたかどうかを心配せずに、アプリケーションでリストを反復することができます。

イベント

HACL は、特定のイベントを非同期通知する機能を備えています。アプリケーションは、特定のイベントが起きたときに通知を受けるように選択することができます。例えば、アプリケーションは、Z and I Emulator for Windows の新しい接続が開始したら通知を受けるようにすることができます。現在、HACL は次のようなイベントの通知をサポートします。

- 接続のスタート・ストップ
- 通信の接続/切断
- オペレーターのキー・ストローク
- 表示スペースまたは OIA の更新

イベントの通知は、ECLNotify 抽象基本クラスによって実装されます。イベント・タイプごとに別々のクラスが存在します。アプリケーションは、イベントの通知を受けられるようにする には、ECLNotify 抽象基本クラスのいずれかから派生したオブジェクト を定義および作成しなければなりません。次に、適切な HACL 登録機能を呼び出して、 そのオブジェクトを登録しなければなりません。アプリケーション・オブジェクトを登録すると、その後該当イベントが発生する度に、NotifyEvent メソッドが呼び出されます。

注:

1. アプリケーションの NotifyEvent メソッドは、別の実行スレッド上に非同期で呼び出されます。そのため、NotifyEvent メソッドは、再入可能でなければならず、また、アプリケーション・リソースにアクセスする場合は、適切なロックまたは同期化を使用する必要があります。
2. HACL 層 (自動化オブジェクトなど) によっては、HACL イベントを完全にはサポートしていなかったり、実装していなかったりすることがあります。

エラー処理

C++ 層では、HACL は C++ の構造化例外処理を使用します。一般に、ECLErr オブジェクトをもった C++ 例外を送り出す ことによって、アプリケーションにエラーが示されます。アプリケーションがエラーを catch するには、次のように HACL オブジェクトの呼び出しを try/catch ブロック内に入れる必要があります。

```
try {
   PSObj = new ECLPS('A');
   x = PSObj->GetSize();

   //...more references to HACL objects...

} catch (ECLErr ErrObj) {
   ErrNumber = ErrObj.GetMsgNumber();
   MessageBox(NULL, ErrObj.GetMsgText(), "ECL Error");
}
```

HACL エラーが catch されたとき、アプリケーションから ECLErr オブジェクトの メソッドを呼び出せば、エラーの正確な原因を判別することができます。また、ECLErr オブジェクトを呼び出して、完全な言語依存のエラー・メッセージを作成することもできます。

自動化オブジェクト層では、実行時エラーは、該当するスクリプト・エラーが作成される原因になります。アプリケーションは、On Error ハンドラーを使用して、エラーを捕そくしてエラーについての 追加情報を照会し、適切なアクションをとることができます。

アドレッシング (行、桁、位置)

HACL には、ホスト表示スペースにおいて点 (文字位置) をアドレッシングする方法が 2 つあります。アプリケーションは、行/列の 番号を使用するか、または単一の線形位置の値を使用して文字をアドレッシングすることができます。表示スペースのアドレッシングの場合、使用するアドレッシング方式に関係なく、常に 1 をベースにします (0 ベースではありません)。

行/列のアドレッシング方式が有用なのは、ホスト・データの物理画面表示に直接関係したアプリケーションの場合です。長方形の座標システム (左上隅が行 1 桁 1) を使うのが、 画面で点をアドレッシングする本来の方法です。線形位置アドレッシング法 (左上隅を位置 1 とし、左から右に向かって、上から下へ進む方法) が便利なのは、 表示スペース全体を単一のデータ・エレメント配列として表示するアプリケーションの場合、 または、このアドレッシング法を使用する EHLLAPI インターフェースから移植されたアプリケーションの場合です。

C++ 層では、同じメソッドでも、呼び出すシグニチャーが異なると、選択するアドレッシング方式も異なります。例えば、ホスト・カーソルを特定の画面座標に移動したい場合、アプリケーションは次の 2 つのいずれかのシグニチャーで ECLPS::SetCursorPos メソッドを呼び出すことができます。

```
PSObj->SetCusorPos(81);
PSObj->SetCursorPos(2, 1);
```

ホスト画面が行あたり 80 桁で構成されている場合、 上記のステートメントはどちらも同じ結果を生じさせます。またこの例は、アドレッシング方式における若干の相違も示します。つまり、表示スペースの行あたりの文字数についてアプリケーションに前提条件がある場合、線形位置メソッドでは予期しない結果を生じることがあります。例えば、この例のコードの 1 行目は、 132 桁に構成された表示スペースの行 1 の桁 81 にカーソルを置きます。コードの 2 行目は、表示スペースの構成に関係なく行 2 桁 1 にカーソルを置きます。

注: HACL 層によっては、1 つのアドレッシング方式しか公開していません。

移行、EHLLAPI からの

現在、エミュレーター高水準言語 API (EHLLAPI) 用に 書かれているアプリケーションは、ホスト・アクセス・クラス・ライブラリー を使用できるよう修正することができます。一般に、EHLLAPI から HACL にマイグレーションするには、 大幅なソース・コードの変更やアプリケーションの再構築が必要です。HACL は、EHLLAPI とは異なるプログラミング・モデルを提供するので、一般に、有効に働くには異なるアプリケーション構造を必要とします。

以下のセクションは、EHLLAPI に慣れたプログラマーが、HACL と EHLLAPI との 類似点および相違点を理解するのに役立ちます。以下の情報を使用すれば、HACL を 使用できるよう個々のアプリケーションを修正する方法を理解することができます。

注: EHLLAPI では、*session* という用語は、HACL の *connection* と同じ意味で使われます。このセクションでは、これらの用語を交換可能なものとして使っています。

実行/言語インターフェース

最も基本的なレベルでは、EHLLAPI と HACL には、アプリケーション・プログラムからの API の呼び出し方のメカニズムにおける違いがあります。

EHLLAPI は、複数用途のパラメーターを使う単一の呼び出し点インターフェースとしてインプリメントされています。DLL 内の 1 つのエントリー・ポイント (hllapi) が、 4 つのパラメーターの固定されたセットを基にしてすべての関数を提供します。パラメーターのうちの 3 つは、4 つ目のパラメーターの値に応じて 異なる意味をとります。この単純なインターフェースによって、さまざまなプログラミング環境や言語から 容易に API を呼び出すことが可能になります。その欠点は、1 つの関数と 4 つのパラメーターの中に多くの複雑性が集約されているということです。

HACL は、 明示的なエントリー・ポイントまたは関数の代わりに一連のプログラミング・オブジェクトを提供するオブジェクト指向インターフェースです。オブジェクトには、ホスト接続を操作するのに使えるプロパティーおよびメソッドがあります。構造の内容やパラメーター・コマンド・コードの詳細について配慮する必要はなく、アプリケーション機能に注意を集中できます。HACL オブジェクトは、サポートされている HACL 層環境 (C++ または自動化オブジェクト) のいずれからでも使用できます。この 3 つの層は、Microsoft® Visual C++®、Visual Basic®、および Lotus® SmartSuite® アプリケーションといった最新鋭のプログラミング環境にアクセスすることができます。

特徴

EHLLAPI レベルでは使えない多数の機能を、HACL はハイレベルで提供します。また、現在どの HACL クラスでもインプリメントされていない EHLLAPI 機能もいくつかあります。

HACL に固有の機能には次のものがあります。

- 接続 (セッション) スタート・ストップ機能
- ホスト通信リンクの接続/切断でのイベント通知
- 接続 (セッション) スタート・ストップでのイベント通知
- 包括的なエラー・トラッピング
- 言語特定のエラー・メッセージ・テキストの生成
- 接続 (セッション) 数を制限しない体系。 現在、Z and I Emulator for Windows は 52 に制限されています。
- 複数の並行接続 (セッション) およびマルチスレッド・アプリケーションのサポート
- ホスト表示スペース用の行/列アドレッシング
- 表示スペースの単純化されたモデル
- フィールドおよび属性のリストの自動生成
- キーワードをベースとするファンクション・キー・ストリング

HACL でまだ実装されていない EHLLAPI 機能には、次のものがあります。

- 構造化フィールドのサポート
- OIA 文字イメージ
- 表示スペースのロック/アンロック

セッション ID

HACL 体系は、52 個のセッションに限定されていません。したがって、EHLLAPI で使われるような単一の文字セッション ID は適していません。HACL では、アプリケーションに対して 特定の意味をもたない単純な 32 ビット値である接続ハンドルの概念を使用しています。接続ハンドルは、個々の接続 (セッション) を固有に識別します。1 つの接続ハンドルを使用して、複数のスレッドおよびプロセスにおいて 同一の接続を参照することができます。

特定の接続を参照する必要のあるすべての HACL オブジェクトおよび メソッドは、接続ハンドルを受け入れます。さらに、後方互換性のためと、エミュレーター・ユーザー・インターフェース (ハンドルを表示しません) から 参照できるようにするため、一部のオブジェクトとメソッドは 従来のセッション ID も受け入れます。アプリケーションは、ECLConnList オブジェクトとの接続を列挙することによって 接続ハンドルを取得できます。それぞれの接続は、ECLConnection オブジェクトによって 表されます。ECLConnection::GetHandle メソッドを使用すると、個々の接続に関連した ハンドルを取り出すことができます。

アプリケーションで、接続名 (EHLLAPI 短縮セッション ID) の代わりに 接続ハンドルを使用することを強くお勧めします。HACL の将来の設定では、接続名を使用するアプリケーションは、52 個を超えるセッションにアクセスできなくなる場合があります。場合によっては、名前を使用する必要があるかもしれません。例えば、アプリケーションが利用する 特定のセッションの名前を入力する必要があるときなどです。以下の C++ の例では、セッション名を

入力します。するとアプリケーションは、接続リスト内でその接続を検出してから、そのセッション用の PS および OIA オブジェクトを作成します。

```
ECLConnList        ConnList;  // Connection list
ECLConnection       *ConnFound;  // Ptr to found connection
ECLPS           *PS;        // Ptr to PS object
ECLOIA             *OIA;      // Ptr to OIA object
char              UserRequestedID;

//... user inputs a session name (A-Z or a-z) and it is put
//... into the UserRequesteID variable.  Then...

ConnList.Refresh();  // Update list of connections
ConnFound = ConnList.FindConnection(UserRequestedID);
if (ConnFound == NULL) {
  // Session name given by user does not exist...
}
else {
  // Create PS and OIA objects using handle of the
  // connection just found:
  PS = new ECLPS(ConnFound.GetHandle());
  OIA= new  ECLOIA(ConnFound.GetHandle());

  // The following would also work, but is not the
  // preferred method:
  PS = new ECLPS(UserRequestedID);
  OIA= new ECLOIA(UserRequestedID);
}
```

例に示された、PS および OIA オブジェクトを作成する第 2 の方法は 望ましくありません。ハンドルではなくセッション名を使用するからです。この場合、このコード・セクションに暗黙の 52 個のセッション制限が設定されます。上記の第 1 の例を使用すると、そのコード・セクションを さまざまなセッションに使用することができます。

表示スペース・モデル

HACL の表示スペース・モデルは、EHLLAPI のものより簡単に使うことができます。HACL 表示スペースは、 おのおのが 1 つのデータ・タイプを持ついくつかのプレーンで構成されます。プレーンには、以下のものがあります。

- テキスト
- フィールド属性
- カラー
- 拡張属性

プレーンは、すべて同サイズであり、ホスト表示スペース内の 各文字位置につき 1 バイトずつを含んでいます。アプリケーションは、ECLPS::GetScreen メソッドを使用して、必要な任意のプレーンを取得できます。

このモデルは、バッファー内で表示スペースのテキストおよび 非テキストのデータがしばしばインターリーブされる EHLLAPI とは異なります。アプリケーションは、どのタイプのデータを取り出すかを指定する EHLLAPI セッション・パラメーターを設定してから、 次にそのデータをバッファーにコピーするために別の呼び出しを行わなければなりません。HACL モデルを使用すると、アプリケーションは、1 回の呼び出しで必要なデータを取得する ことができ、1 つのバッファー内で別々のデータ型が混ざり合うことはありません。

SendKey インターフェース

ホストにキー・ストロークを 送信する HACL メソッド (ECLPS::Sendkeys) は、EHLLAPI の SendKey 関数に似ています。ただし、EHLLAPI では、Enter、PF1、および Backtab などの非テキスト・キーを表すのに 暗号エスケープ・コードが使われます。ECLPS オブジェクトは、そのようなキー・ストロークを表すのに、ブラケットで囲んだキーワードを使用します。例えば、次に示す C++ サンプルでは、現行カーソル位置に ABC の文字を 入力し、その後に Enter キーが続きます。

```
ECLPS  *PS;

PS = new ECLPS('A');   // Get PS object for "A"
PS->SendKeys("ABC[enter]");  // Send keystrokes
```

イベント

EHLLAPI には、特定のイベントについてアプリケーションが非同期通知を受け取るためのいくつかの手段が備わっています。しかし、イベント・モデル相互に一貫性はない (セマフォーを使用する イベントもあれば、ウィンドウ・システム・メッセージを使用する イベントもある) ため、アプリケーションは責任をもって イベント・スレッドをセットアップして管理しなければなりません。HACL では、すべてのイベント処理が単純化され、すべてのイベント・タイプを通して一貫性が保たれます。アプリケーションは、明示的に複数の実行スレッドを作成する 必要はなく、HACL が内部でスレッド化を処理します。

ただし、別の実行スレッドでイベント・プロシージャーが呼び出されることに留意していなければなりません。イベント・プロシージャーからのアクセス時には、 ダイナミック・アプリケーション・データへのアクセスを同期化しなければなりません。イベント・スレッドは、アプリケーションがイベントを登録すると生成され、 イベントが登録抹消されると終了します。

PS 接続/切断およびマルチスレッド化

EHLLAPI アプリケーションは、別々のセッションへの接続を管理するには、ConnectPS および DisconnectPS EHLLAPI 機能を呼び出さなければなりません。アプリケーションは、セッションに永続的に接続されたままにならないようにするため、 慎重にコーディングしなければなりません。 セッションは、すべての EHLLAPI アプリケーションで共有しなければならないからです。また、使用する他の EHLLAPI 機能によっては、 アプリケーションがセッションに接続されていることも事前に確認しなければなりません。

HACL では、アプリケーションが明示的にセッションを接続または切断する必要は ありません。すべての HACL オブジェクトは、その作成時に特定の接続 (セッション) に関連付けられています。アプリケーションが、別の接続にアクセスするには、それぞれのためのオブジェクトを作成するだけでよいのです。例えば、次に示す例は、キー・ストローク ABC をセッション A に送ってから、次に DEF をセッション B に、さらに次に Enter キーをセッション A に送ります。 EHLLAPI プログラムでは、アプリケーションは、セッションを 1 つずつ接続/切断しなければなりません。 一度に 1 つのセッションとしか対話できないからです。HACL アプリケーションは、次のように必要な任意の順序でオブジェクトを使用することができます。

```
ECLPS  *PSA, *PSB;

PSA = new ECLPS('A');
```

```
PSB = new ECLPS('B');

PSA->Sendkeys("ABC");
PSB->Sendkeys("DEF");
PSA->Sendkeys("[enter]");
```

複数の接続 (セッション) と対話するアプリケーションの場合、これによって、複数の接続を管理するのに必要なコードを大幅に単純化することができます。

また EHLLAPI では、作業セッションが 1 つであることに加えて、 アプリケーションのマルチスレッド特性に対する制約もあります。EHLLAPI インターフェースを 呼び出すスレッドが複数あるアプリケーションの場合、表示スペースへの接続と 切断は慎重に管理する必要があり、複数のスレッドがあってもアプリケーションは一度に 1 つの セッションとしか対話できません。

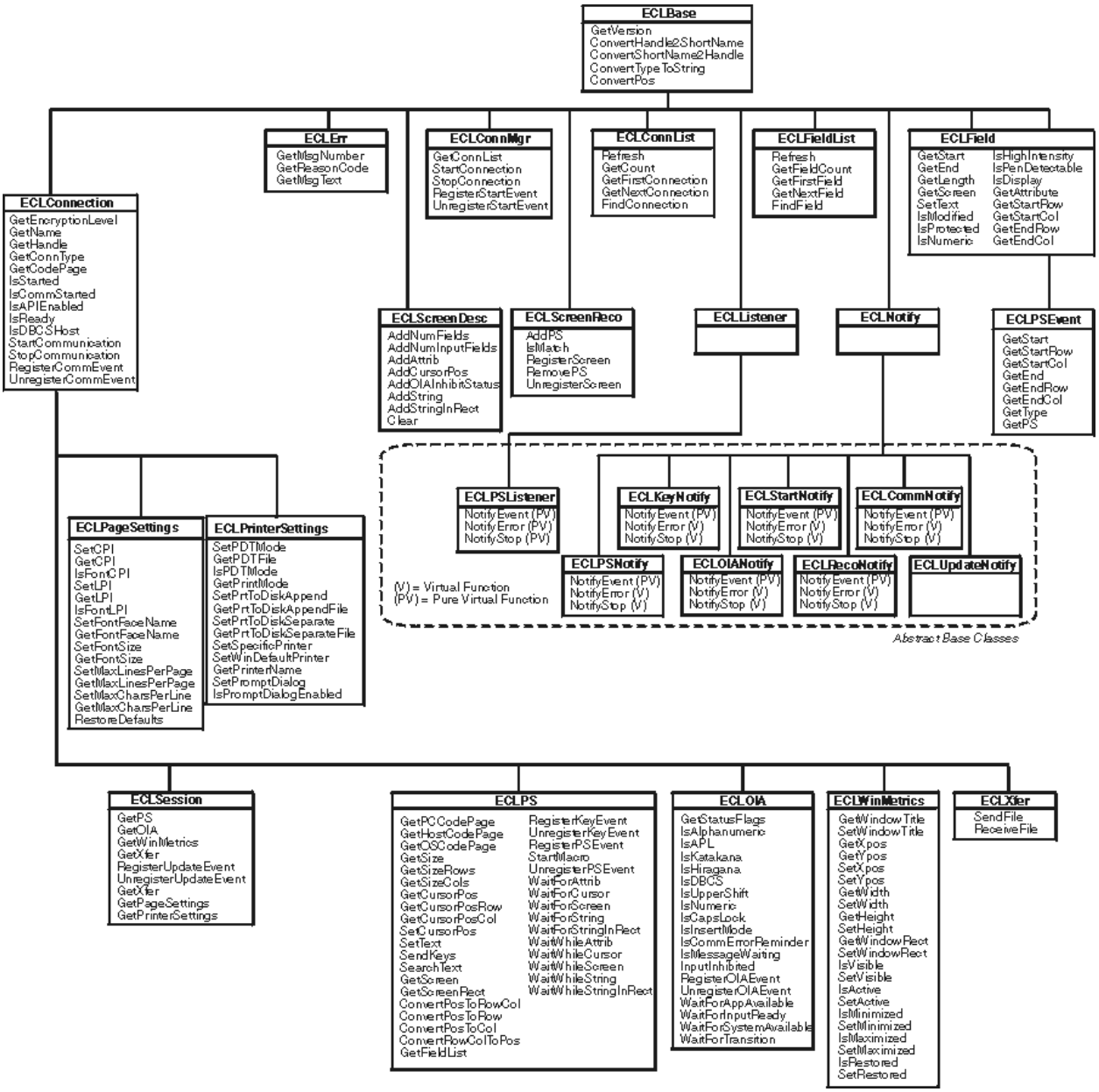
ECLPS では、マルチスレッド化に関してアプリケーションは特に制約を受けません。アプリケーションは、任意の数のスレッド上で任意の数のセッションと並列対話することができます。

第 2 章. ホスト・アクセス・クラス・ライブラリー C++

この C++ クラス・ライブラリーは、ホスト接続をオブジェクト指向によって抽象化したものを完全な形で提供します。それには、ホスト表示スペース (画面) での読み書き、画面上のフィールドのエミュレーション、状況情報についてのオペレーター標識域 (OIA) の読み取り、ビジュアル・エミュレーター・ウィンドウに関する情報のアクセスと更新、ファイルの転送、および重要イベントの非同期通知の実行が含まれます。クラス・ライブラリーは、Microsoft® Visual C++® コンパイラーをサポートしています。

ホスト・アクセス・クラス・ライブラリーの C++ 層は、クラス階層に配置された多数の C++ クラスで構成されています。図 2 : ホスト・アクセス・クラス・オブジェクト (ページ 15) は、ホスト・アクセス・クラス・ライブラリーの C++ 層の C++ 継承階層を示しています。この図では、すべてのオブジェクトは、それぞれすぐ上のクラスから継承します。

図 2. ホスト・アクセス・クラス・オブジェクト



また、図 2 : ホスト・アクセス・クラス・オブジェクト (ページ 15) は、各クラスのすべてのメンバー関数も 示しています。各クラスについて示されている関数に加え、それぞれのクラスは 親クラスの関数をすべて継承していることに注意してください。例えば、 関数 IsReady() は、ECLSession、 ECLPS、ECLOIA、ECLWinMetrics、 および ECLXfer クラスでも使用できます。

各クラスについて、以下のセクションで簡潔に説明します。詳細については、この章の各クラスの説明を参照してください。

この章の例はすべて、ECLSAMPS.CPP ファイルにあります。このファイルを使用して、 サポートされているコンパイラーでサンプルをコンパイルしたり、実行したりできます。

以下に、ホスト・アクセス・クラス・ライブラリー C++ クラスについて概説します。各クラス名は、 ホスト・アクセス・クラス・ライブラリーの共通の接頭部である ECL で始まります。

- ECLBase (ECLBase クラス (ページ 18) ページ) は、 すべての ECL オブジェクトの基本クラスです。これは、接続名やハンドルの変換などの特定の基本ユーティリティー ・メソッドを提供します。すべての ECL オブジェクトはこのクラスを継承するため、 これらのメソッドはどの ECL メソッドでも使用することができます。
- ECLConnection (ECLConnection クラス (ページ 24) ページ) は、単一の Z and I Emulator for Windows 接続を表し、接続状況、接続タイプ (例えば、3270 や 5250)、および接続名と接続ハンドルなどの接続情報を含んでいます。またこのクラスは、ECLPS や ECLOIA などの接続別の ECL オブジェクトすべての 基本クラスでもあります。
- ECLConnList (ECLConnList クラス (ページ 39) ページ) には、 オブジェクトが作成されたときまたは最後に Refresh メソッドが呼び出されたときに存在していた、すべての Z and I Emulator for Windows 接続のリストが入っています。それぞれの接続は、ECLConnection オブジェクトによって 表されます。
- ECLConnMgr (ECLConnMgr クラス (ページ 47) ページ) は、ECLConnList オブジェクトを使用する現在稼働中のすべて Z and I Emulator for Windows 接続 (ウィンドウ) を列挙します。またこれは、新たに接続を開始または停止するためのメソッドも提供します。
- ECLCommNotify (ECLCommNotify クラス (ページ 54) ページ) は、 ホストへの接続が切断されたり接続されたりするたびに、通知を受け取るために アプリケーションで使用できる通知クラスです。これは、接続の状況をモニターして、 接続が不意に切断されたときにアクションをとるのに使用することができます。
- ECLErr (ECLErr クラス (ページ 59) ページ) は、 ホスト・アクセス・クラス・ライブラリー・クラスから実行時エラー情報を戻すためのメソッドを提供します。
- ECLField (ECLField クラス (ページ 63) ページ) には、 フィールド属性、フィールド・カラー、画面上の位置、 または長さなどの、画面上の単一のフィールドについての情報が入っています。入力フィールドを更新するためのメソッドも提供されます。
- ECLFieldList (ECLFieldList クラス (ページ 80) ページ) に は、ECLField オブジェクトの集合が入っています。Refresh メソッドを呼び出すと、 現行ホスト画面が検査され、フィールド・リストが抽出され、 それを使用して ECLField オブジェクトのリストが作成されます。アプリケーションは、この集合を使うことにより、 リストそのものを作成しなくてもフィールドを管理することができます。
- ECLKeyNotify (ECLKeyNotify クラス (ページ 87) ページ) は、 キー・ストローク・イベントについて通知を受けるためにアプリケーション が使用できる通知クラスです。アプリケーションは、キー・ストロークをフィルター処理 (除去) したり、 他のキー・ストロークに置換したり、廃棄したりできます。
- ECLListener (ECLListener クラス (ページ 93) ページ) は、 すべての新規 HACL イベント・リスナー・オブジェクトの基本クラスです。これは、 すべてのリスナー・オブジェクトに共通の機能を提供します。
- ECLOIA (ECLOIA クラス (ページ 93) ページ) によって、シフト標識、 入力禁止状態、通信エラーなど、オペレーター状況の情報へアクセスできます。
- ECLOIANotify (ECLOIANotify クラス (ページ 106) ページ) は、 抽象基本クラスです。アプリケーションはこのクラスから派生したオブジェクトを 作成して、OIA の変更の通知を受け取ります。

- ECLPS (ECLPS クラス (ページ 109) ページ) は、 単一の接続の表示スペース (画面) を表します。これは、データ・プレーンの形式で画面内容のコピーを取得するためのメソッドを含んでいます。各プレーンは、テキスト、フィールド属性、およびカラー属性など、表示スペース特定の側面を表します。表示スペース内のストリングを検索したり、 キー・ストロークをホストに送信したり、ホスト・カーソル位置を取得および設定するため、 さらにその他の多くの機能のためのメソッドが提供されます。また、画面上のフィールド・リストを列挙するのに使える ECLFieldList オブジェクトも提供されます。
- ECLPSEvent (ECLPSEvent クラス (ページ 154) ページ) は、 表示スペースが更新されたときに PS イベント・リスナーに渡される イベント・オブジェクトです。これには、更新の理由および画面の更新個所を含んだ イベントについての情報が入っています。
- ECLPSListener (ECLPSListener クラス (ページ 159) ページ) は、 抽象基本クラスです。アプリケーションはこのクラスから派生したオブジェクトを 作成して、ECLPSEvent オブジェクトにより提供されるすべての情報を使用して 表示スペース更新イベントを受信します。
- ECLPSNotify (ECLPSNotify クラス (ページ 162) ページ) は、 抽象基本クラスです。アプリケーションはこのクラスから派生したオブジェクトを 作成して、最少の情報を使用して表示スペース更新の通知を受信します。
- ECLRecoNotify (ECLRecoNotify クラス (ページ 165) ページ) は、 抽象基本クラスです。アプリケーションはこのクラスから派生したオブジェクトを 作成して、画面認識の通知を受信します。
- ECLScreenDesc (ECLScreenDesc クラス (ページ 168) ページ) は、 単一のホスト画面を記述するのに使用するクラスです。このとき、画面記述クラス・オブジェクト を使用して、この記述されたホスト画面が表示されるときにイベントを起動したり、あるいは 特定のホスト画面を同期して待機します。
- ECLScreenReco (ECLScreenReco クラス (ページ 178) ページ) は、 画面説明オブジェクトのセットを収集したり、収集された画面のいずれか が表示スペースに表示されるときに非同期イベントを生成するのに 使用されるクラスです。
- ECLSession (ECLSession クラス (ページ 183) ページ) には、 すべての接続固有オブジェクトの集合が入っています。ECLSession を使うと、特定の接続用の完全セットのオブジェクトを簡単に作成することができます。
- ECLStartNotify (ECLStartNotify クラス (ページ 191) ページ) は、 接続の開始時または停止時に必ず通知を受けられるようにするために アプリケーションが使用できる通知クラスです。これは、システム状況をモニターして、接続が不意にクローズされたときに アクションをとるために使用することができます。
- ECLUpdateNotify (ECLUpdateNotify クラス (ページ 196) ページ) は、 ホスト画面または OIA が更新されるたびに通知を受けられるようにする ためにアプリケーションが使用できる通知クラスです。
- ECLWinMetrics (ECLWinMetrics クラス (ページ 196) ページ) は、 エミュレーションが実行される物理ウィンドウを表示します。ウィンドウの状態 (最小化、最大化、元のサイズに戻す)、ウィンドウ・サイズ、 および可視属性を取得および設定するためのメソッドが提供されます。
- ECLXfer (ECLXfer クラス (ページ 219) ページ) は、 ホストとの間での接続を介したファイルのやりとりを開始します。
- ECLPageSettings (ECLPageSettings クラス (ページ 226) ページ) は、エミュレーター・セッションの**「ファイル」>「ページ設定」**ダイアログの設定を制御および検索します。
- ECLPrinterSettings (ECLPrinterSettings クラス (ページ 238) ページ) は、エミュレーター・セッションの**「ファイル」>「プリンター設定」**ダイアログの設定を制御および検索します。

作成、C++ ECL プログラムの

このセクションでは、ECL を使用する C++ プログラムの作成方法のメカニズムについて説明します。ソース・コードの準備、コンパイル、およびリンクの各要件についても述べます。

Microsoft Visual C++

以下のセクションでは、ECL を使用する Microsoft® Visual C++ アプリケーションの作成、コンパイル、およびリンクの方法について説明します。現在、Z and I Emulator for Windows は、Microsoft® Visual C++ コンパイラー・バージョン 4.2 以降をサポートします。

ソース・コードの準備

ECL クラスを使用するプログラムでは、クラス定義その他のコンパイル時情報を得る ために、ECL ヘッダー・ファイルを含めなければなりません。アプリケーションに必要なヘッダー・ファイルのサブセットのみを 含めても構いませんが、単純化のため、ECLALL.HPP ファイルを 使用するすべての ECL ヘッダー・ファイルを アプリケーションに含めるようお勧めします。

ECL オブジェクトまたは定義に対する参照を含んだ すべての C++ ソース・ファイルでは、最初の参照の前に、 次に示すステートメントを付けなければなりません。

```
#include "eclall.hpp"
```

コンパイル

コンパイラーに対して、ECL ヘッダー・ファイルの入った ZIEWin サブディレクトリーを検索するよう指示しなければなりません。そのためには、/I コンパイラー・オプションを使用するか、 または「Developer Studio Project Setting」ダイアログを使用します。

/MT (実行可能ファイルの場合) または /MD (DLL の場合) コンパイラー・ オプションを使用して、アプリケーションをマルチスレッド実行用にコンパイル しなければなりません。

リンク

リンカーに対して、ECL リンク可能ライブラリー・ファイル (PCSECLVC.LIB) を 含めるよう指示しなければなりません。そのためには、リンカー・コマンド行でライブラリー・ファイルの完全修飾名を指定するか、 または「Developer Studio Project Settings」ダイアログを使用します。

実行中

ECL を使用するアプリケーションを実行するとき、ZIEWin ライブラリーがシステム・パス内で見つからなければなりません。デフォルトでは、ZIEWin ディレクトリーは ZIEWin のインストール時にシステム・パスに加えられます。

ECLBase クラス

ECLBase は、すべての ECL オブジェクトの基本クラスです。これは、接続名やハンドルの変換などの特定の基本ユーティリティー ・メソッドを提供します。すべての ECL オブジェクトはこのクラスを継承するため、 これらのメソッドはどの ECL メソッドでも使用することができます。

アプリケーションは、このクラスのオブジェクトを直接作成すべきではありません。

派生

なし

ECLBase メソッド

以下に、ECLBase クラスにおいて有効なメソッドを示します。

int GetVersion(void) char ConvertHandle2ShortName(long ConnHandle) long ConvertShortName2Handle(char Name) void ConvertTypeToString(int ConnType,char *Buff) inline void ConvertPos(ULONG Pos, ULONG *Row, ULONG *Col, ULONG PSCols)

GetVersion

このメソッドは、ホスト・アクセス・クラス・ライブラリーのバージョンを戻します。戻される値は、小数点を使用したバージョン番号に 100 を乗算したものです。例えば、1.02 は 102 として戻されます。

プロトタイプ

int GetVersion(void)

パラメーター

なし

戻り値

整数

ECL バージョン番号に 100 を乗算したもの。

例

```
//---------------------------------------------------------------
// ECLBase::GetVersion
//
// Display major version number of ECL library.
//---------------------------------------------------------------
```

```
void Sample2() {

if (ECLBase::GetVersion() >= 200) {
  printf("Running version 2.0 or later.\n");
}
else {
  printf("Running version 1.XX\n");
}

} // end sample
```

ConvertHandle2ShortName

このメソッドは、指定された ECL 接続ハンドルの名前 (A から Z、または a から z) を戻します。指定された接続が存在しない場合でも、この関数は名前を戻すことがある ことに注意してください。

プロトタイプ

char ConvertHandle2ShortName(long ConnHandle)

パラメーター

long ConnHandle

ECL 接続のハンドル。

戻り値

char

A から Z、または a から z の範囲の、ECL 接続の名前。

例

```
//---------------------------------------------------------------------
// ECLBase::ConvertHandle2ShortName
//
// Display name of first connection in the connection list.
//---------------------------------------------------------------------
void Sample3() {

ECLConnList  ConnList;
long Handle;
char Name;

if (ConnList.GetCount() > 0) {
  // Print connection name of first connection in the
  // connection list.
  Handle = ConnList.GetFirstConnection()->GetHandle();
  Name = ConnList.ConvertHandle2ShortName(Handle);
  printf("Name of first connection is: %c \n", Name);
}
else printf("There are no connections.\n");
```

```
} // end sample
```

ConvertShortName2Handle

このメソッドは、指定された名前の付いた ECL 接続の接続ハンドルを戻します。名前は、A から Z、または a から z の範囲でなければなりません。指定された接続が存在しない場合でも、この関数はハンドルを戻すことがある ことに注意してください。

プロトタイプ

char ConvertShortName2Handle(char Name)

パラメーター

char Name

A から Z の、または a から z の範囲の、ECL 接続の名前。

戻り値

char

ECL 接続のハンドル。

例

```
//---------------------------------------------------------------------
// ECLBase::ConvertShortName2Handle
//
// Display handle of connection 'A'.
//---------------------------------------------------------------------
void Sample4() {

ECLConnList  ConnList;
long Handle;
char Name;

Name = 'A';
Handle = ConnList.ConvertShortName2Handle(Name);
printf("Handle of connection A is: 0x%lx \n", Handle);

} // end sample
```

ConvertTypeToString

このメソッドは、ECLConnection::GetConnType() によって戻された接続タイプを ヌル終了ストリングに変換します。戻されるストリングは、言語依存ではありません。

ConnType	戻されるストリング
HOSTTYPE_3270DISPLAY	"3270 DISPLAY"
HOSTTYPE_3270PRINTER	"3270 PRINTER"
HOSTTYPE_5250 DISPLAY	"5250 PRINTER"
HOSTTYPE_5250PRINTER	"5250 PRINTER"
HOSTTYPE_VT	"ASCII TERMINAL"
HOSTTYPE_PC	"PC SESSION"
他の任意の値	UNKNOWN

プロトタイプ

void ConvertTypeToString(int ConnType,char *Buff)

パラメーター

int ConnType

接続タイプ。ECLBASE.HPP に定義されている HOSTTYPE_* 定数のいずれか でなければなりません。

char *Buff

ストリングが戻される ECLBase.hpp に定義された サイズ TYPE_MAXSTRLEN のバッファー。

戻り値

なし

例

```
//---------------------------------------------------------------------
// ECLBase::ConvertTypeToString
//
// Display type of connection 'A'.
//---------------------------------------------------------------------
void Sample5() {

ECLConnection  *pConn;
char           TypeString[21];

pConn = new ECLConnection('A');

pConn->ConvertTypeToString(pConn->GetConnType(), TypeString);
// Could also use:
// ECLBase::ConvertTypeToString(pConn->GetConnType(), TypeString);

printf("Session A is a %s \n", TypeString);

delete pConn;

} // end sample
```

ConvertPos

このメソッドは、表示スペースの位置および幅が分かっている 場合に、ECL 位置座標を行/列座標に変換するためのインライン関数 (マクロ) です。アプリケーションで表示スペースの幅が既に分かっている (または 推測できる) 場合、この関数の方が ECLPS::ConvertPosToRowCol() よりも早いです。

プロトタイプ

inline void ConvertPos(ULONG Pos,ULONG *Row,ULONG *Col,ULONG PSCols).

パラメーター

ULONG Pos

変換 (入力) される線形定位置座標。

ULONG *Row

指定位置 (出力) について戻された行番号を指すポインター。

ULONG *Col

指定位置 (出力) について戻された桁番号を指すポインター。

ULONG *PSCols

ホストの表示スペース (入力) 内の桁数。

戻り値

なし

例

```
//-------------------------------------------------------------------
// ECLBase::ConvertPos
//
// Display row/column coordinate of a given point.
//-------------------------------------------------------------------
void Sample6() {

ECLPS     *pPS;
ULONG     NumRows, NumCols, Row, Col;

try {
  pPS = new ECLPS('A');

  pPS->GetSize(&NumRows, &NumCols);  // Get height and width of PS

  // Get row/column coordinate of position 81
  ECLBase::ConvertPos(81, &Row, &Col, NumCols);
  printf("Position 81 is row %lu, column %lu \n", Row, Col);

  delete pPS;
}
catch (ECLErr Err) {
```

```
  printf("ECL Error: %s\n", Err.GetMsgText());
}

} // end sample
```

ECLConnection クラス

ECLConnection には、特定の接続についての接続関連情報が入っています。このオブジェクトは、アプリケーションが直接作成することが できますが、ECLConnList オブジェクトによって間接的に作成 したり、ECLConnection から継承する任意のオブジェクト (例えば、ECLSession) の作成時に作成したりすることもできます。

このオブジェクトのメソッドから戻される情報は、 そのメソッドが呼び出された時点のものです。

ECLConnection は、ECLSession、ECLPS、ECLOIA、ECLWinMetrics、 および ECLXfer によって継承されます。

派生

ECLBase > ECLConnection

ECLConnection メソッド

以下に、ECLConnection クラスにおいて有効なメソッドを示します。

ECLConnection(char ConnName) ECLConnection(long ConnHandle) ~ECLConnection() long GetHandle() int GetConnType() int GetEncryptionLevel() char GetName() BOOL IsStarted() BOOL IsCommStarted() BOOL IsAPIEnabled() BOOL IsReady() unsigned int GetCodePage() void StartCommunication() void StopCommunication() void RegisterCommEvent(ECLCommNotify *NotifyObject, BOOL InitEvent = TRUE) void UnregisterCommEvent(ECLCommNotify *NotifyObject)

ECLConnection コンストラクター

このメソッドは、接続名または接続ハンドルから ECLConnection オブジェクトを作成します。

プロトタイプ

ECLConnection(long ConnHandle)

ECLConnection(char ConnName)

パラメーター

long ConnHandle

接続オブジェクトを作成するための接続ハンドル。

char ConnName

接続オブジェクトを作成するための接続名 (A から Z、または a から z)。

戻り値

なし

例

```
//---------------------------------------------------------------------
// ECLConnection::ECLConnection    (Constructor)
//
// Create two connection objects for connection 'A', one created
// by name, the other by handle.
//---------------------------------------------------------------------
void Sample7() {

ECLConnection   *pConn1, *pConn2;
long              Hand;

try {
  pConn1  = new ECLConnection('A');
  Hand    = pConn1->GetHandle();
  pConn2  = new ECLConnection(Hand);  // Another ECLConnection for 'A'

  printf("Conn1 is for connection %c, Conn2 is for connection %c.\n",
         pConn1->GetName(), pConn2->GetName());

  delete pConn1;  // Call destructors
  delete pConn2;
}
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}

} // end sample
```

ECLConnection デストラクター

このメソッドは、ECLConnection オブジェクトを破棄します。

プロトタイプ

~ECLConnection()

パラメーター

なし

戻り値

なし

例

```
//---------------------------------------------------------------------
// ECLConnection::~ECLConnection    (Destructor)
//
// Create two connection objects, then delete both of them.
//---------------------------------------------------------------------
void Sample8() {

ECLConnection   *pConn1, *pConn2;
long              Hand;

try {
  pConn1  = new ECLConnection('A');
  Hand    = pConn1->GetHandle();
  pConn2  = new ECLConnection(Hand);  // Another ECLConnection for 'A'

  printf("Conn1 is for connection %c, Conn2 is for connection %c.\n",
         pConn1->GetName(), pConn2->GetName());

  delete pConn1;  // Call destructors
  delete pConn2;
}
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}

} // end sample
```

GetCodePage

このメソッドは、 接続の構成の対象となっているホスト・コード・ページを戻します。

プロトタイプ

unsigned int GetCodePage()

パラメーター

なし

戻り値

unsigned int

接続のホスト・コード・ページ。

例

```
//-------------------------------------------------------------------
// ECLConnection::GetCodePage
//
// Display host code page for each ready connection.
//-------------------------------------------------------------------
void Sample16() {

ECLConnection *Info;     // Pointer to connection object
ECLConnList ConnList;    // Connection list object

for (Info = ConnList.GetFirstConnection();
     Info != NULL;
     Info = ConnList.GetNextConnection(Info)) {

  if (Info->IsReady())
    printf("Connection %c is configured for host code page %u.\n",
            Info->GetName(), Info->GetCodePage());
}

} // end sample
```

GetHandle

このメソッドは、接続ハンドルを戻します。このハンドルは、接続を固有識別し、 また接続ハンドルを必要とする他の ECL 関数内で使用することもできます。

プロトタイプ

long GetHandle()

パラメーター

なし

戻り値

long

ECLConnection オブジェクトの接続ハンドル。

例

以下の例は、接続リスト内の最初の接続のハンドルがどのように戻されるかを示します。

```
//---------------------------------------------------------------------
// ECLConnection::GetHandle
//
// Get the handle of connection 'A' and use it to create another
// connection object.
//---------------------------------------------------------------------
void Sample9() {

ECLConnection   *pConn1, *pConn2;
long              Hand;

try {
  pConn1  = new ECLConnection('A');
  Hand    = pConn1->GetHandle();
  pConn2  = new ECLConnection(Hand);  // Another ECLConnection for 'A'

  printf("Conn1 is for connection %c, Conn2 is for connection %c.\n",
         pConn1->GetName(), pConn2->GetName());

  delete pConn1;  // Call destructors
  delete pConn2;
}
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}

} // end sample
```

GetConnType

このメソッドは、接続タイプを戻します。この接続タイプは、時間の経過とともに変わることがあります (例えば、別のホスト用に接続が再構成されることがあります)。アプリケーションでは、接続タイプは固定のものと見なしてはなりません。戻される接続タイプについては、以下の項を参照してください。

注: ECLBase::ConvertTypeToString 関数は、 接続タイプをヌル終了ストリングに変換します。

プロトタイプ

int GetConn Type()

パラメーター

なし

戻り値

整数

接続タイプ定数 (HOSTBASE.HPP からの HOSTTYPE_*) です。下の表は、戻される値とその意味を示します。

戻される値	意味
HOSTTYPE_3270DISPLAY	3270 表示装置
HOSTTYPE_3270PRINTER	3270 印刷装置
HOSTTYPE_5250DISPLAY	5250 ディスプレイ
HOSTTYPE_5250PRINTER	5250 プリンター
HOSTTYPE_VT	ASCII VT ディスプレイ
HOSTTYPE_UNKNOWN	未知の接続タイプ

例

以下の例は、GetConnType メソッドを使用してどのように接続タイプを戻すかを示します。

```
//-----------------------------------------------------------------------
// ECLConnection::GetConnType
//
// Find the first 3270 display connection in the current list of
// all connections.
//-----------------------------------------------------------------------
void Sample10() {

ULONG      i;             // Connection counter
ECLConnList ConnList;     // Connection list object
ECLConnection *Info=NULL; // Pointer to connection object

for (i=0; i<ConnList.GetCount(); i++) {

  Info = ConnList.GetNextConnection(Info);
  if (Info->GetConnType() == HOSTTYPE_3270DISPLAY) {
    // Found the first 3270 display connection
    printf("First 3270 display connection is '%c'.\n",
            Info->GetName());
    return;
  }

} // for
printf("Found no 3270 display connections.\n");

} // end sample
```

GetName

このメソッドは、接続の接続名 (単一の A から Z、または a から z の英字) を戻します。またこの名前は、EHLLAPI セッション ID に対応します。

プロトタイプ

char GetName()

パラメーター

なし

戻り値

char

接続の短縮名。

例

以下の例は、GetName メソッドを使用してどのように接続名を戻すかを示します。

```
//---------------------------------------------------------------------
// ECLConnection::GetName
//
// Find the first 3270 display connection in the current list of
// all connections and display its name (session ID).
//---------------------------------------------------------------------
void Sample11() {

ULONG      i;             // Connection counter
ECLConnList ConnList;     // Connection list object
ECLConnection *Info=NULL; // Pointer to connection object

for (i=0; i<ConnList.GetCount(); i++) {

  Info = ConnList.GetNextConnection(Info);
  if (Info->GetConnType() == HOSTTYPE_3270DISPLAY) {
    // Found the first 3270 display connection, display the name
    printf("First 3270 display connection is '%c'.\n",
            Info->GetName());
    return;
  }

} // for
printf("Found no 3270 display connections.\n");

} // end sample
```

GetEncryptionLevel

このメソッドは、現行接続の暗号化レベルを戻します。

プロトタイプ

int GetEncryptionLevel()

パラメーター

なし

戻り値

整数

暗号化レベルの定数。下の表は、戻される値とその意味を示します。

戻される値	意味
ENCRYPTION_NONE	暗号化なし
ENCRYPTION_40BIT	40 ビット暗号化
ENCRYPTION_56BIT	56 ビット暗号化
ENCRYPTION_128BIT	128 ビット暗号化
ENCRYPTION_168BIT	168 ビット暗号化
ENCRYPTION_NOKEY	鍵なしの暗号化

例

以下の例は、GetEncryptionLevel メソッドを使用してどのように暗号化レベルを戻すかを示します。

```
//-------------------------------------------------------
// ECLConnection::GetEncryptionLevel
//
// Display the encryption level of session A
//
//-------------------------------------------------------
void SampleEL()
{
int EncryptionLevel = 0;   //Encryption Level
ECLConnection * Info = NULL;  //Pointer to connection object

Info = new ECLConnection('A');
If (Info != NULL)
{
 EncryptionLevel = Info->GetEncryptionLevel();
 switch (EncryptionLevel)
 {
 case ENCRYPTION_NONE:
  printf("Encryption Level = None");
  break;
 case ENCRYPTION_40BIT:
  printf("Encryption Level = 40 BIT");
```

```
  break;
 case ENCRYPTION_56BIT:
  printf("Encryption Level = 56 BIT");
  break;
 case ENCRYPTION_128BIT:
  printf("Encryption Level = 128 BIT");
  break;
 case ENCRYPTION_168BIT:
  printf("Encryption Level = 168 BIT");
  break;

     default:
 }
}
}
```

IsStarted

このメソッドは、接続が開始済みかどうかを示します。接続は、開始済みであっても、 ホストに接続されていないことがあります。接続が現在ホストに接続されているかどうかを 判別するには、IsCommStarted 関数を使用します。

プロトタイプ

BOOL IsStarted()

パラメーター

なし

戻り値

BOOL

接続が開始済みの場合は True 値、未開始の場合は False 値になります。

例

```
//---------------------------------------------------------------------
// ECLConnection::IsStarted
//
// Display list of all started connections.  Note they may or may
// not be communications-connected to a host, and may or may not
// be visible on the screen.
//---------------------------------------------------------------------
void Sample12() {

ECLConnection *Info;     // Pointer to connection object
ECLConnList ConnList;    // Connection list object

// Print list of started connections
```

```
for (Info = ConnList.GetFirstConnection();
     Info != NULL;
     Info = ConnList.GetNextConnection(Info)) {

  if (Info->IsStarted())
    printf("Connection %c is started.\n", Info->GetName());
}

} // end sample
```

IsCommStarted

このメソッドは、 接続が現在ホストに接続されているかどうかを示します (例えば、 接続においてホスト通信が活動化されているかどうかを示します)。この関数は、接続が開始されていない場合は False 値を 戻します (IsStarted (ページ 32)を参照してください)。

プロトタイプ

BOOL IsCommStarted()

パラメーター

なし

戻り値

BOOL

接続がホストに接続されている場合は True 値、 接続がホストに接続されていない場合は False 値になります。

例

```
//---------------------------------------------------------------------
// ECLConnection::IsCommStarted
//
// Display list of all started connections which are currently
// in communications with a host.
//---------------------------------------------------------------------
void Sample13() {

ECLConnection *Info;     // Pointer to connection object
ECLConnList ConnList;    // Connection list object

for (Info = ConnList.GetFirstConnection();
     Info != NULL;
     Info = ConnList.GetNextConnection(Info)) {

  if (Info->IsCommStarted())
    printf("Connection %c is connected to a host.\n", Info->GetName());
}
```

```
} // end sample
```

IsAPIEnabled

このメソッドは、接続に API を使用できるかどうかを示します。API が 使用可能になっていない接続は、ホスト・アクセス・クラス・ライブラリーでは 使用できません。この関数は、接続が開始済みでない場合は False 値を戻します。

プロトタイプ

BOOL IsAPIEnabled()

パラメーター

なし

戻り値

BOOL

API が使用可能になっている場合は True 値、API が使用可能になっていない場合は False 値になります。

例

```
//---------------------------------------------------------------------
// ECLConnection::IsAPIEnabled
//
// Display list of all started connections which have APIs enabled.
//---------------------------------------------------------------------
void Sample14() {

ECLConnection *Info;     // Pointer to connection object
ECLConnList ConnList;    // Connection list object

for (Info = ConnList.GetFirstConnection();
     Info != NULL;
     Info = ConnList.GetNextConnection(Info)) {

  if (Info->IsAPIEnabled())
    printf("Connection %c has APIs enabled.\n", Info->GetName());
}

} // end sample
```

IsReady

このメソッドは、接続が作動可能であることを示します。 つまり、接続は開始済みで接続状態にあり、API が使用可能なことを意味します。この関数の方が、IsStarted、IsCommStarted、および IsAPIEnabled を呼び出すよりも早くて簡単です。

プロトタイプ

BOOL IsReady()

パラメーター

なし

戻り値

BOOL

接続が開始済み、CommStarted、および API が使用可能の場合は True、 そうでない場合は False。

例

```
//---------------------------------------------------------------------
// ECLConnection::IsReady
//
// Display list of all connections which are started, comm-connected
// to a host, and have APIs enabled.
//---------------------------------------------------------------------
void Sample15() {

ECLConnection *Info;     // Pointer to connection object
ECLConnList ConnList;    // Connection list object

for (Info = ConnList.GetFirstConnection();
     Info != NULL;
     Info = ConnList.GetNextConnection(Info)) {

   if (Info->IsReady())
    printf("Connection %c is ready (started, comm-connected, API
          enabled).\n", Info->GetName());
}

} // end sample
```

StartCommunication

このメソッドは、ZIEWin エミュレーターをホスト・データ・ストリームに接続します。これは、ZIEWin エミュレーター「通信」メニューを表示して「接続」を選択した場合と同じ結果になります。

プロトタイプ

void StartCommunication()

パラメーター

なし

戻り値

なし

例

```
//---------------------------------------------------------------------
// ECLConnection::StartCommunication
//
// Start communications link for any connection which is currently
// not comm-connected to a host.
//---------------------------------------------------------------------
void Sample17() {

ECLConnection *Info;     // Pointer to connection object
ECLConnList ConnList;    // Connection list object

for (Info = ConnList.GetFirstConnection();
     Info != NULL;
     Info = ConnList.GetNextConnection(Info)) {

  if (!(Info->IsCommStarted())) {
    printf("Starting comm-link for connection %c...\n", Info->GetName());
    Info->StartCommunication();
  }
}

} // end sample
```

StopCommunication

このメソッドは、ZIEWin エミュレーターをホスト・データ・ストリームから切断します。これは、ZIEWin エミュレーター「通信」メニューを表示して「切断」を選択した場合と同じ結果になります。

プロトタイプ

void StopCommunication()

パラメーター

なし

戻り値

なし

例

```
//---------------------------------------------------------------------
// ECLConnection::StopCommunication
//
// Stop comm-link for any connection which is currently connected
// to a host.
//---------------------------------------------------------------------
void Sample18() {

ECLConnection *Info;     // Pointer to connection object
ECLConnList ConnList;    // Connection list object

for (Info = ConnList.GetFirstConnection();
     Info != NULL;
     Info = ConnList.GetNextConnection(Info)) {

  if (Info->IsCommStarted()) {
    printf("Stopping comm-link for connection %c...\n", Info->GetName());
    Info->StopCommunication();
  }
}

} // end sample
```

RegisterCommEvent

このメンバー関数は、通信リンクのすべての接続/切断イベントの通知を受け取るための アプリケーション・オブジェクトを登録します。アプリケーションでこの関数を使用するには、ECLCommNotify クラスから派生した オブジェクトを作成しなければなりません。作成すると、そのオブジェクトを指すポインターは この登録関数に渡されます。実装上の制約事項: アプリケーションは、通信イベント通知について 1 つのオブジェクトしか登録できません。

この関数を使用して通知オブジェクトが登録されると、 その後、ホストとの接続通信リンクの接続または切断のたびに このオブジェクトが呼び出されます。通信イベントが、StartCommunication() 関数または ユーザーからの明示的な指示のどちらに起因するものであっても、 このオブジェクトは、すべての通信イベントについて通知を受け取ります。このイベントを、新しい ZIEWin 接続の開始または停止のときに起動される接続スタート・ストップ・イベントと混同してはなりません。

オプションの InitEvent パラメーターを使用すると、 オブジェクトが登録されたときに初期イベントが生成されることになります。これは、イベント・オブジェクトと、通信リンクの現在の状態を 同期化するのに便利です。InitEvent を False と指定すると、 オブジェクトが登録されても初期イベントは生成されません。このパラメーターのデフォルト値は True です。

アプリケーションは、通知オブジェクトを破棄するとき、 まず UnregisterCommEvent() を呼び出さなければなりません。オブジェクトの登録先の ECLConnection オブジェクトが破棄されると、 そのオブジェクトは自動的に登録抹消されます。

詳細については、ECLCommNotify クラス (ページ 54)の説明を参照してください。

プロトタイプ

void RegisterCommEvent(ECLCommNotify *NotifyObject, BOOL InitEvent = TRUE)

パラメーター

ECLCommNotify *NotifyObject

ECLCommNotify クラスから派生されたオブジェクトを指すポインター。

BOOL InitEvent

現在の状態を使用して初期イベントを生成します。

戻り値

なし

例

ECLConnection::RegisterCommEvent の例の 詳細については、ECLCommNotify クラス (ページ 54)を参照してください。

UnregisterCommEvent

このメンバー関数は、RegisterCommEvent() 関数を使用して通信イベント用に 事前に登録されているアプリケーション・オブジェクトの登録を抹消します。登録済みのアプリケーション通知オブジェクトの場合、 先にこの関数を呼び出してその登録を抹消しないかぎり、オブジェクトを破棄してはなりません。現在登録されている通知オブジェクトがない場合や、 登録済みオブジェクトが渡された NotifyObject でない場合、 この関数は何も実行しません (エラーになりません)。

通知オブジェクトの登録が抹消されるときは、 その NotifyStop() メンバー関数が呼び出されます。

詳細については、ECLCommNotify クラス (ページ 54)の説明を参照してください。

プロトタイプ

void UnregisterCommEvent(ECLCommNotify *NotifyObject)

パラメーター

ECLCommNotify *NotifyObject

これは、現在登録されているアプリケーション通知オブジェクトです。

戻り値

なし

例

ECLConnection::UnregisterCommEvent の例については、『ECLCommNotify クラス (ページ 54)』を参照してください。

ECLConnList クラス

ECLConnList は、特定のマシン上のすべてのホスト接続についての情報を取得します。ECLConnList オブジェクトには、 現在システムに認知されているすべての接続の集合が含まれます。

ECLConnList オブジェクトには、ECLConnection オブジェクトの集合が含まれます。この集合の各要素には、それぞれ 1 つの接続についての情報が含まれています。このリスト内の接続は、どのような状態 (例えば、停止または切断) にあるものでも構いません。すべての開始済みの接続が、このリストに示されます。ECLConnection オブジェクトには、接続の状態が入っています。

このリストは、このオブジェクトが作成されたとき、または 最後に Refresh メソッドが呼び出されたときの、一連の接続のスナップショットです。このリストは、接続の開始および停止に連動して更新されるわけではありません。アプリケーションは、ECLConnMgr オブジェクトの RegisterStartEvent メンバー を使用して、接続の開始および停止イベントの通知を受けることができます。

ELCConnList オブジェクトは、アプリケーションで直接作成できますが、 ECLConnMgr オブジェクトの作成によって間接的に作成することもできます。

派生

ECLBase > ECLConnList

使用上の注意

ECLConnList オブジェクトは、 現行接続の静的スナップショットを提供します。ECLConnList オブジェクトの作成時に、Refresh メソッドが自動的に呼び出されます。作成のすぐ後に ECLConnList オブジェクトを使用すると、その時点における接続リストの正確な表示を含めることができます。しかし、作成してから時間が経過した後で ECLConnList オブジェクトに 初めてアクセスするときは、事前にこのオブジェクトで Refresh メソッドを 呼び出さなければなりません。

アプリケーションで集合を反復するには、GetFirstConnection メソッド と GetNextConnection メソッドを使用します。GetFirstConnection および GetNextConnection から 戻されるオブジェクト・ポインターは、Refresh メンバーが呼び出されるまで、 または ECLConnList オブジェクトが破棄されるまでしか有効でありません。アプリケーションは、FindConnection 関数を使用して、 リスト内の必要な特定の接続を見つけ出すことができ ま

す。GetNextConnection と同様、戻されるポインターは、 次の Refresh または ECLConnList オブジェクトの破棄までしか有効でありません。

接続リスト内の接続の順序は、定義されていません。アプリケーションで、リストの順序を想定してはなりません。リスト内の接続の順序は、Refresh 関数が呼び出されるまでは変わりません。

ECLConnList オブジェクトは、ECLConnMgr オブジェクトが作成されると 自動的に作成されます。ただし、ECLConnMgr オブジェクトがなくても ECLConnList オブジェクトを作成できます。

ECLConnList メソッド

以下のセクションで、ECLConnList クラスにおいて 有効なメソッドについて説明します。

ECLConnection * GetFirstConnection() ECLConnection * GetNextConnection(ECLConnection *Prev) ECLConnection * FindConnection(Long ConnHandle) ECLConnection * FindConnection(char ConnName) ULONG GetCount() void Refresh()

ECLConnList コンストラクター

このメソッドは、ECLConnList オブジェクトを作成し、 接続の現行リストを使用してそのオブジェクトを初期設定します。

プロトタイプ

ECLConnList();

パラメーター

なし

戻り値

なし

例

```
//---------------------------------------------------------------------
// ECLConnList::ECLConnList        (Constructor)
//
// Dynamically construct a connection list object, display number
// of connections in the list, then delete the list.
//---------------------------------------------------------------------
void Sample19() {

ECLConnList *pConnList;  // Pointer to connection list object

try {
  pConnList = new ECLConnList();
```

```
  printf("There are %lu connections in the connection list.\n",
         pConnList->GetCount());

  delete pConnList;  // Call destructor
}
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}

} // end sample
```

ECLConnList デストラクター

このメソッドは、ECLConnList オブジェクトを破棄します。

プロトタイプ

~ECLConnList()

パラメーター

なし

戻り値

なし

例

```
//---------------------------------------------------------------------
// ECLConnList::~ECLConnList        (Destructor)
//
// Dynamically construct a connection list object, display number
// of connections in the list, then delete the list.
//---------------------------------------------------------------------
void Sample20() {

ECLConnList *pConnList;  // Pointer to connection list object

try {
  pConnList = new ECLConnList();
  printf("There are %lu connections in the connection list.\n",
         pConnList->GetCount());

  delete pConnList;  // Call destructor
}
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}

} // end sample
```

GetFirstConnection

GetFirstConnection メソッドは、 ECLConnList 集合内の最初の接続情報オブジェクトを指すポインターを戻します。この内容の詳細については、ECLConnection クラス (ページ 24)を参照してください。ECLConnList 最新表示メンバーが呼び出されるか、 または ECLConnList オブジェクトが破棄されると、 戻されたポインターは無効になります。アプリケーションは、戻されたオブジェクトを削除してはなりません。リスト内に接続がない場合は、NULL が戻されます。

プロトタイプ

ECLConnection *GetFirstConnection()

パラメーター

なし

戻り値

ECLConnection *

リスト内の最初のオブジェクトを指すポインター。リスト内に接続がない場合は、ヌルが戻されます。

例

```
//-------------------------------------------------------------------
// ECLConnection::GetFirstConnection
//
// Iterate over list of connections and display information about
// each one.
//-------------------------------------------------------------------
void Sample21() {

ECLConnection *Info;     // Pointer to connection object
ECLConnList ConnList;    // Connection list object
char TypeString[21];     // Type of connection

for (Info = ConnList.GetFirstConnection();       // Get first one
     Info != NULL;                               // While there is one
     Info = ConnList.GetNextConnection(Info)) {  // Get next one

  ECLBase::ConvertTypeToString(Info->GetConnType(), TypeString);
  printf("Connection %c is a %s type connection.\n",
         Info->GetName(), TypeString);
}

} // end sample
```

GetNextConnection

このメソッドは、 リスト内に接続があれば ECLConnList 集合内の 次の接続情報オブジェクトを指すポインターを戻します。アプリケーションは、この関数または GetFirstConnection によって前に戻された 接続を指すポインターを提供します。この内容の詳細については、ECLConnection クラス (ページ 24)を参照してください。次の ECLConnList Refresh() 呼び出し後、または ECLConnList オブジェクトの破棄後に、 戻されたポインターは無効になります。リストの終わりを超えて読み込みを行おうとした場合、NULL ポインターが戻されます。このメソッドを連続して呼び出す (呼び出すたびに前のポインターを提供する) と、 接続のリストが反復されます。最後の接続が戻されると、その後の呼び出しでは NULL ポインターが戻されます。リスト内の最初の接続を取得するには、直前の接続に NULL を指定します。

プロトタイプ

ECLConnection *GetNext Connection (ECLConnection *Prev)

パラメーター

ECLConnection *Prev

この関数 GetFirstConnection() に対する前の呼び出しから戻されるポインター、 または NULL。

戻り値

ECLConnection *

これは、次の ECLConnection オブジェクトを指すポインターですが、 リストの終わりの場合は NULL になります。

例

```
//-------------------------------------------------------------------
// ECLConnection::GetNextConnection
//
// Iterate over list of connections and display information about
// each one.
//-------------------------------------------------------------------
void Sample22() {

ECLConnection *Info;     // Pointer to connection object
ECLConnList ConnList;    // Connection list object
char TypeString[21];     // Type of connection

for (Info = ConnList.GetFirstConnection();       // Get first one
     Info != NULL;                               // While there is one
     Info = ConnList.GetNextConnection(Info)) {  // Get next one

  ECLBase::ConvertTypeToString(Info->GetConnType(), TypeString);
```

```
  printf("Connection %c is a %s type connection.\n",
         Info->GetName(), TypeString);
}

} // end sample
```

FindConnection

このメソッドは、 指定された接続を見つけるために現行接続リストを検索します。見つけたい接続は、ハンドルまたは名前を使用して指定でき ます。FindConnection メソッドには 2 種類のシグニチャーがあります。該当する接続が見つかった場合、 ECLConnection オブジェクトを指すポインターが戻されます。指定した接続がリスト内にない場合、NULL が戻されます。リストは、この関数によって自動的に更新されることはありません。そのリストが作成または最新表示された後で新しい接続が開始された場合、接続は見つかりません。戻されるポインターは、ECLConnList オブジェクトによって維持される接続リスト内のオブジェクトを指します。次の ECLConnList::Refresh の呼び出し後、または ECLConnList オブジェクトの 破棄後に、戻されたポインターは無効になります。

プロトタイプ

ECLConnection *FindConnection(Long ConnHandle),

ECLConnection *FindConnection(char ConnName)

パラメーター

Long ConnHandle

リスト内で検索される接続のハンドル。

char ConnName

リスト内で検索される接続の名前。

戻り値

ECLConnection *

要求された ECLConnection オブジェクトを指すポインター。指定した接続がリスト内にない場合、NULL が戻されます。

例

```
//-------------------------------------------------------------------
// ECLConnection::FindConnection
//
// Find connection 'B' in the list of connections.  If found, display
// its type.
//-------------------------------------------------------------------
void Sample23() {
```

```
ECLConnection *Info;     // Pointer to connection object
ECLConnList ConnList;    // Connection list object
char TypeString[21];     // Type of connection

Info = ConnList.FindConnection('B');  // Find connection by name
if (Info != NULL) {

  ECLBase::ConvertTypeToString(Info->GetConnType(), TypeString);
  printf("Connection 'B' is a %s type connection.\n",
          TypeString);
}
else printf("Connection 'B' not found.\n");

} // end sample
```

GetCount

このメソッドは、現在 ECLConnList 集合内にある接続の数を戻します。

プロトタイプ

ULONG GetCount()

パラメーター

なし

戻り値

ULONG

集合内の接続の数。

例

```
//-------------------------------------------------------------------
// ECLConnList::GetCount
//
// Dynamically construct a connection list object, display number
// of connections in the list, then delete the list.
//-------------------------------------------------------------------
void Sample24() {

ECLConnList *pConnList;  // Pointer to connection list object

try {
  pConnList = new ECLConnList();
  printf("There are %lu connections in the connection list.\n",
         pConnList->GetCount());
```

```
  delete pConnList;  // Call destructor
}
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}

} // end sample
```

最新表示

このメソッドは、 システムで現在認知されているすべての接続のリストを 使用して ECLConnList 集合を更新します。それ以前に GetNextConnection、 GetFirstConnection、 および FindConnection によって戻されたポインターはすべて無効になります。

プロトタイプ

void Refresh()

パラメーター

なし

戻り値

なし

例

```
//---------------------------------------------------------------------
// ECLConnection::Refresh
//
// Loop-and-wait until connection 'B' is started.
//---------------------------------------------------------------------
void Sample25() {

ECLConnection *Info;     // Pointer to connection object
ECLConnList ConnList;    // Connection list object
int i;

printf("Waiting up to 60 seconds for connection B to start...\n");
for (i=0; i<60; i++) {   // Limit wait to 60 seconds
  ConnList.Refresh();    // Refresh the connection list
  Info = ConnList.FindConnection('B');
  if ((Info != NULL) && (Info->IsStarted())) {
    printf("Connection B is now started.\n");
    return;
  }
  Sleep(1000L);          // Wait 1 second and try again
}
```

```
printf("Connection 'B' not started after 60 seconds.\n");

} // end sample
```

ECLConnMgr クラス

ECLConnMgr は、マシン上のすべての Z and I Emulator for Windows 接続を管理します。これは、接続の開始および停止など、接続の管理に関連したメソッドを提供します。また、システムに認知されているすべての接続のリストを列挙する ECLConnList オブジェクトも作成します (ECLConnList クラス (ページ 39)を参照)。

派生

ECLBase > ECLConnMgr

ECLConnMgr メソッド

以下に、ECLConnMgr クラスで有効なメソッドを示します。

ECLConnMgr() ~ECLConnMgr() ECLConnList * GetConnList() void StartConnection(char *ConfigParms) void StopConnection(Long ConnHandle, char *StopParms) void RegisterStartEvent(ECLStartNotify *NotifyObject) void UnregisterStartEvent(ECLStartNotify *NotifyObject)

ECLConnMgr コンストラクター

このメソッドは、ECLConnMgr オブジェクトを作成します。

プロトタイプ

ECLConnMgr()

パラメーター

なし

戻り値

なし

例

```
//-------------------------------------------------------------------
// ECLConnMgr::ECLConnMgr          (Constructor)
```

```
//
// Create a connection mangager object, start a new connection,
// then delete the manager.
//---------------------------------------------------------------------
void Sample26() {

ECLConnMgr   *pCM; // Pointer to connection manager object

try {
  pCM = new ECLConnMgr();  // Create connection manager
  pCM->StartConnection("profile=coax connname=e");
  printf("Connection 'E' started with COAX profile.\n");
  delete pCM;              // Delete connection manager
}
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}

} // end sample
```

ECLConnMgr デコンストラクター

このメソッドは、ECLConnMgr オブジェクトを破棄します。

プロトタイプ

~ECLConnMgr()

パラメーター

なし

戻り値

なし

例

```
//---------------------------------------------------------------------
// ECLConnMgr::~ECLConnMgr          (Destructor)
//
// Create a connection mangager object, start a new connection,
// then delete the manager.
//---------------------------------------------------------------------
void Sample27() {

ECLConnMgr   *pCM; // Pointer to connection manager object

try {
  pCM = new ECLConnMgr();  // Create connection manager
  pCM->StartConnection("profile=coax connname=e");
  printf("Connection 'E' started with COAX profile.\n");
  delete pCM;              // Delete connection manager
```

```
}
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}

} // end sample
```

GetConnList

このメソッドは、ECLConnList オブジェクト を指すポインターを戻します。詳しくは、ECLConnList クラス (ページ 39)を参照してください。ECLConnList オブジェクトは、ECLConnMgr オブジェクトが破棄されると破棄されます。

プロトタイプ

ECLConnList * GetConnList()

パラメーター

なし

戻り値

ECLConnList *

ECLConnList オブジェクトを指すポインター。

例

```
//-------------------------------------------------------------------
// ECLConnMgr::GetConnList
//
// Use connection manager's connection list object to display
// number of connections (see also ECLConnList::GetCount).
//-------------------------------------------------------------------
void Sample28() {

ECLConnMgr   CM; // Connection manager object

printf("There are %lu connections in the connection list.\n",
       CM.GetConnList()->GetCount());

} // end sample
```

StartConnection

このメソッドは、新規の Z and I Emulator for Windows のエミュレーター接続を開始します。ConfigParms ストリングには、使用上の注意 (ページ 50)で 説明されているとおりの接続構成情報が入っています。

プロトタイプ

void StartConnection(char *ConfigParms)

パラメーター

char *ConfigParms

NULL 文字で終了する接続構成ストリング。

戻り値

なし

使用上の注意

接続構成ストリングは、インストール・システムによって異なります。異なるインストール・システム上のホスト・アクセス・クラス・ライブラリーには、 構成ストリングに異なる形式または情報が必要とされる場合があります。この呼び出しは、当然非同期になります。 この呼び出しが戻されたとき、新しい接続はまだ開始されていないことがあります。アプリケーションは、RegisterStartEvent 関数を使用すると、 接続の開始時に通知を受けることができます。

Z and I Emulator for Windows の場合、構成ストリングの形式は次のとおりです。

```
PROFILE=[\"]<filename>[\"] [CONNNAME=<c>] [WINSTATE=<MAX|MIN|RESTORE|HIDE>]
```

オプションのパラメーターは、大括弧 [] で囲みます。パラメーターは、少なくとも 1 つのブランクで区切ります。パラメーターは、大文字、小文字、または混合のいずれでも指定可能で、 順序も任意です。各パラメーターの意味は、次のとおりです。

PROFILE=<filename>

接続構成情報の入った Z and I Emulator for Windows のワークステーション・プロファイル (.WS ファイル) の名前を指定します。このパラメーターは、オプションではありません。 プロファイル名を入力しなければなりません。ファイル名にブランクを含める場合、 その名前を二重引用符で囲まなければなりません。<filename> の値は、拡張子のないプロファイル名、.WS 拡張子の付いたプロファイル名、または完全修飾プロファイル名パスのどれでも構いません。

CONNNAME=<c>

新しい接続の接続名 (EHLLAPI 短縮セッション ID) を指定します。この値は、単一の英字 (A から Z、または a から z) でなければなりません。この値を指定しない場合、次の使用可能な接続名が自動的に割り当てられます。指定した名前の付いた接続が既に存在する場合、 エラーが出されます (ERRMAJ_INVALID_SESSION)。

WINSTATE=<MAX|MIN|RESTORE|HIDE>

エミュレーター・ウィンドウの初期状態を指定します。このパラメーターを指定しない場合のデフォルト値は、RESTORE です。

注: この呼び出しは本質的に非同期であるため、 この関数でエラーが戻されなくても接続の開始が失敗することがあります。例えば、同一名を使用して 2 つの接続を短時間のうちに開始すると、1 番目の 接続はまだ開始されていないため、2 番目の StartConnection は エラーになりません。しかし、最終的に 2 番目の接続が接続名を登録しようとしても、1 番目の 接続によってその名前は既に使われているため、開始することはできません。このような可能性を最小化するため、可能なかぎり CONNNAME パラメーターを 指定しないで接続を開始しなければなりません。

例

以下に、StartConnection メソッドの例を示します。

```
ECLConnMgr Manager;  // Connection manager object

// Start a host connection "E" and check for errors

try {
  Manager.StartConnection("profile=coax connname=e");
}
catch (ECLErr Error) {
  MessageBox(NULL, Error.GetMsgText(), "Session start error!", MB_OK);
}
```

StopConnection

このメソッドは、接続ハンドルによって識別された エミュレーター接続を停止 (終了) します。StopParms ストリングの内容の詳細については、使用上の注意 (ページ 52)を 参照してください。

プロトタイプ

void StopConnection(Long ConnHandle, char *StopParms)

パラメーター

Long ConnHandle

停止される接続のハンドル。

char * StopParms

Null 文字で終了する接続停止パラメーター・ストリング。

戻り値

なし

使用上の注意

接続停止パラメーター・ストリングは、インストール・システムによって異なります。異なるインストール・システム上のホスト・アクセス・クラス・ライブラリーには、 異なる形式または内容のパラメーター・ストリングが必要とされる場合があります。Z and I Emulator for Windows の場合、ストリングの形式は次のとおりです。

```
[SAVEPROFILE=<YES|NO|DEFAULT>]
```

オプションのパラメーターは、大括弧 [] で囲みます。パラメーターは、少なくとも 1 つのブランクで区切ります。パラメーターは、大文字、小文字、または混合のいずれでも指定可能で、 順序も任意です。SAVEPROFILE パラメーターの意味は、次のとおりです。

SAVEPROFILE=<YES|NO|DEFAULT> は、現行接続構成を元のワークステーション・プロファイル (.WS ファイル) に戻して保管するかどうかを制御します。その場合、接続の間に加えたすべての構成変更を使用してプロファイルが更新されることになります。NO を指定した場合、接続が停止されるときプロファイルは 更新されません。YES を指定した場合、接続が停止されるとき現行構成 (変更されている ことがある) を使用してプロファイルは更新されます。DEFAULT を指定した場合には、更新オプションは**「ファイル」->「終了時に変更を保管」**エミュレーター・メニュー・オプションによって制御されます。このパラメーターを指定しない場合には、DEFAULT が使用されます。

例

```
//-------------------------------------------------------------------
// ECLConnMgr::StopConnection
//
// Stop the first connection in the connection list.
//-------------------------------------------------------------------
void Sample29() {

ECLConnMgr   CM; // Connection manager object

if (CM.GetConnList()->GetCount() > 0) {

  printf("Stopping connection %c.\n",
         CM.GetConnList()->GetFirstConnection()->GetName());

  CM.StopConnection(
    CM.GetConnList()->GetFirstConnection()->GetHandle(),
    "saveprofile=no");
}
else printf("No connections to stop.\n");


} // end sample
```

RegisterStartEvent

このメソッドは、 すべての接続の開始および停止のイベントの通知を受け取るための アプリケーション・オブジェクトを登録します。アプリケーションでこの関数を使用するには、 ECLStartNotify クラスから派生したオブジェクトを作成しなければなりません。作成すると、そのオブジェクトを指すポインターは この登録関数に渡されま

す。実装上の制約事項: アプリケーションは、通信の開始または停止の通知につき 1 つのオブジェクトしか登録できません。

この関数を使用して通知オブジェクトが登録されると、その後、Z and I Emulator for Windows の接続が開始または停止されるたびにこのオブジェクトが呼び出されます。このオブジェクトは、すべての接続に関して、 それが StartConnection 関数で開始されるか、 または明示的に開始されるたびに通知を受け取ります。このイベントを、ホスト・システムとの接続の接続時または切断時に 起動される通信のスタート・ストップ・イベントと混同してはなりません。

詳しくは、ECLStartNotify クラス (ページ 191)を参照してください。

プロトタイプ

void RegisterStartEvent(ECLStartNotify *NotifyObject)

パラメーター

ECLStartNotify *NotifyObject

ECLStartNotify クラスから派生したオブジェクトを指すポインター。

戻り値

なし

例

```
//-------------------------------------------------------------------
// ECLConnMgr::RegisterStartEvent
//
// See ECLStartNotify クラス (ページ 191) for example of this method.
//-------------------------------------------------------------
```

UnregisterStartEvent

このメソッドは、 以前に RegisterStartEvent 関数を使用して接続開始または停止イベント用に 登録されているアプリケーション・オブジェクトの登録を抹消します。登録済みのアプリケーション通知オブジェクトの場合、 先にこの関数を呼び出してその登録を抹消しないかぎり、オブジェクトを破棄してはなりません。現在登録されている通知オブジェクトがない場合や、 登録済みオブジェクトが渡された NotifyObject でない場合、 この関数は何も実行しません (エラーになりません)。

通知オブジェクトの登録を抹消するときは、 その NotifyStop メソッドが呼び出されます。

詳しくは、ECLStartNotify クラス (ページ 191)を参照してください。

プロトタイプ

void UnregisterStartEvent(ECLStartNotify *NotifyObject)

パラメーター

なし

戻り値

なし

例

```
//----------------------------------------------------------------------
// ECLConnMgr::UnregisterStartEvent
//
// See ECLStartNotify クラス (ページ 191) for example of this method.
//----------------------------------------------------------------------
```

ECLCommNotify クラス

ECLCommNotify は、抽象基本クラスです。アプリケーションは、このクラスの インスタンスを直接作成することはできません。アプリケーションでこのクラスを使用するには、ECLCommNotify から派生した 独自のクラスを定義しなければなりません。アプリケーションは、 その派生クラス内に NotifyEvent() メンバー関数を実装しなければなりません。また、 オプションで NotifyError() および NotifyStop() メンバー関数を実装することも できます。

アプリケーションが ZIEWin 接続上の通信の接続/切断イベントの通知を受けられるようにするには、ECLCommNotify クラスを使用します。接続/切断イベントは、ホスト・システムに対して ZIEWin 接続 (ウィンドウ) が接続または切断されるたびに生成されます。

アプリケーションが通信の接続/切断イベントの通知を受けるには、 次に示すステップを実行しなければなりません。

1. ECLCommNotify から派生したクラスを定義します。
2. その派生クラスを採用し、NotifyEvent() メンバー関数を実装します。
3. オプションで、NotifyError() または NotifyStop() 関数 (あるいはその両方) を実装します。
4. 派生クラスのインスタンスを作成します。
5. そのインスタンスを ECLConnection::RegisterCommEvent() 関数で登録します。

ここに示された例は、それがどのように行われるかを例示しています。上記のステップを完了すると、その後、ホストへの接続の通信リンクが 接続されるかまたは切断されると、そのたびに アプリケーション NotifyEvent() メンバー関数が呼び出されます。

イベントの生成時にエラーが検出された場合、ECLErr オブジェクトを 使用して NotifyError() メンバー関数が呼び出されます。エラーの特性に応じて、 エラー後にイベントが続けて生成されるかどうかが決まります。イベント生成が終了したとき (エラーか、ECLConnection::UnregisterCommEvent の呼び出しか、 または ECLConnection オブジェクトの破棄のいずれかが原因で) には、 NotifyStop() メンバー関数が呼び出されます。イベント通知が終了す

るときには、NotifyStop() メンバー関数が常に呼び出され、 アプリケーション・オブジェクトの登録が抹消されます。

アプリケーションが NotifyError() メンバー関数の実装を行わない場合、 デフォルトの実装が使われます (単純なメッセージ・ボックスがユーザーに対して表示されます)。アプリケーションがデフォルトの振る舞いをオーバーライドするには、 アプリケーションの派生クラス内に NotifyError() 関数を実装します。同様に、アプリケーションがこの関数を提供しない場合、 デフォルトの NotifyStop() 関数が使われます (デフォルトの振る舞いでは何も行われません)。

またアプリケーションは、派生したクラス用に自身のコンストラクターおよび デストラクターを任意で提供できることに注意してください。これが便利なのは、 アプリケーションが特定のインスタンス別データをそのクラス内に保管してから、 その情報をコンストラクター上のパラメーターとして渡したい場合です。例えば、アプリケーションにおいて、通信イベントが発生したら アプリケーション・ウィンドウにメッセージをポストしたい場合があります。アプリケーションは、ウィンドウ・ハンドルをグローバル変数として定義する (このハンドルを、NotifyEvent() 関数に見えるようにするため) 代わりに、 ウィンドウ・ハンドルを受け取ってクラス・メンバーのデータ域に保管する クラス用のコンストラクターとして定義することができます。

アプリケーションは、イベントを受け取るために通知オブジェクトを 登録しているかぎり、そのオブジェクトを破棄してはなりません。

実装上の制約事項: 現在、ECLConnection オブジェクトでは、通信イベントの通知用に 1 つしか通知オブジェクトを登録できません。その ECLConnection オブジェクト用に通知オブジェクトが既に登録されて いる場合、ECLConnection::RegisterCommEvent からエラーがスローされます。

派生

ECLBase > ECLNotify > ECLCommNotify

例

```
//---------------------------------------------------------------------
// ECLCommNotify class
//
// This sample demonstrates the use of:
//
// ECLCommNotify::NotifyEvent
// ECLCommNotify::NotifyError
// ECLCommNotify::NotifyStop
// ECLConnection::RegisterCommEvent
// ECLConnection::UnregisterCommEvent
//---------------------------------------------------------------------


 //.....................................................................
// Define a class derived from ECLCommNotify
//.....................................................................
class MyCommNotify: public ECLCommNotify
{
public:
  // Define my own constructor to store instance data
```

```
  MyCommNotify(HANDLE DataHandle);

  // We have to implement this function
  void NotifyEvent(ECLConnection *ConnObj, BOOL Connected);

  // We choose to implement this function
  void NotifyStop (ECLConnection *ConnObj, int Reason);

  // We will take the default behaviour for this so we
  // don't implement it in our class:
  // void NotifyError (ECLConnection *ConnObj, ECLErr ErrObject);

private:
  // We will store our application data handle here
  HANDLE MyDataH;
};
```

```
//.....................................................................
void MyCommNotify::NotifyEvent(ECLConnection *ConnObj,
                           BOOL Connected)
//
// This function is called whenever the communications link
// with the host connects or disconnects.
//
// For this example, we will just write a message.  Note that we
// have access the the MyDataH handle which could have application
// instance data if we needed it here.
//
// The ConnObj pointer is to the ECLConnection object upon which
// this event was registered.
//.....................................................................
{
  if (Connected)
    printf("Connection %c is now connected.\n", ConnObj->GetName());
  else
    printf("Connection %c is now disconnected.\n", ConnObj->GetName());

  return;
}

 //.....................................................................
MyCommNotify::MyCommNotify(HANDLE DataHandle)   // Constructor
//.....................................................................
{
  MyDataH = DataHandle;  // Save data handle for later use
}

//.....................................................................
void MyCommNotify::NotifyStop(ECLConnection *ConnObj,
                          int Reason)
//.....................................................................
{
  // When notification ends, display message
  printf("Comm link monitoring for %c stopped.\n", ConnObj->GetName());
}

 //.....................................................................
```

```
// Create the class and start notification on connection 'A'.
//.............................................................
void Sample30() {

ECLConnection *Conn;    // Ptr to connection object
MyCommNotify *Event;    // Ptr to my event handling object
HANDLE InstData;      // Handle to application data block (for example)

try {
  Conn = new ECLConnection('A');         // Create connection obj
  Event = new MyCommNotify(InstData);    // Create event handler

  Conn->RegisterCommEvent(Event);        // Register for comm events

  // At this point, any comm link event will cause the
  // MyCommEvent::NotifyEvent() function to execute.  For
  // this sample, we put this thread to sleep during this
  // time.

  printf("Monitoring comm link on 'A' for 60 seconds...\n");
  Sleep(60000);
```

```
  // Now stop event generation.  This will cause the NotifyStop
  // member to be called.
  Conn->UnregisterCommEvent(Event);

  delete Event;  // Don't delete until after unregister!
  delete Conn;
}
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}

} // end sample
```

ECLCommNotify メソッド

以下のセクションでは、ECLCommNotify クラスにおいて有効なメソッドについて説明します。

ECLCommNotify() ~ECLCommNotify() virtual void NotifyEvent (ECLConnection *ConnObj, BOOL Connected) = 0 virtual void NotifyError (ECLConnection *ConnObj, ECLErr ErrObject) virtual void NotifyStop (ECLConnection *ConnObj, int Reason)

NotifyEvent

このメソッドは、純粋仮想メンバー関数です (アプリケーションは ECLCommNotify から派生したクラス内にこの関数を実装しなければなりません)。*must* 接続の開始または停止のときと、スタート・ストップ・イベントのためにオブジェクトが登録されたときは、 常にこの関数が呼び出されます。通信リンクが接続されている場合は Connected BOOL は True、 ホストに接続されていない場合は False になります。

プロトタイプ

virtual void NotifyEvent (ECLConnection *ConnObj, BOOL Connected)

パラメーター

ECLConnection *ConnObj

これは、イベントが発生した ECLConnection オブジェクトを指すポインターです。

BOOL Connected

通信リンクが接続されている場合は True、切断されている場合は False です。

戻り値

なし

NotifyError

ECLConnection オブジェクトがイベントの生成時にエラーを検出する たびにこのメソッドが呼び出されます。エラー・オブジェクトには、そのエラーについての情報が 含まれます (ECLErr クラス (ページ 59)を参照)。エラーの特性に応じて、エラー後にイベントが続けて生成されることがあります。エラーが原因で イベント生成が停止した場合、NotifyStop() 関数が呼び出されます。アプリケーションは、この関数を採用するか、 または ECLCommNotify 基本クラスにエラーを処理させるかを選ぶことができます。基本クラスは、ECLErr::GetMsgText() 関数から提供される テキストを使用して、メッセージ・ボックスにエラーを表示します。アプリケーションが、 その派生クラス内にこの関数を実装すると、それによって 基本クラス関数がオーバーライドされます。

プロトタイプ

virtual void NotifyError (ECLConnection *ConnObj, ECLErr ErrObject)

パラメーター

ECLConnection *ConnObj

これは、エラーが発生した ECLConnection オブジェクトを指すポインターです。

ECLErr ErrObject

これは、エラーを記述した ECLErr オブジェクトです。

戻り値

なし

NotifyStop

イベント生成が何らかの理由 (例えば、エラー条件が原因か、または ECLConnection::UnregisterCommEvent の呼び出しなどが原因) で停止すると、このメソッドが呼び出されます。

実装上の注意: 現在、理由コードは未使用であるため、ゼロになります。

プロトタイプ

virtual void NotifyStop (ECLConnection *ConnObj, int Reason)

パラメーター

ECLConnection *ConnObj

これは、通知を停止する ECLConnection オブジェクトを指すポインターです。

int Reason

これは未使用 (ゼロ) です。

戻り値

なし

ECLErr クラス

ECLErr クラスは、ホスト・アクセス・クラス・ライブラリー・クラスから実行時エラー情報を戻すメソッドを提供します。エラー状態になると、ECLErr オブジェクトが作成され、 エラー情報と診断情報がその中に入れられます。そうすると、ECLErr オブジェクトが C++ 例外として送出されます。次に、 その catch された ECLErr オブジェクトからエラー情報と診断情報を照会できます。

アプリケーションは、ECLErr オブジェクトを作成したり、直接 throw したりしてはなりません。

派生

ECLBase > ECLErr

ECLErr メソッド

以下のセクションでは、ECLErr クラスにおいて有効な メソッドについて説明します。

const int GetMsgNumber() const int GetReasonCode() const char *GetMsgText()

GetMsgNumber

このメソッドは、この ECLErr オブジェクトが 作成されたときに設定されたメッセージ番号を戻します。エラー・メッセージ番号については、ERRORIDS.HPP に説明があります。

プロトタイプ

const int GetMsgNumber()

パラメーター

なし

戻り値

const int

エラー・メッセージ番号。

例

```
//---------------------------------------------------------------------
// ECLErr::GetMsgNumber
//
// Cause an 'invalid parameters' error and tryp the ECL exception.
// The extract the error number and language-sensative text.
//---------------------------------------------------------------------
void Sample31() {

ECLPS   *PS = NULL;

try {
  PS = new ECLPS('A');
  PS->SetCursorPos(999,999);  // Invalid parameters
}
catch (ECLErr ErrObj) {
  printf("The following ECL error was trapped:\n");
  printf("%s \nError number: %lu\nReason code: %lu\n",
    ErrObj.GetMsgText(),
    ErrObj.GetMsgNumber(),
    ErrObj.GetReasonCode());
}

if (PS != NULL)
  delete PS;

} // end sample
```

GetReasonCode

このメソッドは、ECLErr オブジェクトから 理由コード (場合によっては、2 次またはマイナー戻りコードと呼ばれます) を取得します。一般に、このコードは、デバッグおよび診断を目的として使われます。これは、ホスト・アクセス・クラス・ライブラリーの今後のバージョンでは 変わることがあるので、 プログラマチックに使用しないでください。理由コードの説明は、ERRORIDS.HPP にあります。

プロトタイプ

const int GetReasonCode()

パラメーター

なし

戻り値

const int

ECLErr 理由コード。

例

```
//-------------------------------------------------------------------
// ECLErr::GetReasonCode
//
// Cause an 'invalid parameters' error and tryp the ECL exception.
// The extract the error number and language-sensative text.
//-------------------------------------------------------------------
void Sample32() {

ECLPS   *PS = NULL;

try {
  PS = new ECLPS('A');
  PS->SetCursorPos(999,999);  // Invalid parameters
}
catch (ECLErr ErrObj) {
  printf("The following ECL error was trapped:\n");
  printf("%s \nError number: %lu\nReason code: %lu\n",
    ErrObj.GetMsgText(),
    ErrObj.GetMsgNumber(),
    ErrObj.GetReasonCode());
}

if (PS != NULL)
  delete PS;

} // end sample
```

GetMsgText

このメソッドは、この ECLErr オブジェクトを 作成するのに使われたエラー・コードに関連したメッセージ・テキストを戻します。このメッセージ・テキストは、現在インストールされている Z and I Emulator for Windows の言語で戻されます。

注: ECLErr オブジェクトを削除すると、戻されたポインターは無効になります。

プロトタイプ

const char *GetMsgText()

パラメーター

なし

戻り値

char *

この ECLErr オブジェクトの一部を成すエラー・コードと 関連したメッセージ・テキスト。

例

```
//-------------------------------------------------------------------
// ECLErr::GetMsgText
//
// Cause an 'invalid parameters' error and tryp the ECL exception.
// The extract the error number and language-sensative text.
//-------------------------------------------------------------------
void Sample33() {

ECLPS   *PS = NULL;

try {
  PS = new ECLPS('A');
  PS->SetCursorPos(999,999);  // Invalid parameters
}
catch (ECLErr ErrObj) {
  printf("The following ECL error was trapped:\n");
  printf("%s \nError number: %lu\nReason code: %lu\n",
    ErrObj.GetMsgText(),
    ErrObj.GetMsgNumber(),
    ErrObj.GetReasonCode());
}

if (PS != NULL)
  delete PS;

} // end sample
```

使用上の注意

メッセージ・テキストは、Z and I Emulator for Windows のメッセージ機能から取り出されます。

ECLField クラス

ECLField には、ECLPS オブジェクトに収容される ECLFieldList オブジェクト内の特定のフィールドの情報が入っています。アプリケーションで、 直接このタイプのオブジェクトを作成しないようにしてください。ECLField オブジェクト は、ECLFieldList オブジェクトによって間接的に作成されます。

ECLField オブジェクトは、ホストの表示スペースの 単一のフィールドを記述します。これには、フィールドのさまざまな属性を照会したり、フィールドのテキストを更新したり する (例えば、フィールド・テキストを修正する) ためのメソッドが備わっています。フィールド属性は、変更できません。

派生

ECLBase > ECLField

コピー・コンストラクターおよび代入演算子

このオブジェクトは、コピー構築および割り当てをサポートします。このことは、 後で処理するためにホスト画面でフィールドを簡単に確保したいアプリケーションには 役に立ちます。テキスト・バッファーを割り振ったりフィールドのストリングの内容を コピーするよりも、アプリケーションは専用 ECLField オブジェクトにフィールドを 単純に保管することができます。保管されたコピーには、フィールドのテキスト値、属性、開始位置、長さなどを含む ECLField オブジェクトのすべての機能が保存されます。例えば、画面の最初の入力フィールドを確保するのに必要なアプリケーションを考えてみてください。表 1 : コピー構築および割り当ての例 (ページ 63) は、これが達成できる 2 つの方法を示しています。

表 1. コピー構築および割り当ての例

ストリングとしてフィールドを保管

```
#include "eclall.hpp"

{
   char *SavePtr;  // Ptr to saved string
   ECLPS Ps('A'); // PS object
   ECLFieldList *List;
   ECLField       *Fld;

   // Get fld list and rebuild it
   List = Ps->GetFieldList();
   List->Refresh();

   // See if there is an input field
   Fld = List->GetFirstField(GetUnmodified);
   if  (Fld !=NULL) {
      // Copy the field's text value
```

ECLField オブジェクトとしてフィールドを保管

```
#include "eclall.hpp"

{
   ECLField SaveFld;  // Saved field
   ECLPS Ps('A');      // PS object
   ECLFieldList *List;
   ECLField       *Fld;

   // Get fld list and rebuild it
   List = Ps->GetFieldList();
   List->Refresh();

   // See if there is an input field
   Fld = List->GetFirstField(GetUnmodified);
   if  (Fld !=NULL) {
      // Copy the field object
```

表 1. コピー構築および割り当ての例 (続く)

ストリングとしてフィールドを保管	ECLField オブジェクトとしてフィールドを保管

`SavePtr=malloc(Fld->Length() + 1);`
`Fld->GetScreen(SavePtr, Fld->Length()+1);`
`}`
 	
`// We now have captured the field text`
	`SaveFld = *Fld;`
	`}`
	
	`// We now have captured the field text`
	`// including text, position, attrib`

フィールドを保管するのに、ストリングの代わりに ECLField オブジェクトを使用すると 次のようないくつかの利点があります。

- ECLField オブジェクトは、フィールドのテキスト・バッファーのストレージ管理を すべて行います。アプリケーションは、テキスト・バッファーを割り振ったり、解放したり、 または、必要なバッファー・サイズの計算をする必要はありません。
- 保管されたフィールドには、属性および開始位置を含むオリジナル・フィールドの特性の すべてが保存されています。通常の ECLField メンバーの機能のすべては、SetText() 以外の 保管されたフィールドで使用することができます。保管されたフィールドはオリジナルのコピーであり、その値はホスト画面の変更、あるいは ECLFieldList::Refresh() 関数が呼び出された場合にも更新されないことに注意してください。結果として、このフィールドは 保管することが可能であり、後でアプリケーションで使用することができます。

代入演算子のオーバーライドは、 文字ストリングおよび Long integer 値型にも提供されます。このオーバーライドによって、 新しいストリングまたは数値を無保護フィールドに割り当てやすくなります。例えば、 次の例は、画面の最初の 2 つの入力フィールドを設定します。

```
ECLField *Fld1; //Ptr to 1st unprotected field in field list
ECLField *Fld2; // PTR to 2nd unprotected field in field list

Fld1 = FieldList->GetFirstField(GetUnprotected);
Fld2 = FieldList->GetNextField(Fld1, GetUnprotected);
if ((Fld1 == NULL) || (Fld2 == NULL)) return;

*Fld1 = "Easy string assignment";
*Fld2 = 1087;
```

Notes:

1. コピー構築または割り当てにより初期化された ECLField オブジェクトは、 オリジナル・フィールド・オブジェクトの読み取り専用コピーです。SetText() メソッドは、 このようなオブジェクトには無効であり、ECLErr 例外が発生する原因となります。これらのオブジェクトはコピーであるため、オリジナル・フィールド・オブジェクトが 更新あるいは削除されたときに、更新あるいは削除されることはありません。アプリケーションは、フィールド・オブジェクトが不必要になった場合に、これらのオブジェクトのコピーを削除しなければなりません。
2. 初期設定されていない ECLField オブジェクトでメソッドを呼び出すと、 未定義の結果が戻されます。
3. アプリケーションにより作成された ECLField オブジェクトは、何度でも 再割り当てすることができます。

4. 別の ECLField オブジェクト、文字ストリング、 または Long integer 値からのみ、割り当てを行うことが できます。ECLField オブジェクトに対して、他のデータ型を割り当てるのは無効です。
5. 現在 ECLFieldList の一部である ECLField オブジェクトに割り当てを行っても、 フィールドのテキスト値を更新するだけです。これは、フィールド・オブジェクトが 無保護フィールドである場合にのみ許可されます。例えば、次の例では、最初の 入力フィールドから値をコピーすることにより、画面の 2 番目の入力フィールドを変更します。

```
ECLField *Fld1;   // Ptr to 1st unprotected field in field list
ECLField *Fld2;   // Ptr to 2nd unprotected field in field list

Fld1 = FieldList->GetFirstField(GetUnprotected);
Fld2 = FieldList->GetNextField(Fld1, GetUnprotected);
if ((Fld1 == NULL) || (Fld2 == NULL)) return;

// Update the 2nd input field using text from the first
FLD2 = * Fld1;
```

   Fld2 は ECLFieldList の一部であるため、上記の割り当ては以下の例と同じです。

```
{ char temp[Fld1->GetLength()+1];
  Fld1->GetText(temp, Fld1->GetLength()+1);
  Fld2->SetText(temp);
  delete []temp;
}
```

   Fld2 が保護されている場合は、ECLErr 例外がスローされることに注意してください。また、Fld2 のテキストだけが更新され、その属性、位置、または長さは更新されない ことにも注意してください。
6. ストリングをフィールド・オブジェクトに割り当てることは、 SetText() メソッドを呼び出すことと同じです。数値は、ストリングへの最初の変換なしに、割り当てることができます。

```
*Field = 1087;
```

   これは、数をストリングに変換した後で SetText() メソッドを呼び出すことと同じです。

ECLField メソッド

以下のセクションでは、ECLField クラスにおいて有効な メソッドについて説明します。

ULONG GetStart() void GetStart(ULONG *RowULONG *Col) ULONG GetStartRow() ULONG GetStartCol() ULONG GetEnd() void GetEnd(ULONG *RowULONG *Col) ULONG GetEndRow() ULONG GetEndCol() ULONG GetLength() ULONG GetScreen(char *Buff, ULONG BuffLen, PS_PLANE Plane = TextPlane) void SetText(char *text) BOOL IsModified() BOOL IsProtected() BOOL IsNumeric() BOOL IsHighIntensity() BOOL IsPenDetectable() BOOL IsDisplay() unsigned charGetAttribute()

次のメソッドは、ECLField クラスで有効です。

ULONG GetScreen(WCHAR *Buff, ULONG BuffLen, PS_PLANE Plane = TextPlane) void SetText(WCHAR *text)

GetStart

このメソッドは、 フィールドの最初の文字の表示スペース内の位置を 戻します。GetStart メソッドには、2 種類のシグニチャーがあります。ULONG GetStart は、表示スペースの左上隅を「1」として、線形値で位置を戻します。void GetStart(ULONG *Row, ULONG *Col) は、行および桁の座標で位置を戻します。

プロトタイプ

ULONG GetStart(),

void GetStart(ULONG *Row, ULONG *Col)

パラメーター

ULONG *Row

この出力パラメーターは、更新される行値を指すポインターです。

ULONG *Col

この出力パラメーターは、更新される桁値を指すポインターです。

戻り値

ULONG

線形配列として示される表示スペース内の位置。

例

以下の例は、フィールドの最初の文字の表示スペース内の位置を どのように戻すかを示します。

```
/--------------------------------------------------------------------
// ECLField::GetStart
//
// Iterate over list of fields and print each field
// starting pos, row, col, and ending pos, row, col.
//--------------------------------------------------------------------
void Sample34() {

ECLPS        *pPS;           // Pointer to PS object
ECLFieldList *pFieldList;    // Pointer to field list object
ECLField     *pField;        // Pointer to field object

try {
  pPS = new ECLPS('A');               // Create PS object for 'A'

  pFieldList = pPS->GetFieldList();    // Get pointer to field list
  pFieldList->Refresh();               // Build the field list

  printf("Start(Pos,Row,Col)  End(Pos,Row,Col) Length(Len)\n");
```

```
  for (pField = pFieldList->GetFirstField();      // First field
    pField != NULL;                               // While more
    pField = pFieldList->GetNextField(pField)) {  // Next field

    printf("Start(%04lu,%04lu,%04lu)  End(%04lu,%03lu,%04lu)
     Length(%04lu)\n",
      pField->GetStart(), pField->GetStartRow(),
      pField->GetStartCol(),
      pField->GetEnd(), pField->GetEndRow(),
      pField->GetEndCol(), pField->GetLength());
  }
  delete pPS;
}
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}

} // end sample
```

GetStartRow

このメソッドは、ECLPS オブジェクトに関連した 接続での、ECLFieldList 集合内の指定フィールドの開始行位置を戻します。

プロトタイプ

ULONG GetStartRow()

パラメーター

なし

戻り値

ULONG

これは、所定のフィールドの開始行です。

例

```
/*------------------------------------------------------------------
// ECLField::GetStartRow
//
// Iterate over list of fields and print each field
// starting pos, row, col, and ending pos, row, col.
//------------------------------------------------------------------
void Sample34() {

ECLPS        *pPS;           // Pointer to PS object
ECLFieldList *pFieldList;    // Pointer to field list object
ECLField     *pField;        // Pointer to field object
```

```
try {
  pPS = new ECLPS('A');                // Create PS object for 'A'

  pFieldList = pPS->GetFieldList();    // Get pointer to field list
  pFieldList->Refresh();               // Build the field list

  printf("Start(Pos,Row,Col)  End(Pos,Row,Col) Length(Len)\n");
  for (pField = pFieldList->GetFirstField();      // First field
    pField != NULL;                               // While more
    pField = pFieldList->GetNextField(pField)) {  // Next field

    printf("Start(%04lu,%04lu,%04lu)  End(%04lu,%03lu,%04lu)  Length(%04lu)\n",
      pField->GetStart(), pField->GetStartRow(), pField->GetStartCol(),
      pField->GetEnd(), pField->GetEndRow(),
      pField->GetEndCol(), pField->GetLength());
  }
  delete pPS;
}
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}

} // end sample
```

GetStartCol

このメソッドは、ECLPS オブジェクトに関連した 接続での、ECLFieldList 集合内の指定フィールドの開始桁位置を戻します。

プロトタイプ

ULONG GetStartCol()

パラメーター

なし

戻り値

ULONG

これは、所定のフィールドの開始桁です。

例

```
/-------------------------------------------------------------------
// ECLField::GetStartCol
//
// Iterate over list of fields and print each field
// starting pos, row, col, and ending pos, row, col.
//-------------------------------------------------------------------
void Sample34() {
```

```
ECLPS        *pPS;           // Pointer to PS object
ECLFieldList *pFieldList;    // Pointer to field list object
ECLField     *pField;        // Pointer to field object

try {
  pPS = new ECLPS('A');               // Create PS object for 'A'

  pFieldList = pPS->GetFieldList();    // Get pointer to field list
  pFieldList->Refresh();               // Build the field list

  printf("Start(Pos,Row,Col)  End(Pos,Row,Col) Length(Len)\n");
  for (pField = pFieldList->GetFirstField();      // First field
    pField != NULL;                               // While more
    pField = pFieldList->GetNextField(pField)) {  // Next field

    printf("Start(%04lu,%04lu,%04lu)  End(%04lu,%03lu,%04lu)
      Length(%04lu)\n",
      pField->GetStart(), pField->GetStartRow(),
      pField->GetStartCol(),
      pField->GetEnd(), pField->GetEndRow(),
      pField->GetEndCol(), pField->GetLength());
  }
  delete pPS;
}
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}

} // end sample
```

GetEnd

このメソッドは、 フィールドの最後の文字の表示スペース内の位置を戻します。GetEnd メソッドには、2 種類のシグニチャーがあります。ULONG GetEnd は、表示スペースの左上隅を「1」として、線形値で位置を戻します。void GetEnd(ULONG *Row, ULONG *Col) は、行および桁の座標で位置を戻します。

プロトタイプ

ULONG GetEnd()

void GetEnd(ULONG *Row, ULONG *Col)

パラメーター

ULONG *Row

この出力パラメーターは、更新される行値を指すポインターです。

ULONG *Col

この出力パラメーターは、更新される桁値を指すポインターです。

戻り値

ULONG

線形配列として示される表示スペース内の位置。

例

以下の例は、フィールドの最後の文字の表示スペース内の位置を どのように戻すかを示します。

```
/-------------------------------------------------------------------
// ECLField::GetEnd
//
// Iterate over list of fields and print each field
// starting pos, row, col, and ending pos, row, col.
//-------------------------------------------------------------------
void Sample34() {

ECLPS        *pPS;           // Pointer to PS object
ECLFieldList *pFieldList;    // Pointer to field list object
ECLField     *pField;        // Pointer to field object

try {
  pPS = new ECLPS('A');               // Create PS object for 'A'

  pFieldList = pPS->GetFieldList();    // Get pointer to field list
  pFieldList->Refresh();               // Build the field list

  printf("Start(Pos,Row,Col)  End(Pos,Row,Col) Length(Len)\n");
  for (pField = pFieldList->GetFirstField();      // First field
    pField != NULL;                               // While more
    pField = pFieldList->GetNextField(pField)) {  // Next field

    printf("Start(%04lu,%04lu,%04lu)  End(%04lu,%03lu,%04lu)
     Length(%04lu)\n",
      pField->GetStart(), pField->GetStartRow(),
      pField->GetStartCol(),
      pField->GetEnd(), pField->GetEndRow(),
      pField->GetEndCol(), pField->GetLength());
  }
  delete pPS;
}
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}

} // end sample
```

GetEndRow

このメソッドは、フィールドの終了行位置を戻します。

プロトタイプ

ULONG GetEndRow()

パラメーター

なし

戻り値

ULONG

これは、指定されたフィールドの終了行です。

例

```
/--------------------------------------------------------------------
// ECLField::GetEndRow
//
// Iterate over list of fields and print each field
// starting pos, row, col, and ending pos, row, col.
//--------------------------------------------------------------------
void Sample34() {

ECLPS        *pPS;           // Pointer to PS object
ECLFieldList *pFieldList;    // Pointer to field list object
ECLField     *pField;        // Pointer to field object

try {
  pPS = new ECLPS('A');               // Create PS object for 'A'

  pFieldList = pPS->GetFieldList();    // Get pointer to field list
  pFieldList->Refresh();               // Build the field list

  printf("Start(Pos,Row,Col)  End(Pos,Row,Col) Length(Len)\n");
  for (pField = pFieldList->GetFirstField();      // First field
    pField != NULL;                               // While more
    pField = pFieldList->GetNextField(pField)) {  // Next field

    printf("Start(%04lu,%04lu,%04lu)  End(%04lu,%03lu,%04lu)
     Length(%04lu)\n",
      pField->GetStart(), pField->GetStartRow(),
      pField->GetStartCol(),
      pField->GetEnd(), pField->GetEndRow(),
      pField->GetEndCol(), pField->GetLength());
  }
  delete pPS;
}
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}

} // end sample
```

GetEndCol

このメソッドは、フィールドの終了桁位置を戻します。

プロトタイプ

ULONG GetEndCol()

パラメーター

なし

戻り値

ULONG

これは、指定されたフィールドの終了行です。

例

```
/------------------------------------------------------------------
// ECLField::GetEndCol
//
// Iterate over list of fields and print each field
// starting pos, row, col, and ending pos, row, col.
//------------------------------------------------------------------
void Sample34() {

ECLPS        *pPS;           // Pointer to PS object
ECLFieldList *pFieldList;    // Pointer to field list object
ECLField     *pField;        // Pointer to field object

try {
  pPS = new ECLPS('A');               // Create PS object for 'A'

  pFieldList = pPS->GetFieldList();    // Get pointer to field list
  pFieldList->Refresh();               // Build the field list

  printf("Start(Pos,Row,Col)  End(Pos,Row,Col) Length(Len)\n");
  for (pField = pFieldList->GetFirstField();      // First field
    pField != NULL;                               // While more
    pField = pFieldList->GetNextField(pField)) {  // Next field

    printf("Start(%04lu,%04lu,%04lu)  End(%04lu,%03lu,%04lu)
     Length(%04lu)\n",
      pField->GetStart(), pField->GetStartRow(),
      pField->GetStartCol(),
      pField->GetEnd(), pField->GetEndRow(),
      pField->GetEndCol(), pField->GetLength());
  }
  delete pPS;
}
catch (ECLErr Err) {
```

```
  printf("ECL Error: %s\n", Err.GetMsgText());
}

} // end sample
```

GetLength

このメソッドは、フィールドの長さを戻します。長さは、表示スペースの複数行にまたがっていても、 フィールド全体を含みます。それには、フィールドの先頭にあるフィールド属性文字は含まれません。

プロトタイプ

ULONG GetLength()

パラメーター

なし

戻り値

ULONG

フィールドの長さ。

例

以下の例は、フィールドの長さをどのように戻すかを示します。

```
/-------------------------------------------------------------------
// ECLField::GetLength
//
// Iterate over list of fields and print each field
// starting pos, row, col, and ending pos, row, col.
//-------------------------------------------------------------------
void Sample34() {

ECLPS        *pPS;           // Pointer to PS object
ECLFieldList *pFieldList;    // Pointer to field list object
ECLField     *pField;        // Pointer to field object

try {
  pPS = new ECLPS('A');               // Create PS object for 'A'

  pFieldList = pPS->GetFieldList();    // Get pointer to field list
  pFieldList->Refresh();               // Build the field list

  printf("Start(Pos,Row,Col)  End(Pos,Row,Col) Length(Len)\n");
  for (pField = pFieldList->GetFirstField();      // First field
    pField != NULL;                               // While more
    pField = pFieldList->GetNextField(pField)) {  // Next field

    printf("Start(%04lu,%04lu,%04lu)  End(%04lu,%03lu,%04lu)  Length(%04lu)\n",
```

```
      pField->GetStart(), pField->GetStartRow(), pField->GetStartCol(),
      pField->GetEnd(), pField->GetEndRow(),
      pField->GetEndCol(), pField->GetLength());
  }
  delete pPS;
}
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}

} // end sample
```

GetScreen

GetScreen メソッドは、 アプリケーション提供のバッファーにフィールドからのデータを入れます。バッファーにコピーされるデータの型は、 オプションの Plane パラメーターを使用して選択します。デフォルトでは、テキスト・プレーン・データが戻されます。戻されるデータは、このフィールド・オブジェクトの作成時に存在していたままの フィールドです。これは、ECLFieldList::Refresh 関数の呼び出し以降に更新されている場合は 、フィールドの現在の内容を反映していません。

戻されたデータの長さが、フィールドの長さになります (GetLength (ページ 73)を参照してください)。TextPlane を コピーすると、最後のデータ型の後にヌル終了バイトが付加されます。したがって、アプリケーションは、テキスト・プレーンを取得するとき、 フィールド長より最低限 1 バイト大きいバッファーを用意しなければなりません。アプリケーション・バッファーが小さすぎると、戻されたデータは切り捨てられます。アプリケーション・バッファーにコピーされたバイト数が、関数の結果として 戻されます (テキスト・プレーンのヌル終了文字は含まれません)。

FieldPlane は、この関数で取得できません。ECLField::GetAttribute を使用して、 フィールド属性値を取得できます。

プロトタイプ

```
ULONG GetScreen(char *Buff, ULONG BuffLen, PS_PLANE Plane=TextPlane)
```

パラメーター

char * Buff

フィールド・データが入るアプリケーション・バッファーを指すポインター。

ULONG BuffLen

アプリケーション・バッファーの長さ。

PS_PLANE Plane

オプション・パラメーター。フィールド・データのどのプレーンを検索するかについての指示の 列挙。TextPlane、ColorPlane、または ExtendedFieldPlane のいずれかでなければなりません。

戻り値

ULONG

アプリケーション・バッファーにコピーされる バイト数。TextPlane データの後続ヌル文字は含みません。

例

以下の例は、Plane パラメーターによって指示されたフィールド・データを指す ポインターをどのように戻すかを示します。

```
/-------------------------------------------------------------------
// ECLField::GetScreen
//
// Iterate over list of fields and print each fields text contents.
//-------------------------------------------------------------------
void Sample35() {

ECLPS        *PS;           // Pointer to PS object
ECLFieldList *FieldList;    // Pointer to field list object
ECLField     *Field;        // Pointer to field object
char         *Buff;         // Screen data buffer
ULONG        BuffLen;

try {
  PS = new ECLPS('A');               // Create PS object for 'A'

  BuffLen = PS->GetSize() + 1;       // Make big enough for entire screen
  Buff = new char[BuffLen];          // Allocate screen buffer

  FieldList = PS->GetFieldList();    // Get pointer to field list
  FieldList->Refresh();              // Build the field list

  for (Field = FieldList->GetFirstField();     // First field
    Field != NULL;                             // While more
    Field = FieldList->GetNextField(Field)) {  // Next field

      Field->GetScreen(Buff, BuffLen); // Get this fields text
      printf("%02lu,%02lu: %s\n",      // Print "row,col: text"
             Field->GetStartRow(),
             Field->GetStartCol(),
             Buff);
  }
  delete []Buff;
  delete PS;
}
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}

} // end sample
```

SetText

このメソッドは、表示スペース内の 特定のフィールドに、テキストとして渡された文字ストリングを移植します。このテキストがフィールドの長さを超える場合、テキストは 切り捨てられます。テキストがフィールドより短い場合、フィールドの残りはヌルで埋められます。

プロトタイプ

void SetText(char *text)

パラメーター

char *text

フィールドに設定するヌル終了ストリング。

戻り値

なし

例

以下の例は、表示スペース内の特定のフィールドに、 テキストとして渡された文字ストリングを移植する方法を示します。

```
//---------------------------------------------------------------------
// ECLField::SetText
//
// Set the field that contains row 2, column 10 to a value.
//---------------------------------------------------------------------
void Sample36() {

ECLPS        *PS;           // Pointer to PS object
ECLFieldList *FieldList;    // Pointer to field list object
ECLField     *Field;        // Pointer to field object

try {
  PS = new ECLPS('A');               // Create PS object for 'A'
  FieldList = PS->GetFieldList();    // Get pointer to field list
  FieldList->Refresh();              // Build the field list

  // If the field at row 2 col 10 is an input field, set
  // it to a new value.
  Field = FieldList->FindField(2, 10);    // Find field at this location
  if (Field != NULL) {
    if (!Field->IsProtected())            // Make sure its an input field
      Field->SetText("Way cool!");        // Assign new field text
    else
      printf("Position 2,10 is protected.\n");
  }
  else printf("Cannot find field at position 2,10.\n");

  delete PS;
}
```

```
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}

} // end sample
```

IsModified, IsProtected, IsNumeric, IsHighIntensity, IsPenDetectable, IsDisplay

これらの メソッドは、表示スペース内の指定のフィールドに特定の属性があるか どうかを判別します。これらのメソッドは、フィールドが該当する属性を もっている場合は True 値を、その属性をもっていない場合は False 値を戻します。

プロトタイプ

BOOL IsModified()

BOOL IsProtected()

BOOL IsNumeric()

BOOL IsHighIntensity()

BOOL IsPenDetectable()

BOOL IsDisplay()

パラメーター

なし

戻り値

BOOL

該当する属性がある場合は True 値を、属性がない場合は False 値を戻します。

例

以下の例は、指定されたフィールドが属性をもっているかどうかを判別する方法を示します。

```
//-------------------------------------------------------------------
// ECLField::IsModified
// ECLField::IsProtected
// ECLField::IsNumeric
// ECLField::IsHighIntensity
// ECLField::IsPenDetectable
// ECLField::IsDisplay
//
// Iterate over list of fields and print each fields attributes.
//-------------------------------------------------------------------
void Sample37() {

ECLPS        *PS;           // Pointer to PS object
```

```
ECLFieldList *FieldList;    // Pointer to field list object
ECLField     *Field;        // Pointer to field object

try {
  PS = new ECLPS('A');               // Create PS object for 'A'

  FieldList = PS->GetFieldList();     // Get pointer to field list
  FieldList->Refresh();               // Build the field list

  for (Field = FieldList->GetFirstField();     // First field
    Field != NULL;                             // While more
    Field = FieldList->GetNextField(Field)) {  // Next field

      printf("Field at %02lu,%02lu is: ",
             Field->GetStartRow(), Field->GetStartCol());

      if (Field->IsProtected())
        printf("Protect ");
      else
        printf("Input   ");

      if (Field->IsModified())
        printf("Modified   ");
      else
        printf("Unmodified ");

      if (Field->IsNumeric())
        printf("Numeric  ");
      else
        printf("Alphanum ");

      if (Field->IsHighIntensity())
        printf("HiIntensity ");
      else
        printf("Normal      ");

      if (Field->IsPenDetectable())
        printf("Penable ");
      else
        printf("NoPen   ");

      if (Field->IsDisplay())
        printf("Display \n");
      else
        printf("Hidden  \n");
  }
  delete PS;
}
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}

} // end sample

//------------------------------------------------------------------------
```

GetAttribute

このメソッドは、フィールドの属性を戻します。戻される値には、指定可能な各フィールド属性 (modified、protected、numeric、high intensity、pen、および display) について、 ビット・フラグが入っています。これらのビットの詳細については、ECL プレーン ー 形式および内容 (ページ 433)を参照してください。各属性タイプごとに、1 つずつメソッドが提供されます (例えば、IsModified、 IsHighIntensity など)。このメソッドを使用すると、1 回の呼び出しで完全な属性情報を取得することができます。

プロトタイプ

unsigned char GetAttribute()

パラメーター

なし

戻り値

符号なし char

フィールドの属性ビット。

例

以下の例は、フィールドの属性をどのように戻すかを示します。

```
/ ECLField::GetAttribute
//
// Iterate over list of fields and print each fields attribute
// value.
//-------------------------------------------------------------------
void Sample38() {

ECLPS        *PS;           // Pointer to PS object
ECLFieldList *FieldList;    // Pointer to field list object
ECLField     *Field;        // Pointer to field object

try {
  PS = new ECLPS('A');               // Create PS object for 'A'

  FieldList = PS->GetFieldList();    // Get pointer to field list
  FieldList->Refresh();              // Build the field list

  for (Field = FieldList->GetFirstField();     // First field
    Field != NULL;                             // While more
    Field = FieldList->GetNextField(Field)) {  // Next field

      printf("Attribute value for field at %02lu,%02lu is: 0x%02x\n",
             Field->GetStartRow(), Field->GetStartCol(),
             Field->GetAttribute());
  }
  delete PS;
}
catch (ECLErr Err) {
```

```
  printf("ECL Error: %s\n", Err.GetMsgText());
}

} // end sample
```

ECLFieldList クラス

ECLFieldList クラスは、 ホストの表示スペース内のフィールド・リストの処理を行います。アプリケーションは、ECLFieldList オブジェクトを直接作成しては なりません。ECLPS オブジェクトの作成を介して、間接的にしか作成できません。

ECLFieldList には、 表示スペース内のすべてのフィールドの集合が入っています。この集合の各要素が ECLField オブジェクトです。そのプロパティーおよびメソッドの詳細については、 ECLField クラス (ページ 63)を参照してください。

ECLFieldList オブジェクトは、Refresh メソッドを呼び出したときに 表示スペースに入っている内容の静的スナップショットを 提供します。Refresh() の呼び出し後に表示スペースを更新した場合、 その変更内容はフィールド・リストに反映されません。アプリケーションは、フィールド・リストを最新表示にするには、 明示的に Refresh を呼び出さなければなりません。

アプリケーションは、一度 Refresh を呼び出すと、GetFirstField および GetNextField を 使用して、フィールドの集合の中を次々に移動することができます。フィールドの位置が分からない場合、FindField を使用すれば、 リスト内で直接それを見つけられます。

注: GetFirstField、GetNextField、および FindField で戻された すべての ECLField オブジェクト・ポインターは、Refresh を呼び出すか、 または ECLFieldList オブジェクトが破棄されると無効になります。

派生

ECLBase > ECLFieldList

プロパティー

None

ECLFieldList メソッド

以下のセクションでは、ECLFieldList クラスにおいて 有効なメソッドについて説明します。

void Refresh(PS_PLANE Planes) ULONG GetFieldCount() ECLField * GetFirstField() ECLField *GetNextField(ECLField *Prev) ECLField * FindField(ULONG Pos) ECLField * FindField(ULONG Row, ULONG Col)

ECLField *FindField(char* text, PS_DIR DIR=SrchForward); ECLField *FindField(char* text, ULONG Pos, PS_DIR DIR=SrchForward); ECLField *FindField(char* text, ULONG Row, ULONG Col, PS_DIR DIR=SrchForward);

最新表示

このメソッドは、現在表示スペースにある すべてのフィールドのスナップショットを取得します。このオブジェクトによってそれ以前に戻された、 すべての ECLField オブジェクトは無効になります。パフォーマンスを向上するには、必要なプレーンにフィールド・データを 限定します。TextPlane および FieldPlane は、常に取得されることに注意してください。

プロトタイプ

void Refresh(PS_PLANE Planes=TextPlane)

パラメーター

PS_PLANE Planes

フィールドが作成されるプレーン。有効な値は、**TextPlane**、**ColorPlane**、**FieldPlane**、**ExfieldPlane**、および **AllPlanes** (全部について作成する) です。これは、ECLPS.HPP で定義される列挙型です。このオプショナル・パラメーターのデフォルト値は、TextPlane です。

戻り値

なし

例

以下の例は、Refresh メソッドを使用して現在表示スペースにある すべてのフィールドのスナップショットを取得する方法を示します。

```
///----------------------------------------------------------------------
// ECLFieldList::Refresh
//
// Display number of fields on the screen.
//----------------------------------------------------------------------
void Sample39() {

ECLPS        *PS;           // Pointer to PS object
ECLFieldList *FieldList;    // Pointer to field list object

try {
  PS = new ECLPS('A');               // Create PS object for 'A'

  FieldList = PS->GetFieldList();    // Get pointer to field list
  FieldList->Refresh();              // Build the field list

  printf("There are %lu fields on the screen of connection %c.\n",
    FieldList->GetFieldCount(), PS->GetName());
```

```
  delete PS;
}
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}

} // end sample

----------------------------------------
```

GetFieldCount

このメソッドは、ECLFieldList の集合の中にある フィールドの数を戻します (Refresh メソッドの最新の呼び出しをベースにします)。

プロトタイプ

ULONG GetFieldCount()

パラメーター

なし

戻り値

ULONG

ECLFieldList の集合の中にあるフィールドの数。

例

以下の例は、GetFieldCount メソッドを使用して ECLFieldList の集合の中にある フィールド数を戻す方法を示します。

```
//---------------------------------------------------------------------
// ECLFieldList::GetFieldCount
//
// Display number of fields on the screen.
//---------------------------------------------------------------------
void Sample40() {

ECLPS        *PS;           // Pointer to PS object
ECLFieldList *FieldList;    // Pointer to field list object

try {
  PS = new ECLPS('A');               // Create PS object for 'A'

  FieldList = PS->GetFieldList();    // Get pointer to field list
  FieldList->Refresh();              // Build the field list

  printf("There are %lu fields on the screen of connection %c.\n",
    FieldList->GetFieldCount(), PS->GetName());
```

```
  delete PS;
}
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}

} // end sample
```

GetFirstField

このメソッドは、集合の中の 最初の ECLField オブジェクトを指すポインターを 戻します。ECLFieldList は、ECLField オブジェクトの集合を含んでいます。詳しくは、ECLField クラス (ページ 63)を参照してください。集合の中にフィールドがない場合、メソッドは NULL ポインターを戻します。

プロトタイプ

ECLField * GetFirstField();

パラメーター

なし

戻り値

ECLField *

ECLField オブジェクトを指すポインター。接続内にフィールドがない場合は、ヌルが戻されます。

例

以下の例は、GetFirstField メソッドを使用して集合の中の 最初の ECLField オブジェクトを指すポインターを戻す方法を示します。

```
/-------------------------------------------------------------------
// ECLFieldList::GetFirstField
//
// Display starting position of every input (unprotected) field.
//-------------------------------------------------------------------
void Sample41() {

ECLPS        *PS;           // Pointer to PS object
ECLFieldList *FieldList;    // Pointer to field list object
ECLField     *Field;        // Pointer to field object

try {
  PS = new ECLPS('A');               // Create PS object for 'A'

  FieldList = PS->GetFieldList();    // Get pointer to field list
  FieldList->Refresh();              // Build the field list

  // Interate over (only) unprotected fields
```

```
  printf("List of input fields:\n");
  for (Field = FieldList->GetFirstField(GetUnprotected);
    Field != NULL;
    Field = FieldList->GetNextField(Field, GetUnprotected)) {

    printf("Input field starts at %02lu,%02lu\n",
            Field->GetStartRow(), Field->GetStartCol());
  }
  delete PS;
}
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}

} // end sample
```

GetNextField

このメソッドは、集合の中の 指定されたオブジェクトの次の ECLField オブジェクトを戻します。集合の中で、指定したオブジェクトの後にもうオブジェクトがない場合は、NULL ポインターが戻されます。アプリケーションは、このメソッドを繰り返して呼び出すことで、 集合の中の ECLField オブジェクトを反復することができます。

プロトタイプ

ECLField *GetNextField(ECLField *Prev)

パラメーター

ECLField *Prev

集合の中の任意の ECLField オブジェクトを指すポインター。戻されるポインターは、このオブジェクトの次のオブジェクトです。この値が NULL の場合、集合の中の最初のオブジェクトを指すポインターが戻されます。このポインターは、GetFirstField、GetNextField、または FindField メンバー関数によって 戻されるポインターです。

戻り値

ECLField *

集合の中の次のオブジェクトを指すポインター。集合の中で、Prev オブジェクトの後にもうオブジェクトがない場合、 NULL が戻されます。

例

以下の例は、GetNextFieldInfo メソッドを使用して、集合の中の 次の ECLField オブジェクトを指すポインターを戻す方法を示します。

```
///-------------------------------------------------------------------
// ECLFieldList::GetNextField
//
```

```
// Display starting position of every input (unprotected) field.
//-------------------------------------------------------------------
void Sample42() {

ECLPS        *PS;           // Pointer to PS object
ECLFieldList *FieldList;    // Pointer to field list object
ECLField     *Field;        // Pointer to field object

try {
  PS = new ECLPS('A');               // Create PS object for 'A'

  FieldList = PS->GetFieldList();    // Get pointer to field list
  FieldList->Refresh();              // Build the field list

  // Interate over (only) unprotected fields
  printf("List of input fields:\n");
  for (Field = FieldList->GetFirstField(GetUnprotected);
    Field != NULL;
    Field = FieldList->GetNextField(Field, GetUnprotected)) {

    printf("Input field starts at %02lu,%02lu\n",
            Field->GetStartRow(), Field->GetStartCol());
  }
  delete PS;
}
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}

} // end sample
```

FindField

このメソッドは、テキストまたは位置を 使用して ECLFieldList の集合の中のフィールドを検出します。位置は、線形位置または行/列の位置のどちらでも構いません。フィールドにテキストまたは位置を含めると、 そのフィールド用の ECLField オブジェクトを指すポインターが戻されます。戻されるポインターは、フィールド・リストの集合の中のオブジェクトを指す ポインターです。フィールドが見つからない場合、NULL が戻されます。テキストの検索の場合、開始位置を指定しないかぎり、検索は 行 1 桁 1 から開始されます。またテキストの場合も、デフォルトではこのメソッドは順方向に検索します。 ただし、検索方向を明示的に指定することもできます。

注: テキストの検索は、テキストが 複数のフィールドにまたがっていても正常に行われます。戻されるフィールド・オブジェクトは、見つかったテキストが始まるフィールドです。

プロトタイプ

```
ECLField *FindField(ULONG Pos);
ECLField *FindField(ULONG Row, ULONG Col);
ECLField *FindField(char* text, PS_DIR DIR=SrchForward);
```

ECLField *FindField(char* text, ULONG Pos, PS_DIR DIR=SrchForward);
ECLField *FindField(char* text, ULONG Row, ULONG Col, PS_DIR DIR=SrchForward);

パラメーター

ULONG Pos

検索する線形位置またはテキスト検索を開始する線形位置。

ULONG Row

検索する行位置またはテキスト検索を開始する行。

ULONG Col

検索する桁位置またはテキスト検索を開始する桁。

char *text

検索するストリング。

PS_DIR Dir

検索する方向。

戻り値

ECLField *

フィールドが見つかった場合は、ECLField オブジェクトを指すポインター。フィールドが見つからなかった場合は NULL。次の Refresh の呼び出し後、戻されたポインターは無効になります。

例

以下に、FindField メソッドの例を示します。

```
//-------------------------------------------------------------------
// ECLFieldList::FindField
//
// Display the field which contains row 2 column 10.  Also find
// the first field containing a particular string.
//-------------------------------------------------------------------
void Sample43() {

ECLPS        *PS;           // Pointer to PS object
ECLFieldList *FieldList;    // Pointer to field list object
ECLField     *Field;        // Pointer to field object
char         Buff[4000];

try {
  PS = new ECLPS('A');               // Create PS object for 'A'

  FieldList = PS->GetFieldList();    // Get pointer to field list
  FieldList->Refresh();              // Build the field list

  // Find by row,column coordinate
```

```
  Field = FieldList->FindField(2, 10);
  if (Field != NULL) {
    Field->GetText(Buff, sizeof(Buff));
    printf("Field at 2,10: %s\n", Buff);
  }
  else printf("No field found at 2,10.\n");

  // Find by text.  Note that text may span fields, this
  // will find the field in which the text starts.

  Field = FieldList->FindField("HCL");
  if (Field != NULL) {
    printf("String 'HCL' found in field that starts at %lu,%lu.\n",
            Field->GetStartRow(), Field->GetStartCol());
  }
  else printf("String 'HCL' not found.\n");

  delete PS;
}
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}

} // end sample

//---------------------------
```

ECLKeyNotify クラス

ECLKeyNotify は、抽象基本クラスです。アプリケーションは、このクラスの インスタンスを直接作成することはできません。アプリケーションでこのクラスを使用するには、ECLKeyNotify から派生した 独自のクラスを定義しなければなりません。アプリケーションは、 その派生クラス内に NotifyEvent() メンバー関数を実装しなければなりません。また、 オプションで NotifyError() および NotifyStop() メンバー関数を実装することも できます。

アプリケーションがキー・ストローク・イベントの通知を受けられるように するには、ECLKeyNotify クラスを使用します。またアプリケーションは、キー・ストロークをフィルターに掛ける (除去する) ことにより、キー・ストロークがホスト画面に送られないようにしたり、 キー・ストロークを他のキー・ストロークに置換したりすることができます。アプリケーションがすべてのキー・ストロークについて常に通知を受けられるようにするため、キー・ストローク通知がキューに入れられます。実際の物理キーボードで行われたキー・ストロークだけが、このオブジェクトで 検出されます。他の ECL オブジェクト (ECLPS::SendKeys など) からホストに送られた キー・ストロークの場合、キー・ストローク通知イベントは生じません。

アプリケーションがキー・ストローク・イベントの通知を受けるには、 次に示すステップを実行しなければなりません。

1. ECLKeyNotify から派生したクラスを定義します。
2. その派生クラスを採用し、NotifyEvent() メンバー関数を実装します。
3. オプションで、NotifyError() または NotifyStop() 関数 (あるいはその両方) を 実装します。

4. 派生クラスのインスタンスを作成します。
5. そのインスタンスを ECLPS::RegisterKeyEvent() 関数で登録します。

ここに示された例は、それがどのように行われるかを例示しています。上記のステップを完了すると、エミュレーター・ウィンドウでの各キー・ストロークのたびに、 アプリケーションの NotifyEvent() メンバー関数が呼び出されます。この関数には、キー・ストロークのタイプ (プレーンな ASCII キーまたは 特殊ファンクション・キー) およびキーの値 (単一の ASCII 文字、または ファンクション・キーを表すキーワード) を示すパラメーターが渡されます。アプリケーションは、ECLPS::SendKeys() などの他の ECL 関数の呼び出しを含め、 NotifyEvent() プロシージャーで必要な任意の関数を実行できます。アプリケーションは、キー・ストロークをフィルターに掛けるかどうか (フィルターに掛ける (廃棄) する場合は 1 を戻し、通常に処理する場合は 0 を戻します) を 示すため NotifyEvent() から値を戻します。

キー・ストローク・イベントの生成時にエラーが検出された場合、 ECLErr オブジェクトを使用して NotifyError() メンバー関数が呼び出されます。エラーの特性に応じて、エラー後にキー・ストローク・イベントが続けて生成されるかどうかが決まります。イベント生成が終了したとき (エラーか、ECLPS::UnregisterKeyEvent の呼び出しか、 または ECLPS オブジェクトの破棄のいずれかが原因で)、NotifyStop() メンバー関数が呼び出されます。イベント通知が終了するときには、NotifyStop() メンバー関数が常に呼び出され、 アプリケーション・オブジェクトの登録が抹消されます。

アプリケーションが NotifyError() メンバー関数の実装を行わない場合、 デフォルトの実装が使われます (単純なメッセージ・ボックスがユーザーに対して表示されます)。アプリケーションがデフォルトの振る舞いをオーバーライドするには、 アプリケーションの派生クラス内に NotifyError() 関数を実装します。同様に、アプリケーションがこの関数を提供しない場合、 デフォルトの NotifyStop() 関数が使われます (デフォルトの振る舞いでは何も行われません)。

またアプリケーションは、派生したクラス用に自身のコンストラクターおよび デストラクターを任意で提供できることに注意してください。これが便利なのは、 アプリケーションが特定のインスタンス別データをそのクラス内に保管してから、 その情報をコンストラクター上のパラメーターとして渡したい場合です。例えば、アプリケーションにおいて、キー・ストロークが発生したら アプリケーション・ウィンドウにメッセージをポストしたい場合があります。アプリケーションは、ウィンドウ・ハンドルをグローバル変数として定義する (このハンドルを、NotifyEvent() 関数に見えるようにするため) 代わりに、 ウィンドウ・ハンドルを受け取ってクラス・メンバーのデータ域に保管する クラス用のコンストラクターとして定義することができます。

アプリケーションは、イベントを受け取るために通知オブジェクトを 登録しているかぎり、そのオブジェクトを破棄してはなりません。

複数の接続でのキー・ストロークを受け取れるように、 キー・ストローク通知オブジェクトの同一のインスタンスを、 複数の ECLPS オブジェクトに登録することができます。つまり、アプリケーションは、このオブジェクトの 1 つのインスタンスを使用して、 さまざまなセッションにおいてキー・ストロークを処理できるということです。アプリケーションが別々の接続でのイベントをそれぞれ区別できるように、 イベントの発生対象の ECLPS オブジェクトを指すポインターがメンバー関数に渡されます。ここに示した例は、2 つの接続においてキー・ストロークを処理するために 同一のオブジェクトを使用しています。

実装上の制約事項: 現在、ECLPS オブジェクトでは、1 つの指定された接続に対して 1 つしか通知オブジェクトを登録できません。その ECLPS オブジェクト用に通知オブジェクトが既に登録されて いる場合、ECLPS::RegisterKeyEvent からエラーがスローされます。

派生

ECLBase > ECLNotify > ECLKeyNotify

例

以下に、ECLKeyNotify オブジェクトの作成方法と使用方法の例を示します。

```
// ECLKeyNotify class
//
// This sample demonstrates the use of:
//
// ECLKeyNotify::NotifyEvent
// ECLKeyNotify::NotifyError
// ECLKeyNotify::NotifyStop
// ECLPS::RegisterKeyEvent
// ECLPS::UnregisterKeyEvent
//---------------------------------------------------------------------

//.....................................................................
// Define a class derived from ECLKeyNotify
//.....................................................................
class MyKeyNotify: public ECLKeyNotify
{
public:
  // Define my own constructor to store instance data
  MyKeyNotify(HANDLE DataHandle);

  // We have to implement this function
  virtual int NotifyEvent(ECLPS *PSObj, char const KeyType[2],
                          const char * const KeyString);

  // We choose to implement this function
  void NotifyStop (ECLPS *PSObj, int Reason);

  // We will take the default behaviour for this so we
  // don't implement it in our class:
  // void NotifyError (ECLPS *PSObj, ECLErr ErrObject);

private:
  // We will store our application data handle here
  HANDLE MyDataH;
};

 //.....................................................................
MyKeyNotify::MyKeyNotify(HANDLE DataHandle)   // Constructor
//.....................................................................
{
  MyDataH = DataHandle;  // Save data handle for later use
}

//.....................................................................
int MyKeyNotify::NotifyEvent(ECLPS *PSObj,
                             char const KeyType[2],
                             const char * const KeyString)
```

```
//.............................................................

{
  // This function is called whenever a keystroke occurs.  We will
  // just do something simple: when the user presses PF1 we will
  // send a PF2 to the host instead.  All other keys will be unchanged.

  if (KeyType[0] == 'M') {             // Is this a mnemonic keyword?
    if (!strcmp(KeyString, "[pf1]")) { // Is it a PF1 key?
      PSObj->SendKeys("[pf2]");        // Send PF2 instead
      printf("Changed PF1 to PF2 on connection %c.\n",
             PSObj->GetName());
      return 1;                        // Discard this PF1 key
    }
  }

  return 0;                            // Process key normally
}
```

```
//.............................................................
void MyKeyNotify::NotifyStop (ECLPS *PSObj, int Reason)
//.............................................................
{
  // When notification ends, display message
  printf("Keystroke intercept for connection %c stopped.\n", PSObj->GetName());
}

//.............................................................
// Create the class and start keystroke processing on A and B.
//.............................................................
void Sample44() {

ECLPS *PSA, *PSB;       // PS objects
MyKeyNotify *Event;     // Ptr to my event handling object
HANDLE InstData;      // Handle to application data block (for example)

try {

  PSA = new ECLPS('A');               // Create PS objects
  PSB = new ECLPS('B');
  Event = new MyKeyNotify(InstData);  // Create event handler

  PSA->RegisterKeyEvent(Event);       // Register for keystroke events
  PSB->RegisterKeyEvent(Event);       // Register for keystroke events
```

```
  // At this point, any keystrokes on A or B will cause the
  // MyKeyEvent::NotifyEvent() function to execute.  For
  // this sample, we put this thread to sleep during this
  // time.

  printf("Processing keystrokes for 60 seconds on A and B...\n");
  Sleep(60000);

  // Now stop event generation.  This will cause the NotifyStop
  // member to be called.
  PSA->UnregisterKeyEvent(Event);
```

```
  PSB->UnregisterKeyEvent(Event);

  delete Event;  // Don't delete until after unregister!
  delete PSA;
  delete PSB;
}
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}

} // end sample

//---------------------------------------------------------------------
```

ECLKeyNotify メソッド

以下のセクションでは、ECLKeyNotify クラスにおいて有効なメソッドについて説明します。

virtual int NotifyEvent (ECLPS *PSObj, char const KeyType [2], const char * const KeyString) =0 virtual void NotifyError (ELLPS *PSobj, ECLErr ErrObject) virtual void NotifyStop (ELLPS *PSObj, int Reason)

NotifyEvent

このメソッドは、純粋仮想メンバー関数です (アプリケーションは ECLKeyNotify から派生したクラス内にこの関数を実装しなければなりません)。*must* キー・ストローク・イベントが発生したときと、キー・ストローク・イベントのために オブジェクトが登録されたときは、常にこの関数が呼び出されます。戻り値は、キー・ストロークの処置を示します (廃棄する場合は 1、処理する場合は 0)。

プロトタイプ

virtual int NotifyEvent (ECLPS *PSObj, char const KeyType [2], const char * const KeyString) =0

パラメーター

ECLPS *PSObj

これは、イベントが発生した ECLPS オブジェクトを指すポインターです。

char const KeyType[2]

これは、次のとおりの、キー・タイプを示す 1 文字のヌル終了ストリングです。

A = プレーンな ASCII キー・ストローク M = ニーモニック・キーワード

const char * const KeyString

これは、キー・ストロークまたは略号キーワードを含んだヌル終了ストリングです。キーワードは、常に小文字になります (例えば、"[enter]")。略号キーワードのリストは、Sendkeys 略号キーワード (ページ 430)を参照してください。

戻り値

整数

これは、フィルター標識です。

1 = フィルター (廃棄) キー・ストローク 0 = 処理キー・ストローク (ホストへ送信)

NotifyError

このメソッドは、ECLPS オブジェクトがキー・ストローク・イベント生成時に、 エラーを検出するたびに呼び出されます。エラー・オブジェクトには、そのエラーについての情報が 含まれます (ECLErr クラス (ページ 59)を参照)。エラーの特性に応じて、エラー後に キー・ストローク・イベントが続けて生成されることがあります。エラーが原因でキー・ストローク・イベント生成が停止した場合、 NotifyStop() 関数が呼び出されます。

プロトタイプ

virtual void NotifyError (ELLPS *PSobj, ECLErr ErrObject)

パラメーター

ECLPS *PSObj

これは、エラーが発生した ECLPS オブジェクトを指すポインターです。

ECLErr ErrObject

これは、エラーを記述した ECLErr オブジェクトです。

戻り値

なし

NotifyStop

キー・ストローク・イベント生成が何らかの理由で (例えば、 エラー条件、ECLPS::UnregisterKeyEvent の呼び出し、または ECLPS オブジェクトの破棄などが原因で) 停止すると、このメソッドが呼び出されます。

プロトタイプ

virtual void NotifyStop (ELLPS *PSObj, int Reason)

パラメーター

ECLPS *PSObj

これは、イベントが停止している ECLPS オブジェクトを指すポインターです。

int Reason

これは未使用 (ゼロ) です。

戻り値

なし

ECLListener クラス

ECLListener は、すべての HACL "リスナー" オブジェクトの基本クラスです。リスナーは、非同期イベントの特定のタイプに登録されたオブジェクトです。リスナー・オブジェクトのメソッドは、イベントが起こるかエラーが検出されたときに 呼び出されます。

ECLListener クラスには、共用メソッドはありません。

派生

ECLBase > ECLListener

使用上の注意

アプリケーションはこのクラスを直接使用しませんが、これから派生したクラスの インスタンスを作成します (例えば、ECLPSListener)。

ECLOIA クラス

ECLOIA は、オペレーター情報域 (OIA) サービスを提供します。

ECLOIA は ECLConnection から派生するため、ECLConnection オブジェクトに 含まれるすべての情報を取得できます。詳しくは、ECLConnection クラス (ページ 24)を参照してください。

ECLOIA オブジェクトは、作成時に識別された接続用に作成されます。ECLOIA オブジェクトを作成するには、 通常は ECLConnList オブジェクトから取得される接続名 (単一の A から Z、または a から z の英字) または接続ハンドルを 渡します。1 つの名前またはハンドルに対して、一度に 1 つの Z and I Emulator for Windows 接続しかオープンできません。

派生

ECLBase > ECLConnection > ECLOIA

使用上の注意

ECLSession クラスは、このオブジェクトのインスタンスを作成します。アプリケーションで他のサービスを必要としない場合、 このオブジェクトを直接作成しても構いません。そうでない場合は、ECLSession オブジェクトを使用して、 必要なすべてのオブジェクトを作成することを検討してください。

ECLOIA メソッド

以下のセクションでは、ECLOIA クラスにおいて 有効なメソッドについて説明します。

ECLOIA(char ConnName) ECLOIA(long ConnHandle) ~ECLOIA() BOOL IsAlphanumeric() BOOL IsAPL() BOOL IsUpperShift() BOOL IsNumeric() BOOL IsCapsLock() BOOL IsInsertMode() BOOL IsCommErrorReminder() BOOL IsMessageWaiting() BOOL WaitForInputReady(long nTimeOut = INFINITE) BOOL WaitForAppAvailable(long nTimeOut = INFINITE) BOOL WaitForSystemAvailable(long nTimeOut = INFINITE) BOOL WaitForTransition(BYTE nIndex = 0xFF, long nTimeOut = INFINITE) INHIBIT_REASON InputInhibited() ULONG GetStatusFlags()

ECLOIA コンストラクター

このメソッドは、接続名 (単一の A から Z、または a から z の英字) または接続ハンドル から ECLOIA オブジェクトを作成します。Z and I Emulator for Windows は、1 つの名前につき 1 つだけ接続を開始できます。

プロトタイプ

ECLOIA(char ConnName)

ECLOIA(long ConnHandle)

パラメーター

char ConnName

1 文字の接続の短縮名 (A から Z、または a から z)。

long ConnHandle

ECL 接続のハンドル。

戻り値

なし

例

以下の例は、接続名を使用して ECLOIA オブジェクトを作成する方法を示します。

```
// ECLOIA::ECLOIA              (Constructor)
//
```

```
// Build an OIA object from a name, and another from a handle.
//-------------------------------------------------------------------
void Sample45() {

ECLOIA *OIA1, *OIA2;    // Pointer to OIA objects
ECLConnList ConnList;   // Connection list object

try {
  // Create OIA object for connection 'A'
  OIA1 = new ECLOIA('A');

  // Create OIA object for first connection in conn list
  OIA2 = new ECLOIA(ConnList.GetFirstConnection()->GetHandle());

  printf("OIA #1 is for connection %c, OIA #2 is for connection %c.\n",
         OIA1->GetName(), OIA2->GetName());
  delete OIA1;
  delete OIA2;
}
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}

} // end sample

------------------------------------------
```

IsAlphanumeric

このメソッドは、カーソルが英数字ロケーションに あると OIA が示すかどうかを判別します。

プロトタイプ

BOOL IsAlphanumeric()

パラメーター

なし

戻り値

BOOL

キーボードが英数字モードの場合は True、 キーボードが英数字モードでない場合は False。

例

以下の例は、キーボードが英数字モードであると OIA が示すかどうかを 判別する方法を示します。

```
//-------------------------------------------------------------------
// ECLOIA::IsAlphanumeric
//
// Determine status of connection 'A' OIA indicator
//-------------------------------------------------------------------
```

```
void Sample46() {

ECLOIA OIA('A');   // OIA object for connection A

if (OIA.IsAlphanumeric())
  printf("Alphanumeric.\n");
else
  printf("Not Alphanumeric.\n");

} // end sample
```

IsAPL

このメソッドは、キーボードが APL モードである と OIA が示すかどうかを判別します。

プロトタイプ

BOOL IsAPL()

パラメーター

なし

戻り値

BOOL

キーボードが APL モードの場合は True、 キーボードが APL モードでない場合は False。

例

以下の例は、キーボードが APL モードであると OIA が示すかどうかを 判別する方法を示します。

```
//---------------------------------------------------------------------
// ECLOIA::IsAPL
//
// Determine status of connection 'A' OIA indicator
//---------------------------------------------------------------------
void Sample47() {

ECLOIA OIA('A');   // OIA object for connection A

if (OIA.IsAPL())
  printf("APL.\n");
else
  printf("Not APL.\n");

} // end sample

//------------------------
```

IsUpperShift

このメソッドは、キーボードが上段シフト・モードである と OIA が示すかどうかを判別します。

プロトタイプ

BOOL IsUpperShift()

パラメーター

なし

戻り値

BOOL

キーボードが上段シフト・モードの場合は True、 キーボードが上段シフト・モードでない場合は False。

例

以下の例は、キーボードが上段シフト・モードであると OIA が示すか どうかを判別する方法を示します。

```
//-------------------------------------------------------------------
// ECLOIA::IsUpperShift
//
// Determine status of connection 'A' OIA indicator
//-------------------------------------------------------------------
void Sample51() {

ECLOIA OIA('A');   // OIA object for connection A

if (OIA.IsUpperShift())
  printf("UpperShift.\n");
else
  printf("Not UpperShift.\n");

} // end sample
```

IsNumeric

このメソッドは、カーソルが数字のみのロケーションに あると OIA が示すかどうかを判別します。

プロトタイプ

BOOL IsNumLock()

パラメーター

なし

戻り値

BOOL

Numeric がオンのときは True で、Numeric がオフのときは False。

例

以下の例は、カーソルが数字ロケーションにある と OIA が示すかどうかを判別する方法を示します。

```
//-------------------------------------------------------------------
// ECLOIA::IsNumeric
//
// Determine status of connection 'A' OIA indicator
//-------------------------------------------------------------------
void Sample52() {

ECLOIA OIA('A');   // OIA object for connection A

if (OIA.IsNumeric())
  printf("Numeric.\n");
else
  printf("Not Numeric.\n");

} // end sample
```

IsCapsLock

このメソッドは、キーボードの Caps Lock がオンに なっていると OIA が示すかどうかを判別します。

プロトタイプ

BOOL IsCapsLock()

パラメーター

なし

戻り値

BOOL

Caps Lock がオンの場合は True、Caps Lock がオンでない場合は False。

例

以下の例は、キーボードの Caps Lock がオンになっている と OIA が示すかどうかを判別する方法を示します。

```
//-------------------------------------------------------------------
// ECLOIA::IsCapsLock
//
```

```
// Determine status of connection 'A' OIA indicator
//-------------------------------------------------------------------
void Sample53() {

ECLOIA OIA('A');   // OIA object for connection A

if (OIA.IsCapsLock())
  printf("CapsLock.\n");
else
  printf("Not CapsLock.\n");

} // end sample
```

IsInsertMode

このメソッドは、キーボードが挿入モードである と OIA が示すかどうかを判別します。

プロトタイプ

BOOL IsInsertMode()

パラメーター

なし

戻り値

BOOL

キーボードが挿入モードの場合は True、 キーボードが挿入モードでない場合は False。

例

以下の例は、キーボードが挿入モードであると OIA が示すかどうかを 判別する方法を示します。

```
//-------------------------------------------------------------------
// ECLOIA::IsInsertMode
//
// Determine status of connection 'A' OIA indicator
//-------------------------------------------------------------------
void Sample54() {

ECLOIA OIA('A');   // OIA object for connection A

if (OIA.IsInsertMode())
  printf("InsertMode.\n");
else
  printf("Not InsertMode.\n");

} // end sample
```

IsCommErrorReminder

このメソッドは、通信エラー状況メッセージが 存在すると OIA が示すかどうかを判別します。

プロトタイプ

BOOL IsCommErrorReminder()

パラメーター

なし

戻り値

BOOL

条件が存在する場合は True、条件が存在しない場合は False。

例

以下の例は、通信エラー状況メッセージが存在する と OIA が示すかどうかを判別する方法を示します。

```
//---------------------------------------------------------------------
// ECLOIA::IsCommErrorReminder
//
// Determine status of connection 'A' OIA indicator
//---------------------------------------------------------------------
void Sample55() {

ECLOIA OIA('A');   // OIA object for connection A

if (OIA.IsCommErrorReminder())
  printf("CommErrorReminder.\n");
else
  printf("Not CommErrorReminder.\n");

} // end sample

//
```

IsMessageWaiting

このメソッドは、メッセージ待機標識がオンである と OIA が示すかどうかを判別します。これは、5250 接続にのみ起こります。

プロトタイプ

BOOL IsMessageWaiting()

パラメーター

なし

戻り値

BOOL

メッセージ待機標識がオンの場合は True、 メッセージ待機標識がオンでない場合は False。

例

以下の例は、メッセージ待機標識がオンである と OIA が示すかどうかを判別する方法を示します。

```
------------------------------------------------------------------
// ECLOIA::IsMessageWaiting
//
// Determine status of connection 'A' OIA indicator
//------------------------------------------------------------------
void Sample56() {

ECLOIA OIA('A');   // OIA object for connection A

if (OIA.IsMessageWaiting())
  printf("MessageWaiting.\n");
else
  printf("Not MessageWaiting.\n");

} // end sample
```

WaitForInputReady

WaitForInputReady メソッドは、autECLOIA オブジェクトに関連した接続の OIA が、 この接続にキーボード入力の受け入れが可能であることを示すまで待機します。

プロトタイプ

BOOL WaitForInputReady(long nTimeOut = INFINITE)

パラメーター

long nTimeOut

待機時間の最大長をミリ秒で指定します。 このパラメーターはオプションです。デフォルトは、Infinite (無期限) です。

戻り値

メソッドは、条件が合致していれば True を 戻し、nTimeOut (ミリ秒) が経過した場合は False を戻します。

WaitForSystemAvailable

WaitForSystemAvailable メソッドは、ECLOIA オブジェクトに接続されたセッション の OIA が、そのセッションがホスト・システムに接続されていると知らせるまで待機します。

プロトタイプ

BOOL WaitForSystemAvailable(long nTimeOut = INFINITE)

パラメーター

long nTimeOut

待機時間の最大長をミリ秒で指定します。 このパラメーターはオプションです。デフォルトは、Infinite (無期限) です。

戻り値

メソッドは、条件が合致していれば True を 戻し、nTimeOut (ミリ秒) が経過した場合は False を戻します。

WaitForAppAvailable

WaitForAppAvailable メソッドは、接続されたセッションの OIA が、 アプリケーションが初期設定され、使用可能であることを示すまで待機します。

プロトタイプ

BOOL WaitForAppAvailable(long nTimeOut = INFINITE)

パラメーター

long nTimeOut

待機時間の最大長をミリ秒で指定します。 このパラメーターはオプションです。デフォルトは、Infinite (無期限) です。

戻り値

メソッドは、条件が合致していれば True を 戻し、nTimeOut (ミリ秒) が経過した場合は False を戻します。

WaitForTransition

WaitForTransition メソッドは、接続されたセッションの OIA の指定された位置の値が 変わるのを待ちます。

プロトタイプ

BOOL WaitForTransition(BYTE nIndex = 0xFF, long nTimeOut = INFINITE)

パラメーター

BYTE nIndex

モニターする OIA の 1 バイトの 16 進数位置。このパラメーターはオプションです。デフォルトは、3 です。

long nTimeOut

待機時間の最大長をミリ秒で指定します。 このパラメーターはオプションです。デフォルトは、Infinite (無期限) です。

戻り値

メソッドは、条件が合致していれば True を 戻し、nTimeOut (ミリ秒) が経過した場合は False を戻します。

InputInhibited

このメソッドは、入力が使用禁止かどうかを示す列挙値を戻します。入力が使用禁止の場合、使用禁止の理由を判別できます。複数の理由で入力が使用禁止になっている場合、最大の列挙値が戻されます (例えば、通信エラーとプロトコル・プログラム・エラーがあると、ProgCheck 値が戻されます)。

プロトタイプ

INHIBIT_REASON InputInhibited ()

パラメーター

なし

戻り値

INHIBIT_REASON

ECLOIA.HPP に定義されているとおりの、INHIBIT_REASON 値のいずれかを戻します。現在入力が使用禁止になっていない場合、NotInhibited 値が戻されます。

例

以下の例は、入力が使用禁止になっているかどうかを判別する方法を示します。

```
//-------------------------------------------------------------------
// ECLOIA::InputInhibited
//
// Determine status of connection 'A' OIA indicator
```

```
//---------------------------------------------------------------------
void Sample57() {

ECLOIA OIA('A');   // OIA object for connection A

switch (OIA.InputInhibited()) {
case NotInhibited:
  printf("Input not inhibited.\n");
  break;
case SystemWait:
  printf("Input inhibited for SystemWait.\n");
  break;
case CommCheck:
  printf("Input inhibited for CommCheck.\n");
  break;
case ProgCheck:
  printf("Input inhibited for ProgCheck.\n");
  break;
case MachCheck:
  printf("Input inhibited for MachCheck.\n");
  break;
case OtherInhibit:
  printf("Input inhibited for OtherInhibit.\n");
  break;
default:
  printf("Input inhibited for unknown reason.\n");
  break;
}
} // end sample
```

GetStatusFlags

このメソッドは、各種 OIA 標識を表す状況ビットのセットを戻します。このメソッドを使用すれば、いくつもの異なる IsXXX メソッドを呼び出す代わりに、 単一の呼び出しで OIA 標識のセットを収集することができます。戻される各ビットは、 単一の OIA 標識を表しており、この値が 1 の場合は標識はオン (True)、0 の場合は オフ (False) を意味します。ビット・マスク定数のセットは ECLOIA.HPP ヘッダー・ファイルで定義されており、戻された 32 ビット値の個々の標識を分離します。

プロトタイプ

ULONG GetStatusFlags()

パラメーター

なし

戻り値

ULONG

ビット・フラグのセットは、次のように定義されます。

ビット位置	マスク定数	説明
31 (msb)	OIAFLAG_ALPHANUM	IsAlphanumeric
30	OIAFLAG_APL	IsAPL
26	OIAFLAG_UPSHIFT	IsUpperShift
25	OIAFLAG_NUMERIC	IsNumeric
24	OIAFLAG_CAPSLOCK	IsCapsLock
23	OIAFLAG_INSERT	IsInsertMode
22	OIAFLAG_COMMERR	IsCommErrorReminder
21	OIAFLAG_MSGWAIT	IsMessageWaiting
20	OIAFLAG_ENCRYPTED	IsConnectionEncrypted
19-4		予約済み
3-0	OIAFLAG_INHIBMASK	InputInhibited: 0=NotInhibited 1=SystemWait 2=CommCheck 3=ProgCheck 4=MachCheck 5=OtherInhibit

RegisterOIAEvent

このメンバー関数は、OIA 更新イベントの通知を受け取るためのアプリケーション・オブジェクト を登録します。アプリケーションでこの関数を使用するには、 ECLOIANotify から派生したオブジェクトを作成しなければなりません。作成すると、そのオブジェクトを指すポインターは この登録関数に渡されます。通知オブジェクトの数が、同時に登録されることがあります。複数のリスナーがイベントを受信する順序は、定義されず想定することはできません。

この関数を使用して ECLOIANotify オブジェクトが登録されれば、OIA に更新が行われるたびに、 この NotifyEvent() メソッドが呼び出されます。短時間内での OIA に対する複数の更新は、 単一のイベントに集約されることがあります。

アプリケーションは、これを破棄するのに先立って通知オブジェクトを登録抹消する必要があります。ECLOIA オブジェクトが破棄されると、このオブジェクトは自動的に抹消されます。

プロトタイプ

void RegisterOIAEvent(ECLOIANotify * notify)

パラメーター

ECLOIANotify *

登録すべき ECLOIANotify オブジェクトに対するポインター。

戻り値

なし

UnregisterOIAEvent

このメンバー関数は、RegisterOIAEvent 関数を使用して通信イベント用に事前に 登録されているアプリケーション・オブジェクトの登録を抹消します。イベントを受信するための登録済みオブジェクトの場合、先にこの関数を呼び出して その登録を抹消しないかぎり、オブジェクトを破棄してはなりません。特定のオブジェクトが現在登録されていない場合は、アクションは取られず エラーは発生しません。

ECLOIANotify オブジェクトが登録されていない場合は、この NotifyStop() メソッドが 呼び出されます。

プロトタイプ

void UnregisterOIAEvent(ECLOIANotify * notify)

パラメーター

ECLPSNotify *

抹消すべき ECLOIANotify オブジェクトに対するポインター。

戻り値

なし

ECLOIANotify クラス

ECLOIANotify は、抽象基本クラスです。アプリケーションは、このクラスの インスタンスを直接作成することはできません。アプリケーションでこのクラスを使用するには、ECLOIANotify から派生した 独自のクラスを定義しなければなりません。アプリケーションは、 その派生クラス内に NotifyEvent() メンバー関数を実装しなければなりません。また、 オプションで NotifyError() および NotifyStop() メンバー関数を実装することも できます。

アプリケーションがオペレーター情報域に対する更新についての通知を 受けられるようにするには、ECLOIANotify クラスを使用します。OIA の標識が更新されるたびにイベントが生成されます。

派生

ECLBase > ECLNotify > ECLOIANotify

使用上の注意

アプリケーションがこのクラスを使用する OIA 更新の通知を受けるには、 次に示すステップを実行しなければなりません。

1. ECLOIANotify から派生したクラスを定義します。
2. ECLOIANotify から派生したクラスの NotifyEvent メソッドを実装します。
3. オプションで、ECLOIANotify の他のメンバー関数を実装します。
4. 派生クラスのインスタンスを作成します。
5. そのインスタンスを ECLOIA::RegisterOIAEvent() メソッドで登録します。

登録が完了した後では、OIA 標識の更新は ECLOIANotify から派生したクラス の NotifyEvent() メソッドが呼び出される原因となります。

短時間に発生する複数の OIA の更新は、単一のイベント通知に集約される ことがあるので注意してください。

アプリケーションは、派生したクラス用に自身のコンストラクター およびデストラクターを任意で提供することができます。これが便利なのは、 アプリケーションが特定のインスタンス固有データをそのクラス内に保管してから、 その情報をコンストラクター上のパラメーターとして渡す必要がある場合です。

イベントの登録時にエラーが検出された場合、ECLErr オブジェクトを 使用して NotifyError() メンバー関数が呼び出されます。エラーの後で、続いて イベントが生成されることも、生成されないこともあります。イベント生成が終了した とき (エラーか、あるいはその他の理由から) には、NotifyStop() メンバー関数が 呼び出されます。NotifyError() のデフォルトの実装によって、 ユーザーにメッセージ・ボックスが用意され、ECLErr オブジェクトから取り出された エラー・メッセージのテキストが示されます。

何らかの理由 (エラー、または ECLOIA::UnregisterOIAEvent 呼び出し) で、 イベント通知が停止すると、NotifyStop() メンバー関数が呼び出されます。デフォルトの NotifyStop() の 実装は、何も実行しません。

ECLOIANotify メソッド

以下のセクションでは、ECLOIANotify クラスおよびそれから派生した、 すべてのクラスにおいて有効なメソッドについて説明します。

ECLOIANotify() ~ECLOIANotify() virtual void NotifyEvent(ECLOIA * OIAObj) = 0 virtual void NotifyError(ECLOIA * OIAObj, ECLErr ErrObj) virtual void NotifyStop(ECLOIA * OIAObj, int Reason)

NotifyEvent

このメソッドは、純粋仮想 メンバー関数です (**必ず**アプリケーションで、ECLOIANotify から派生したクラス内にこの関数を実装してください)。このメソッドは、OIA が更新され、 更新イベントを受信するためにこのオブジェクトが登録されるたびに、呼び出されます。

複数の OIA 更新は、単一のイベントに集約されることがあり、結果として、 このメソッドに対してはただ 1 つの呼び出しとなります。

プロトタイプ

virtual void NotifyEvent(ECLOIA * OIAObj) = 0

パラメーター

ECLOIA *

このイベントを生成した ECLOIA オブジェクトに対するポインター。

戻り値

なし

NotifyError

このメソッドは、イベントの生成時に ECLOIA オブジェクトがエラーを検出する たびに呼び出されます。エラー・オブジェクトには、そのエラーについての情報が含まれます (ECLErr クラスの説明を参照)。エラーの特性に応じて 、エラーの後で続いてイベントが生成されることがあります。エラーが原因でイベント生成が停止した 場合、NotifyStop() メソッドが呼び出されます。

アプリケーションは、この関数を実装するか、または ECLOIANotify の 基本クラスにこれを処理させるかを選ぶことができます。デフォルトの実装 は、ECLErr::GetMsgText() メソッドから提供される テキストを使用して、メッセージ・ボックスにエラーを表示します。アプリケーションが、 その派生クラス内にこの関数を実装すると、それによって この振る舞いがオーバーライドされます。

プロトタイプ

virtual void NotifyError(ECLOIA * OIAObj, ECLErr ErrObj)

パラメーター

ECLOIA *

このイベントを生成した ECLOIA オブジェクトに対するポインター。

ECLErr

エラーを記述する ECLErr オブジェクト。

戻り値

なし

NotifyStop

イベント生成が何らかの理由 (例えば、エラー条件、 または ECLOIA::UnregisterOIAEvent 呼び出し) で停止すると、このメソッドが呼び出されます。

現在、理由コード・パラメーターは未使用であり、ゼロになります。

この関数のデフォルトの実装では、何も実行しません。

プロトタイプ

virtual void NotifyStop(ECLOIA * OIAObj, int Reason)

パラメーター

ECLOIA *

このイベントを生成した ECLOIA オブジェクトに対するポインター。

整数

理由イベントの生成が停止しました (現在は未使用でゼロです)。

戻り値

なし

ECLPS クラス

ECLPS クラスは、ホストの表示スペースで処理を行います。

ECLPS オブジェクトは、作成時に識別された接続用に作成されます。ECLPS オブジェクトを 作成するには、通常は ECLConnection オブジェクトから取得される 接続名 (単一の A から Z の英字) または接続ハンドルを渡します。1 つの名前またはハンドルに対して、一度に 1 つの Z and I Emulator for Windows 接続しかオープンできません。

派生

ECLBase > ECLConnection > ECLPS

プロパティー

None

使用上の注意

ECLSession クラスは、このオブジェクトのインスタンスを作成します。アプリケーションで他のサービスを必要としない場合、 このオブジェクトを直接作成しても構いません。そうでない場合は、ECLSession オブジェクトを使用して、 必要なすべてのオブジェクトを作成することができます。

ECLPS メソッド

以下のセクションでは、ECLPS で使用可能なメソッドについて説明します。

ECLPS(char ConnName)
ECLPS(char ConnName)
ECLPS(long ConnHandle)
~ECLPS()
int GetPCCodePage()
int GetHostCodePage()
int GetOSCodePage()
void GetSize(ULONG *Rows, ULONG *Cols) ULONG GetSize()
ULONG GetSizeCols() ULONG GetSizeRows()
void GetCursorPos(ULONG *Row, ULONG *Col) ULONG GetCursorPos()
ULONG GetCursorPosRow()
ULONG GetCursorPosCol()
void SetCursorPos(ULONG pos),
void SetCursorPos(ULONG Row, ULONG Col)
void SendKeys(Char *text, ULONG AtPos),
void SendKeys(Char * text),
void SendKeys(Char *text, ULONG AtRow, ULONG AtCol)
ULONG SearchText(const char * const text, PS_DIR Dir=SrchForward,
BOOL FoldCase=FALSE)
ULONG SearchText(const char * const text,
ULONG StartPos, PS_DIR Dir=SrchForward, BOOL FoldCase=FALSE)
ULONG SearchText(const char char * const text, ULONG StartRow,
ULONG StartCol, PS_DIR Dir=SrchForward, BOOL FoldCase=FALSE)
ULONG GetScreen(char * Buff, ULONG BuffLen, PS_PLANE Plane=TextPlane)
ULONG GetScreen(char * Buff, ULONG BuffLen, ULONG StartPos,
ULONG Length, PS_PLANE Plane=TextPlane)
ULONG GetScreen(char * Buff, ULONG BuffLen, ULONG StartRow,
ULONG StartCol, ULONG Length, PS_PLANE Plane=TextPlane)
ULONG GetScreenRect(char * Buff, ULONG BuffLen, ULONG StartPos,
ULONG EndPos, PS_PLANE Plane=TextPlane)
ULONG StartCol, ULONG EndRow, ULONG EndCol,
ULONG GetScreenRect(char * Buff, ULONG BuffLen, ULONG StartRow,
ULONG StartCol, ULONG EndRow, ULONG EndCol,
PS_PLANE Plane=TextPlane)
void SetText(char *text);

void SetText(char *text, ULONG AtPos);
void SetText(char *text, ULONG AtRow, ULONG AtCol);
void CopyText ();
void CopyText (ULONG Long Len);
void CopyText (ULONG AtPos, ULONG Long Len);
void CopyText (ULONG AtRow, ULONG AtCol, ULONG Long Len);
void PasteText ();
void PasteText (ULONG Long Len);
void PasteText (ULONG AtPos, ULONG Long Len);
void PasteText (ULONG AtRow, ULONG AtCol, ULONG Long Len);
void ConvertPosToRowCol(ULONG pos, ULONG *row, ULONG *col)
ULONG ConvertRowColToPos(ULONG row, ULONG col)
ULONG ConvertPosToRow(ULONG Pos)
ULONG ConvertPosToCol(ULONG Pos)
void RegisterKeyEvent(ECLKeyNotify *NotifyObject)
virtual UnregisterKeyEvent(ECLKeyNotify *NotifyObject)
ECLFieldList *GetFieldList()
BOOL WaitForCursor(int Row, int Col, long nTimeOut=INFINITE,
BOOL bWaitForIR=TRUE)
BOOL WaitWhileCursor(int Row, int Col, long nTimeOut=INFINITE,
BOOL bWaitForIR=TRUE)
BOOL WaitForString(char* WaitString, int Row=0, int Col=0,
long nTimeOut=INFINITE, BOOL bWaitForIR=TRUE, BOOL bCaseSens=TRUE)
BOOL WaitWhileString(char* WaitString, int Row=0, int Col=0,
long nTimeOut=INFINITE, BOOL bWaitForIR=TRUE, BOOL bCaseSens=TRUE)
BOOL WaitForStringInRect(char* WaitString, int sRow, int sCol,
int eRow,int eCol, long nTimeOut=INFINITE,
BOOL bWaitForIR=TRUE, BOOL bCaseSens=TRUE)
BOOL WaitWhileStringInRect(char* WaitString, int sRow, int sCol,
int eRow,int eCol, long nTimeOut=INFINITE, BOOL bWaitForIR=TRUE,
BOOL bCaseSens=TRUE)
BOOL WaitForAttrib(int Row, int Col, unsigned char AttribDatum,
unsigned char MskDatum = 0xFF, PS_PLANE plane = FieldPlane,
long TimeOut = INFINITE, BOOL bWaitForIR = TRUE)
BOOL WaitWhileAttrib(int Row, int Col, unsigned char AttribDatum,
unsigned char MskDatum = 0xFF, PS_PLANE plane = FieldPlane,
long TimeOut = INFINITE, BOOL bWaitForIR = TRUE)
BOOL WaitForScreen(ECLScreenDesc* screenDesc, long TimeOut = INFINITE)
BOOL WaitWhileScreen(ECLScreenDesc* screenDesc, long TimeOut = INFINITE)
void RegisterPSEvent(ECLPSNotify * notify)

void RegisterPSEvent(ECLPSListener * listener)
void RegisterPSEvent(ECLPSListener * listener, int type)
void StartMacro(String MacroName)
void UnregisterPSEvent(ECLPSNotify * notify)
void UnregisterPSEvent(ECLPSListener * listener)
void UnregisterPSEvent(ECLPSListener * listener, int type)

次のメソッドは、ECLPS で有効です。

void SendKeys(WCHAR * text), void SendKeys(WCHAR *text, ULONG AtPos), void SendKeys(WCHAR *text, ULONG AtRow, ULONG AtCol) ULONG SearchText(const WCHAR * const text, PS_DIR Dir=SrchForward, BOOL FoldCase=FALSE) ULONG SearchText(const WCHAR * const text, ULONG StartPos, PS_DIR Dir=SrchForward, BOOL FoldCase=FALSE) ULONG SearchText(const WCHAR * const text, ULONG StartRow, ULONG StartCol, PS_DIR Dir=SrchForward, BOOL FoldCase=FALSE) ULONG GetScreen(WCHAR * Buff, ULONG BuffLen, PS_PLANE Plane=TextPlane) ULONG GetScreen(WCHAR * Buff, ULONG BuffLen, ULONG StartPos, ULONG Length, PS_PLANE Plane=TextPlane) ULONG GetScreen(WCHAR * Buff, ULONG BuffLen, ULONG StartRow, ULONG StartCol, ULONG Length, PS_PLANE Plane=TextPlane)

ECLPS コンストラクター

このメソッドは、接続名またはハンドルを 使用して ECLPS オブジェクトを作成します。

プロトタイプ

ECLPS(char ConnName) ECLPS(long ConnHandle)

パラメーター

char ConnName

1 文字の接続の短縮名 (A から Z、または a から z)。

long ConnHandle

ECL 接続のハンドル。

戻り値

なし

例

以下の例は、接続名を使用して ECLPS オブジェクトを作成する方法を示します。

```
//----------------------------------------------------------------------
// ECLPS::ECLPS              (Constructor)
//
```

```
// Build a PS object from a name, and another from a handle.
//---------------------------------------------------------------------
void Sample58() {

ECLPS *PS1, *PS2;       // Pointer to PS objects
ECLConnList ConnList;   // Connection list object

try {
  // Create PS object for connection 'A'
  PS1 = new ECLPS('A');

  // Create PS object for first connection in conn list
  PS2 = new ECLPS(ConnList.GetFirstConnection()->GetHandle());

  printf("PS #1 is for connection %c, PS #2 is for connection %c.\n",
         PS1->GetName(), PS2->GetName());
  delete PS1;
  delete PS2;
}
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}
} // end sample
```

ECLPS デストラクター

このメソッドは、ECLPS オブジェクトを破棄します。

プロトタイプ

~ECLPS()

パラメーター

なし

戻り値

なし

例

以下の例は、ECLPS オブジェクトを破棄する方法を示します。

```
ULONG  RowPos, ColPos;
ECLPS  *pPS;

try {
      pPS = new ECLPS('A');
      RowPos = pPS->ConvertPosToRow(544);
      ColPos = pPS->ConvertPosToCol(544);
      printf("PS position is at row %lu column %lu.",
```

```
      RowPos, ColPos);
      // Done with PS object so kill it
      delete pPS;
}
catch (ECLErr HE) {
  // Just report the error text in a message box
  MessageBox( NULL, HE.GetMsgText(), "Error!", MB_OK );
}
```

GetPCCodePage

GetPCCodePage メソッドは、パーソナル・コンピューターの コード・ページを指定する番号の検索を実施します。

プロトタイプ

int GetPCCodePage()

パラメーター

なし

戻り値

整数

コード・ページ番号

GetHostCodePage

GetHostCodePage メソッドは、ホスト・コンピューターの コード・ページを指定する番号の検索を実施します。

プロトタイプ

int GetHostCodePage()

パラメーター

なし

戻り値

整数

コード・ページ番号

GetOSCodePage

GetOSCodePage メソッドは、パーソナル・コンピューターのオペレーティング・システムの コード・ページを指定する番号の検索を実施します。

プロトタイプ

int GetOSCodePage()

パラメーター

なし

戻り値

整数

コード・ページ番号

GetSize

このメソッドは、ECLPS オブジェクトに関連した 接続用の表示スペースのサイズを 戻します。GetSize メソッドのシグニチャーは 2 つあります。ULONG GetSize() を使用すると、サイズは、線形値で戻され、表示スペース内の 文字の合計数を表します。void GetSize(ULONG *Rows, ULONG *Cols) を使用すると、 表示スペースの行および桁の数が戻されます。

プロトタイプ

ULONG GetSize()

void GetSize(ULONG *Rows, ULONG *Cols)

パラメーター

ULONG *Rows

この出力パラメーターは、表示スペース内の行数です。

ULONG *Cols

この出力パラメーターは、表示スペース内の桁数です。

戻り値

ULONG

線形値による表示スペースのサイズ。

例

以下に、GetSize メソッドの使用例を示します。

```
//---------------------------------------------------------------------
// ECLPS::GetSize
//
// Display dimensions of connection 'A'
//---------------------------------------------------------------------
void Sample59() {

ECLPS PS('A');       // PS object for connection A
ULONG Rows, Cols, Len;

PS.GetSize(&Rows, &Cols);   // Get num of rows and cols
// Could also write as:
Rows = PS.GetSizeRows();    // Redundant
Cols = PS.GetSizeCols();    // Redundant

Len = PS.GetSize();         // Get total size

printf("Connection A has %lu rows and %lu columns (%lu total length)\n",
       Rows, Cols, Len);

} // end sample
```

GetSizeRows

このメソッドは、ECLPS オブジェクトに関連した接続用の 表示スペース内の行数を戻します。

プロトタイプ

ULONG GetSizeRows()

パラメーター

なし

戻り値

ULONG

これは、表示スペース内の行数です。

例

以下に、GetSizeRows メソッドの使用例を示します。

```
//---------------------------------------------------------------------
// ECLPS::GetSizeRows
//
// Display dimensions of connection 'A'
```

```
//-------------------------------------------------------------------
void Sample59() {

ECLPS PS('A');       // PS object for connection A
ULONG Rows, Cols, Len;

PS.GetSize(&Rows, &Cols);   // Get num of rows and cols
// Could also write as:
Rows = PS.GetSizeRows();    // Redundant
Cols = PS.GetSizeCols();    // Redundant

Len = PS.GetSize();         // Get total size

printf("Connection A has %lu rows and %lu columns (%lu total length)\n",
       Rows, Cols, Len);

} // end sample
```

GetSizeCols

このメソッドは、ECLPS オブジェクトに関連した接続用の 表示スペース内の桁数を戻します。

プロトタイプ

ULONG GetSizeCols()

パラメーター

なし

戻り値

ULONG

これは、表示スペース内の桁数です。

例

以下に、GetSizeCols メソッドの使用例を示します。

```
//-------------------------------------------------------------------
// ECLPS::GetSizeCols
//
// Display dimensions of connection 'A'
//-------------------------------------------------------------------
void Sample59() {

ECLPS PS('A');       // PS object for connection A
ULONG Rows, Cols, Len;

PS.GetSize(&Rows, &Cols);   // Get num of rows and cols
// Could also write as:
Rows = PS.GetSizeRows();    // Redundant
```

```
Cols = PS.GetSizeCols();    // Redundant

Len = PS.GetSize();         // Get total size

printf("Connection A has %lu rows and %lu columns (%lu total length)\n",
       Rows, Cols, Len);

} // end sample
```

GetCursorPos

このメソッドは、ECLPS オブジェクトに 関連した接続用の表示スペース内のカーソル位置を 戻します。GetCursorPos メソッドには 2 種類のシグニチャーが あります。ULONG GetCursorPos() を使用すると、位置は線形位置 (1 がベース) で 戻されます。void GetCursorPos(ULONG *Row, ULONG * Col) を使用すると、 位置は行および桁の座標で戻されます。

プロトタイプ

ULONG GetCursorPos() void GetCursorPos(ULONG *Row, ULONG *Col)

パラメーター

ULONG *Row

この出力パラメーターは、ホスト・カーソルの行座標です。

ULONG *Col

この出力パラメーターは、ホスト・カーソルの桁座標です。

戻り値

ULONG

線形値で表されたカーソル位置。

例

以下に、GetCursorPos メソッドの使用例を示します。

```
//---------------------------------------------------------------------
// ECLPS::GetCursorPos
//
// Display position of host cursor in connection 'A'
//---------------------------------------------------------------------
void Sample60() {

ECLPS PS('A');       // PS object for connection A
ULONG Row, Col, Pos;

PS.GetCursorPos(&Row, &Col);   // Get row/col position
// Could also write as:
```

```
Row = PS.GetCursorPosRow();    // Redundant
Col = PS.GetCursorPosCol();    // Redundant

Pos = PS.GetCursorPos();       // Get linear position

printf("Host cursor of connection A is at row %lu column %lu
 (linear position %lu)\n", Row, Col, Pos);

} // end sample

/
```

GetCursorPosRow

このメソッドは、ECLPS オブジェクトに関連した接続用の 表示スペース内のカーソルの行位置を戻します。

プロトタイプ

ULONG GetCursorPosRow()

パラメーター

なし

戻り値

ULONG

これは、表示スペース内のカーソルの行位置です。

例

以下に、GetCursorPosRow メソッドの使用例を示します。

```
//---------------------------------------------------------------------
// ECLPS::GetCursorPosRow
//
// Display position of host cursor in connection 'A'
//---------------------------------------------------------------------
void Sample60() {

ECLPS PS('A');       // PS object for connection A
ULONG Row, Col, Pos;

PS.GetCursorPos(&Row, &Col);   // Get row/col position
// Could also write as:
Row = PS.GetCursorPosRow();    // Redundant
Col = PS.GetCursorPosCol();    // Redundant

Pos = PS.GetCursorPos();       // Get linear position

printf("Host cursor of connection A is at row %lu column %lu
 (linear position %lu)\n", Row, Col, Pos);
```

```
} // end sample
```

GetCursorPosCol

このメソッドは、ECLPS オブジェクトに関連した接続用の 表示スペース内のカーソルの桁位置を戻します。

プロトタイプ

ULONG GetCursorPosCol()

パラメーター

なし

戻り値

ULONG

これは、表示スペース内のカーソルの桁位置です。

例

以下に GetCursorPosCol メソッドの使用例を示します。

```
//---------------------------------------------------------------------
// ECLPS::GetCursorPosCol
//
// Display position of host cursor in connection 'A'
//---------------------------------------------------------------------
void Sample60() {

ECLPS PS('A');       // PS object for connection A
ULONG Row, Col, Pos;

PS.GetCursorPos(&Row, &Col);   // Get row/col position
// Could also write as:
Row = PS.GetCursorPosRow();    // Redundant
Col = PS.GetCursorPosCol();    // Redundant

Pos = PS.GetCursorPos();       // Get linear position

printf("Host cursor of connection A is at row %lu column %lu
 (linear position %lu)\n", Row, Col, Pos);

} // end sample

//---------------------------------------------------------------------
```

SetCursorPos

SetCursorPos メソッドは、 ECLPS オブジェクトに関連した接続用の表示スペース内のカーソルの 位置を設定します。SetCursorPos メソッドには 2 種類のシグニチャーがあります。位置は、void SetCursorPos(ULONG pos) を使用して線形位置 (1 がベース) で 指定することも、void SetCursorPos(ULONG Row, ULONG Col) を使用して行と桁の座標で指定することもできます。

プロトタイプ

void SetCursorPos(ULONG pos),

void SetCursorPos(ULONG Row, ULONG Col)

パラメーター

ULONG pos

線形位置によるカーソル位置。

ULONG Row

カーソルの行座標。

ULONG Col

カーソルの桁座標。

戻り値

なし

例

以下に SetCursorPos メソッドの使用例を示します。

```
--
// ECLPS::SetCursorPos
//
// Set host cursor to row 2 column 1.
//---------------------------------------------------------------------
void Sample61() {

ECLPS PS('A');          // PS object for connection A

PS.SetCursorPos(2, 1);  // Put cursor at row 2, column 1
printf("Cursor of connection A set to row 2 column 1.\n");

} // end sample

/
```

SendKeys

SendKeys メソッドは、ECLPS オブジェクトに関連した接続用の表示スペース内に、 キーのヌル終了ストリングを送ります。SendKeys メソッドには 3 種類のシグニチャーがあります。位置を指定しないと、現行ホスト・カーソル位置を先頭として キー・ストロークが入力されます。位置を指定することができ (線形または行/列座標で)、 その場合、ホスト・カーソルはまず指定位置に移動されます。

テキスト・ストリングにはプレーンなテキスト文字を含めることができ、 それは、そのまま表示スペースに書き込まれます。さらに、このストリングには、3270 Enter キーや 5250 PageUp キーなどの さまざまな制御キー・ストロークを表す組み込みキーワード (略号) を 含めることができます。キーワードは、大括弧で囲みます (例えば、[enter])。ストリング内でこのようなキーワードが検出されると、 適切なエミュレーター・コマンドに変換されてから送られます。テキスト・ストリングには、さまざまな平文の文字および組み込みキーワード を含めることができます。キーワードは左から右に向かって、最後にストリングの終わりに達するまで 処理されます。例えば、次に示すストリングの場合、文字 ABC は 現行カーソル位置に入力され、その後に 3270 EOF 消去 (Erase-end-of-field) キー・ストローク が続き、その後に 3270 Tab キー・ストローク、XYZ および PF1 キーと続きます。

```
ABC[eraseeof][tab]XYZ[pf1]
```

注: ストリング内のブランク文字は、他のすべてのプレーン・テキスト文字と同様、 ホストの表示スペースに書き込まれます。したがって、キーワードやテキストを区切るのにブランクを使用してはなりません。

左または右の大括弧文字をホストに送るには、 テキスト・ストリング内でそれを 2 度繰り返します (例えば、1 つのブラケットが 書き込まれるようにするには、ブラケットは 2 回出現する必要があります)。次に示す例の場合、ストリング「A [:]」が表示スペースに書き込まれます。

```
A[[:]]
```

画面上の保護位置にキー・ストロークを書き込もうとした場合、 キーボードはロックされ、残りのキー・ストロークは破棄されます。

キーワードのリストは、Sendkeys 略号キーワード (ページ 430)を参照してください。

プロトタイプ

void SendKeys(char * text), void SendKeys(char * text, ULONG AtPos), void SendKeys(char * text, ULONG AtRow, ULONG AtCol)

パラメーター

Char *text

表示スペースに送るキー・ストリング。

ULONG AtPos

キー・ストロークの書き込みの開始位置。

ULONG AtRow

キー・ストロークの書き込みの開始行。

ULONG AtCol

キー・ストロークの書き込みの開始桁。

戻り値

なし

例

以下に SendKeys メソッドの使用例を示します。

```
//-------------------------------------------------------------------
// ECLPS::SendKeys
//
// Sends a series of keystrokes, including 3270 function keys, to
// the host on connection A.
//-------------------------------------------------------------------
void Sample62() {

ECLPS PS('A');         // PS object for connection A

// The following key string will erase from the current cursor
// position to the end of the field, and then type the given
// characters into the field.
char  SendStr[] = "[eraseeof]ZIEWin is really cool";

// Note that an ECL error is thrown if we try to send keys to
// a protected field.

try {
  PS.SendKeys(SendStr);        // Do it at the current cursor position
  PS.SendKeys(SendStr, 3, 10); // Again at row 3 column 10
}
catch (ECLErr Err) {
  printf("Failed to send keys: %s\n", Err.GetMsgText());
}

} // end sample
```

SearchText

SearchText メソッドは、ECLPS オブジェクトに関連した接続の 表示スペースでテキストを検索します。このメソッドは、テキストが見つかった線形位置を戻しますが、 テキストが見つからない場合にはゼロを戻します。検索は、オプションの Dir パラメーターを使用して、 正方向 (左から右へ、上から下へ) または逆方向 (右から左へ、下から上へ) に行うことができます。検索には、オプションの FoldCase パラメーターを使用して、 大/小文字の区別を設定したり、 大文字変換 (大/小文字の区別なし) を指定したりできます。

開始位置を指定しないと、検索は画面の先頭から正方向に開始 されるか、または画面の終わりから逆方向に開始されます。開始位置は、線形位置または行と桁の座標を使用して指定できます。開始位置を指定した場合、それは、検

索を開始する位置を示します。正方向の検索では、開始位置 (その位置を含む) から画面の最後の文字までが検索されます。逆方向の検索では、開始位置 (その位置を含む) から画面の最初の文字までが検索されます。

検索を正常に完了するには、検索ストリングが検索域内に完全に 収まっていなければなりません (例えば、指定した開始位置をまたがって 検索ストリングが続く場合、そのストリングは見つかりません)。

基本クラス ConvertPosToRowCol メソッドを使用すると、 戻された線形位置を行と桁の座標に変換できます。

プロトタイプ

```
ULONG SearchText(const char * const text, PS_DIR Dir=SrchForward, BOOL FoldCase=FALSE) ULONG SearchText(const char * const text, ULONG StartPos, PS_DIR Dir=SrchForward, BOOL FoldCase=FALSE) ULONG SearchText(const char char * const text, ULONG StartRow, ULONG StartCol, PS_DIR Dir=SrchForward, BOOL FoldCase=FALSE)
```

パラメーター

char *text

検索するヌル終了ストリング。

PS_DIR Dir

検索方向を指示するオプションのパラメーター。指定する場合、**SrchForward** または **SrchBackward** のいずれかでなければなりません。デフォルト値は **SrchForward** です。

BOOL FoldCase

大/小文字を同一視して検索することを示すオプションのパラメーター。これを False と指定した場合、大文字と小文字の区別を含め、 テキスト・ストリングは表示スペースに完全に一致しなければなりません。True と指定した場合、大文字小文字に関係なくテキスト・ストリングが検索されます。デフォルトは FALSE です。

ULONG StartPos

検索を開始する線形位置を示します。この位置は、検索内に含まれます。

ULONG StartRow

検索を開始する行を指示します。

ULONG StartCol

検索を開始する桁を指示します。

戻り値

ULONG

見つかったストリングの線形位置。見つからなかった場合はゼロ。

例

以下に、SearchText メソッドの使用例を示します。

```
/---------------------------------------------------------------------
// ECLPS::SearchText
//
// Search for a string in various parts of the screen.
//---------------------------------------------------------------------
void Sample63() {

ECLPS PS('A');            // PS object
char  FindStr[] = "HCL";  // String to search for
ULONG LastOne;            // Position of search result

// Case insensative search of entire screen

printf("Searching for '%s'...\n", FindStr);
printf("  Anywhere, any case: ");
if (PS.SearchText(FindStr, TRUE) != 0)
  printf("Yes\n");
else
  printf("No\n");

// Backward, case sensative search on line 1

printf("  Line 1, exact match: ");
if (PS.SearchText(FindStr, 1, 80, SrchBackward) != 0)
  printf("Yes\n");
else
  printf("No\n");

// Backward, full screen search

LastOne = PS.SearchText(FindStr, SrchBackward, TRUE);
if (LastOne != 0)
  printf("  Last occurance on the screen is at row %lu, column %lu.\n",
         PS.ConvertPosToRow(LastOne), PS.ConvertPosToCol(LastOne));

} // end sample
```

GetScreen

このメソッドは、ECLPS オブジェクトに関連した接続の 表示スペースからデータを取り出します。データは、表示スペースの文字位置あたり 1 バイトずつの、 バイト値の線形配列で戻されます。データが TextPlane から取り出される (その場合、単一のヌル終了バイトが 付加されます) 場合を除き、配列はヌル終了しません。

アプリケーションは、戻されるデータ用のバッファーとそのバッファーの長さ を指定しなければなりません。要求データは、バッファーに収容しきれない場合、切り捨てられ ます。TextPlane データの場合、バッファーには、終了ヌル用に 少なくとも 1 バイトが余分に含まれていなければなりません。このメソッドは、アプリケーション・バッファーにコピーされたバイト数 を戻します (TextPlane コピーでの終了ヌルは含まれません)。

アプリケーションは、表示スペースから取り出すデータのバイト数を 指定しなければなりません。開始位置にその長さを加えたものが表示スペースのサイズを超える場合、 エラーになります。データは、指定された開始位置を先頭にして戻されますが、 開始位置を指定しないと、行 1 桁 1 に戻されます。戻されたデータは、左から右および上から下へと複数行にわたって指定された長さまで、線形方式で表示スペースからコピーされます。アプリケーションが画面の長方形領域の画面データを取得 したい場合、GetScreenRect メソッドを使用してください。

アプリケーションは、データの取り出しの対象に任意のプレーンを指定することができます。プレーンを指定しない場合は、TextPlane が取り出されます。その他の ECL プレーンの詳細については、ECL プレーン ー 形式および内容 (ページ 433)を参照してください。

プロトタイプ

```
ULONG GetScreen(char * Buff, ULONG BuffLen, PS_PLANE Plane=TextPlane) ULONG GetScreen(char * Buff,
ULONG BuffLen, ULONG StartPos, ULONG Length, PS_PLANE Plane=TextPlane) ULONG GetScreen(char * Buff,
ULONG BuffLen, ULONG StartRow, ULONG StartCol, ULONG Length, PS_PLANE Plane=TextPlane)
```

パラメーター

char *Buff

BuffLen サイズ以上のアプリケーション提供のバッファーを指すポインター。

ULONG BuffLen

提供されるバッファーのバイト数。

ULONG StartPos

コピーを開始する表示スペース内の線形位置。

ULONG StartRow

コピーを開始する表示スペース内の行。

ULONG StartCol

コピーを開始する表示スペース内の桁。

ULONG Length

表示スペースからコピーする線形バイト数。

PS_PLANE plane

どの表示スペースのプレーンをコピーするかを指定する オプショナル・パラメーター。指定する場合、**TextPlane**、**ColorPlane**、**FieldPlane**、**ExfieldPlane** のいずれかでなければなりません。デフォルト値は **TextPlane** です。その他の ECL プレーンの内容および形式に ついては、ECL プレーン ー 形式および内容 (ページ 433)を参照してください。

戻り値

ULONG

表示スペースからコピーするデータ・バイト数。この値には、TextPlane コピー用の後続ヌル・バイトは含まれません。

例

以下に、GetScreen メソッドの使用例を示します。

```
//-------------------------------------------------------------------
// ECLPS::GetScreen
//
// Get text and other planes of data from the presentation space.
//-------------------------------------------------------------------
void Sample64() {

ECLPS PS('A');            // PS object
char  *Text;              // Text plane data
char  *Field;             // Field plane data
ULONG Len;                // Size of PS

Len = PS.GetSize();

// Note text buffer needs extra byte for null terminator

Text  = new char[Len + 1];
Field = new char[Len];

PS.GetScreen(Text, Len+1);                // Get entire screen (text)
PS.GetScreen(Field, Len, FieldPlane);     // Get entire field plane
PS.GetScreen(Text, Len+1, 1, 1, 80);      // Get line 1 of text

printf("Line 1 of the screen is:\n%s\n", Text);

delete []Text;
delete []Field;

} // end sample
```

GetScreenRect

このメソッドは、ECLPS オブジェクトに関連した接続の 表示スペースからデータを取り出します。データは、表示スペースの文字位置あたり 1 バイトずつの、 バイト値の線形配列で戻されます。この配列はヌル終了ではありません。

アプリケーションは、表示スペース内の開始および終了座標を提供します。これらの座標は、長方形内の互いに反対側の角の位置を示します。長方形内の表示スペースは、単一の線形配列として アプリケーション・バッファーにコピーされます。開始点と終了点は、 空間内で任意の相関関係にすることができます。コピーは、上端の点を含んだ行から 下端の点を含んだ行へ向かって、また、左端の桁から右端の桁に向かって開始するよう 定義されます。2 つ

の座標はともに、表示スペースのサイズ境界内に なければなりません。そうでないと、エラーになります。この座標は、 線形位置で指定しても、行番号と桁番号で指定しても構いません。

提供するアプリケーション・バッファーは、最低限、長方形内のバイト数 を含めるのに十分な大きさでなければなりません。バッファーが小さすぎると、 データはコピーされず、メソッドの結果としてゼロが戻されます。小さすぎなければ、メソッドから、コピーされたバイト数が戻されます。

アプリケーションは、データの取り出しの対象に任意のプレーンを指定することができます。プレーンを指定しない場合は、TextPlane が取り出されます。その他の ECL プレーンの詳細については、ECL プレーン – 形式および内容 (ページ 433)を参照してください。

プロトタイプ

ULONG GetScreenRect(char * Buff, ULONG BuffLen, ULONG StartPos, ULONG EndPos, PS_PLANE Plane=TextPlane)
ULONG GetScreenRect(char * Buff, ULONG BuffLen, ULONG StartRow, ULONG StartCol, ULONG EndRow, ULONG EndCol, PS_PLANE Plane=TextPlane)

パラメーター

char *Buff

BuffLen サイズ以上のアプリケーション提供のバッファーを指すポインター。

ULONG BuffLen

提供されるバッファーのバイト数。

ULONG StartPos

コピー長方形の対角線上の他方の角の表示スペース内の線形位置。

ULONG EndPos

コピー長方形の対角線上の他方の角の表示スペース内の線形位置。

ULONG StartRow

コピー長方形の対角線上の他方の角の表示スペース内の行。

ULONG StartCol

コピー長方形の対角線上の他方の角の表示スペース内の桁。

ULONG EndRow

コピー長方形の対角線上の他方の角の表示スペース内の行。

ULONG EndCol

コピー長方形の対角線上の他方の角の表示スペース内の桁。

PS_PLANE plane

どの表示スペースのプレーンをコピーするかを指定する オプショナル・パラメーター。指定する場合、**TextPlane**、**ColorPlane**、**FieldPlane**、**ExfieldPlane** のいずれかでなければなりません。デフォル

ト値は **TextPlane** です。その他の ECL プレーンの内容および形式に ついては、ECL プレーン ー 形式および内容 (ページ 433)を参照してください。

戻り値

ULONG

表示スペースからコピーするデータ・バイト数。

例

以下に GetScreenRect メソッドの使用例を示します。

```
------------------------------
// ECLPS::GetScreenRect
//
// Get rectangular parts of the host screen.
//--------------------------------------------------------------------
void Sample66() {

ECLPS PS('A');         // PS object for connection A
char Buff[4000];       // Big buffer

// Get first 2 lines of the screen text
PS.GetScreenRect(Buff, sizeof(Buff), 1, 1, 2, 80);

// Get last 2 lines of the screen
PS.GetScreenRect(Buff, sizeof(Buff),
                 PS.GetSizeRows()-1,
                 1,
                 PS.GetSizeRows(),
                 PS.GetSizeCols());

// Get just a part of the screen (VM main menu calendar)
PS.GetScreenRect(Buff, sizeof(Buff),
                 5,  51,
                 13, 76);

// Same as previous (specify any 2 oposite corners of the rectangle)
PS.GetScreenRect(Buff, sizeof(Buff),
                 13, 51,
                 5,  76);

// Note results are placed in buffer end-to-end with no line delimiters
printf("Contents of rectangular screen area:\n%s\n", Buff);

} // end sample
```

SetText

SetText メソッドは、ECLPS オブジェクトに関連した接続用の 表示スペース内に文字配列を送ります。これは SendKeys メソッドと似ていますが、 略号キー・ストローク (例えば、[enter] または [pf1]) は送らない点で異なります。

位置の指定がない場合、テキストは現在のカーソル位置から書き込まれます。

プロトタイプ

void SetText(char *text);

void SetText(char *text, ULONG AtPos);

void SetText(char *text, ULONG AtRow, ULONG AtCol);

パラメーター

char *text

表示スペースにコピーする文字のヌル終了ストリング。

ULONG AtPos

コピーを開始する表示スペース内の線形位置。

ULONG AtRow

コピーを開始する表示スペース内の行。

ULONG AtCol

コピーを開始する表示スペース内の桁。

戻り値

なし

例

以下に SetText メソッドの使用例を示します。

```
//---------------------------------------------------------------------
// ECLPS::SetText
//
// Update various input fields of the screen.
//---------------------------------------------------------------------
void Sample65() {

ECLPS PS('A');         // PS object for connection A

// Note that an ECL error is thrown if we try to write to
// a protected field.

try {
  // Update first 2 input fields of the screen.  Note
  // fields are not erased before update.
  PS.SendKeys("[home]");
  PS.SetText("Field 1");
  PS.SendKeys("[tab]");
  PS.SetText("Field 2");
  // Note: Above 4 lines could also be written as:
  // PS.SendKeys("[home]Field 1[tab]Field 2");
```

```
  // But SetText() is faster, esp for long strings
}
catch (ECLErr Err) {
  printf("Failed to send keys: %s\n", Err.GetMsgText());
}

} // end sample

//-------------------------------------
```

CopyText

このメソッドは、指定された長さの表示スペース内の所定の場所からクリップボードにテキストをコピーします。コピーされるテキストの長さは指定された長さになります。 長さを指定しない場合、表示スペースの終わりまでのテキストがコピーされます。場所を指定しない場合、テキストは表示スペース内の現行カーソル位置からコピーされます。パラメーターを指定しないと、表示スペース全体がクリップボードにコピーされます。

プロトタイプ

void CopyText ();

void CopyText (ULONG Long Len);

void CopyText (ULONG AtPos, ULONG Long Len);

void CopyText (ULONG AtRow, ULONG AtCol, ULONG Long Len);

パラメーター

ULONG Long Len

表示スペースからコピーする線形バイト数。

ULONG AtPos

コピーを開始する表示スペース内の線形位置。

ULONG AtRow

コピーを開始する表示スペース内の行。

ULONG AtCol

コピーを開始する表示スペース内の桁。

戻り値

なし

例

以下に CopyText メソッドの使用例を示します。

```
//---------------------------------------------------------------------
// ECLPS::CopyText
//
// Copy text from Presentation Space to clipboard.
//---------------------------------------------------------------------
void Sample126() {

ECLPS PS('A');         // PS object for connection A
long row, col, length2copy;

// Note that an ECL error is thrown if we try to write to
// a protected field.
try {
   printf("Please enter the position and length to copy from PS [row col length2copy] \n");

   scanf("%ld %ld %ld", &row, &col, &length2copy);
  PS.CopyText(row, col, length2copy);
  }
catch (ECLErr Err) {
   printf("Failed to copy text: %s\n", Err.GetMsgText());
}
} // end sample
//-------------------------------------
```

PasteText

このメソッドは、指定された長さのテキストをクリップボードから表示スペースの指定の場所に貼り付けます。貼り付けられるテキストの長さは、指定された長さになります。長さを指定しない場合、クリップボードのテキスト全体が、表示スペースの最後に達するまで貼り付けられます。ロケーションが指定されていない場合、テキストは、表示スペースの現在のカーソル位置に貼り付けられます。表示スペースがフィールド形式であり、クリップボードの内容の貼り付け中にタブ文字 '\t' が指定されている場合、残りの貼り付け内容は次の書き込み可能フィールドに移動します。

プロトタイプ

void PasteText ();

void PasteText (ULONG Long Len);

void PasteText (ULONG AtPos, ULONG Long Len);

void PasteText (ULONG AtRow, ULONG AtCol, ULONG Long Len);

パラメーター

ULONG Long Len

表示スペースから貼り付ける線形バイト数。

ULONG AtPos

貼り付けを開始する表示スペース内の線形位置。

ULONG AtRow

貼り付けを開始する表示スペース内の行。

ULONG AtCol

貼り付けを開始する表示スペース内の桁。

戻り値

なし

例

以下に PasteText メソッドの使用例を示します。

```
//-------------------------------------------------------------------
// ECLPS::PasteText
//
// Paste text to Presentation Space from clipboard.
//-------------------------------------------------------------------
void Sample127() {

ECLPS PS('A');         // PS object for connection A
long row, col, length2paste;

// Note that an ECL error is thrown if we try to write to
// a protected field.
try {
   printf("Please enter the position and length to paste from clipboard [row col length2paste] \n");
   scanf("%ld %ld %ld", &row, &col, &length2paste);
  PS.PasteText(row, col, length2paste);
}
catch (ECLErr Err) {
  printf("Failed to paste text: %s\n", Err.GetMsgText());
}
} // end sample
//-----------------------------------------------
```

ConvertPosToRowCol

ConvertPosToRowCol メソッドは、 線形配列として表された表示スペース内の位置を、 行と桁の座標で示される表示スペース内の位置に変換します。変換後の位置は、ECLPS オブジェクトに関連した 接続用の表示スペース内にあります。

プロトタイプ

void ConvertPosToRowCol(ULONG pos, ULONG *row, ULONG *col)

パラメーター

ULONG pos

線形配列として示された、表示スペース内の変換する位置。

ULONG *row

表示スペース内の変換後の行座標。

ULONG *col

表示スペース内の変換後の桁座標。

戻り値

なし

例

以下の例は、線形配列として表された表示スペース内の位置を、 行と桁の座標で示される表示スペース内の位置に変換する方法を示します。

```
///-------------------------------------------------------------------
// ECLPS::ConvertPosToRowCol
//
// Find a string in the presentation space and display the row/column
// coordinate of its location.
//-------------------------------------------------------------------
void Sample67() {

ECLPS PS('A');            // PS Object
ULONG FoundPos;           // Linear position
ULONG FoundRow,FoundCol;


FoundPos = PS.SearchText("HCL", TRUE);
if (FoundPos != 0) {
  PS.ConvertPosToRowCol(FoundPos, &FoundRow, &FoundCol);
  // Another way to do the same thing:
  FoundRow = PS.ConvertPosToRow(FoundPos);
  FoundCol = PS.ConvertPosToCol(FoundPos);

  printf("String found at row %lu column %lu (position %lu)\n",
         FoundRow, FoundCol, FoundPos);
}
else printf("String not found.\n");

} // end sample
```

ConvertRowColToPos

ConvertRowColToPos メソッドは、 行と桁の座標で示された表示スペース内の位置を、 線形配列として表される表示スペース内の位置に変換します。変換後の位置は、ECLPS オブジェクトに関連した 接続用の表示スペース内にあります。

プロトタイプ

ULONG ConvertRowColToPos(ULONG row, ULONG col)

パラメーター

ULONG row

表示スペース内の変換する行の座標。

ULONG col

表示スペース内の変換する桁の座標。

戻り値

ULONG

線形配列として示される表示スペース内の変換後の位置。

例

以下の例は、行と桁の座標で示された表示スペース内の位置を、 線形配桁位置に変換する方法を示します。

```
///----------------------------------------------------------------------
// ECLPS::ConvertRowColToPos
//
// Find a string in the presentation space and display the row/column
// coordinate of its location.
//----------------------------------------------------------------------
void Sample67() {

ECLPS PS('A');            // PS Object
ULONG FoundPos;           // Linear position
ULONG FoundRow,FoundCol;


FoundPos = PS.SearchText("HCL", TRUE);
if (FoundPos != 0) {
  PS.ConvertPosToRowCol(FoundPos, &FoundRow, &FoundCol);
  // Another way to do the same thing:
  FoundRow = PS.ConvertPosToRow(FoundPos);
  FoundCol = PS.ConvertPosToCol(FoundPos);

  printf("String found at row %lu column %lu (position %lu)\n",
         FoundRow, FoundCol, FoundPos);
}
else printf("String not found.\n");
```

```
} // end sample
```

ConvertPosToRow

このメソッドは表示スペース内の線形位置値を 取り出し、ECLPS オブジェクトに関連した接続について、その値が存在している行を戻します。

プロトタイプ

ULONG ConvertPosToRow(ULONG Pos)

パラメーター

ULONG Pos

これは、表示スペース内の変換する線形位置です。

戻り値

ULONG

これは、線形位置に対する行位置です。

例

以下に、ConvertPosToRow メソッドの使用例を示します。

```
///--------------------------------------------------------------------
// ECLPS::ConvertPosToRow
//
// Find a string in the presentation space and display the row/column
// coordinate of its location.
//--------------------------------------------------------------------
void Sample67() {

ECLPS PS('A');            // PS Object
ULONG FoundPos;           // Linear position
ULONG FoundRow,FoundCol;


FoundPos = PS.SearchText("HCL", TRUE);
if (FoundPos != 0) {
  PS.ConvertPosToRowCol(FoundPos, &FoundRow, &FoundCol);
  // Another way to do the same thing:
  FoundRow = PS.ConvertPosToRow(FoundPos);
  FoundCol = PS.ConvertPosToCol(FoundPos);

  printf("String found at row %lu column %lu (position %lu)\n",
         FoundRow, FoundCol, FoundPos);
}
else printf("String not found.\n");
```

```
} // end sample
```

ConvertPosToCol

このメソッドは表示スペース内の線形位置値を取り出し、ECLPS オブジェクトに 関連した接続について、その値が存在している桁を戻します。

プロトタイプ

ULONG ConvertPosToCol(ULONG Pos)

パラメーター

ULONG Pos

これは、表示スペース内の変換する線形位置です。

戻り値

ULONG

これは、線形位置に対する桁位置です。

例

以下に、ConvertPosToCol メソッドの使用例を示します。

```
///--------------------------------------------------------------------
/// ECLPS::ConvertPosToCol
//
// Find a string in the presentation space and display the row/column
// coordinate of its location.
//--------------------------------------------------------------------
void Sample67() {

ECLPS PS('A');            // PS Object
ULONG FoundPos;           // Linear position
ULONG FoundRow,FoundCol;


FoundPos = PS.SearchText("HCL", TRUE);
if (FoundPos != 0) {
  PS.ConvertPosToRowCol(FoundPos, &FoundRow, &FoundCol);
  // Another way to do the same thing:
  FoundRow = PS.ConvertPosToRow(FoundPos);
  FoundCol = PS.ConvertPosToCol(FoundPos);

  printf("String found at row %lu column %lu (position %lu)\n",
         FoundRow, FoundCol, FoundPos);
}
else printf("String not found.\n");
```

```
} // end sample
```

RegisterKeyEvent

RegisterKeyEvent 関数は、 オペレーター・キー・ストローク・イベントの通知を受け取るため、 アプリケーション提供のオブジェクトを登録します。アプリケーションは、ECLKeyNotify 抽象基本クラスから派生した オブジェクトを作成しなければなりません。オペレーター・キー・ストロークが発生すると、アプリケーション提供 オブジェクトの NotifyEvent() メソッドが呼び出されます。アプリケーションは、選択によってキー・ストロークをフィルターに掛けることも、 または通常の方法で渡して処理することもできます。詳しくは、ECLKeyNotify クラス (ページ 87)を参照してください。

実装上の制約事項: キー・ストローク・イベントを受け取るためのオブジェクトは一度に 1 つしか登録できません。

プロトタイプ

void RegisterKeyEvent(ECLKeyNotify *NotifyObject)

パラメーター

ECLKeyNotify *NotifyObject

ECLKeyNotify クラスから派生したアプリケーション・オブジェクト。

戻り値

なし

例

以下の例は、オペレーター・キー・ストローク・イベントの通知を受け取るために アプリケーション提供のオブジェクトを登録する方法を示します。RegisterKeyEvent の例については、『ECLKeyNotify クラス (ページ 87)』を参照してください。

```
// This is the declaration of your class derived from ECLKeyNotify....
class MyKeyNotify: public ECLKeyNotify
{
public:
  // App can put parms on constructors if needed
  MyKeyNotify();          // Constructor
  MyKeyNotify();         // Destructor

  // App must define the NotifyEvent method
  int NotifyEvent(char KeyType[2], char KeyString[7]);  // Keystroke callback

private:
  // Whatever you like...
};
// this is the implementation of app methods...

int MyKeyNotify::NotifyEvent( ECLPS *, char *KeyType, char *Keystring )
```

```
{
  if (...) {
    ...
    return 0;  // Remove keystroke (filter)
  }
  else
    ...
    return 1;  // Pass keystroke to emulator as usual
  }
}

// this would be the code in say, WinMain...

ECLPS *pPS;            // Pointer to ECLPS object
MyKeyNotify *MyKeyNotifyObject; // My key notification object,derived
                                // from ECLKeyNotify

try {
  pPS = new ECLPS('A');              // Create PS object for 'A' session

  // Register for keystroke events
  MyKeyNotifyObject = new MyKeyNotify();
  pPS->RegisterKeyEvent(MyKeyNotifyObject);

  // After this, MyKeyNotifyObject->NotifyEvent() will be called
  // for each operator keystroke...
}
catch (ECLErr HE) {
  // Just report the error text in a message box
  MessageBox( NULL, HE.GetMsgText(), "Error!", MB_OK );
}
```

UnregisterKeyEvent

UnregisterKeyEvent メソッドは、 以前に RegisterKeyEvent 関数を使用してキー・ストローク・イベント用に登録されている アプリケーション・オブジェクトの登録を抹消します。登録済みのアプリケーション通知オブジェクトの場合、 先にこの関数を呼び出してその登録を抹消しないかぎり、オブジェクトを破棄してはなりません。現在登録されている通知オブジェクトがない場合や、 登録済みオブジェクトが渡された NotifyObject でない場合、 この関数は何も実行しません (エラーになりません)。

プロトタイプ

virtual UnregisterKeyEvent(ECLKeyNotify *NotifyObject)

パラメーター

ECLKeyNotify *NotifyObject

現在キー・ストローク・イベント用に登録されているオブジェクト。

戻り値

なし

例

UnregisterKeyEvent の例については、『ECLKeyNotify クラス (ページ 87)』を参照してください。

GetFieldList

このメソッドは、ECLFieldList オブジェクト を指すポインターを戻します。フィールド・リスト・オブジェクトを使用すると、 ホストの表示スペース内のフィールド・リストを反復できます。この関数によって戻された ECLFieldList オブジェクト は、ECLPS オブジェクトが破棄されると自動的に破棄されます。このオブジェクトの詳細については、『ECLFieldList クラス (ページ 80)』を参照してください。

プロトタイプ

ECLFieldList *GetFieldList()

パラメーター

なし

戻り値

ECLFieldList *

ECLFieldList オブジェクトを指すポインター。

例

以下の例は、ECLFieldList オブジェクトを指すポインター がどのように戻されるかを示します。

```
// ECLPS::GetFieldList
//
// Display number of fields on the screen.
//--------------------------------------------------------------------
void Sample68() {

ECLPS        *PS;           // Pointer to PS object
ECLFieldList *FieldList;    // Pointer to field list object

try {
  PS = new ECLPS('A');               // Create PS object for 'A'

  FieldList = PS->GetFieldList();    // Get pointer to field list
  FieldList->Refresh();              // Build the field list

  printf("There are %lu fields on the screen of connection %c.\n",
    FieldList->GetFieldCount(), PS->GetName());
```

```
  delete PS;
}
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}
} // end sample
```

WaitForCursor

WaitForCursor メソッドは、指定された位置に配置する ECLPS オブジェクトに関連した 接続の表示スペースでカーソルを待ちます。

プロトタイプ

BOOL WaitForCursor(int Row, int Col, long nTimeOut=INFINITE, BOOL bWaitForIR=TRUE)

パラメーター

int Row

カーソルの行の位置。負数の場合は、この値は PS の下部からの行位置を示します。

int Col

カーソルの列の位置。負数の場合は、この値は PS の端からの桁位置を示します。

long nTimeOut

待機する時間の最大長 (ミリ秒)。このパラメーターはオプションです。デフォルトは、Infinite (無期限) です。

BOOL bWaitForIR

この値が True である場合、待ち条件の基準を満たすと、関数は OIA が PS の入力受け入れ準備完了 を示すまで待機します。このパラメーターはオプションで、デフォルト値は TRUE です。

戻り値

メソッドは、条件が合致していれば True を 戻し、nTimeOut (ミリ秒) が経過した場合は False を戻します。

注: テスト条件が FALSE を戻すときに、nTimeOut がデフォルト値 (INFINITE) である場合は、 このメソッドはブロックされます。

例

```
// set up PS
ECLPS ps = new ECLPS('A');

// do the wait
int TimeOut = 5000;
```

```
BOOL waitOK = ps.WaitForCursor(23,1,TimeOut, TRUE);

// do the processing for the screen
```

WaitWhileCursor

WaitWhileCursor メソッドは、ECLPS オブジェクトに関連した接続の 表示スペースでカーソルが指定された位置に配置されている間待機します。

プロトタイプ

BOOL WaitWhileCursor(int Row, int Col, long nTimeOut=INFINITE, BOOL bWaitForIR=TRUE)

パラメーター

int Row

カーソルの行の位置。負数の場合は、この値は PS の下部からの行位置を示します。

int Col

カーソルの列の位置。負数の場合は、この値は PS の端からの桁位置を示します。

long nTimeOut

待機する時間の最大長 (ミリ秒)。このパラメーターはオプションです。デフォルトは、Infinite (無期限) です。

BOOL bWaitForIR

この値が True である場合、待ち条件の基準を満たすと、関数は OIA が PS の入力受け入れ準備完了 を示すまで待機します。このパラメーターはオプションで、デフォルト値は TRUE です。

戻り値

メソッドは、条件が合致していれば True を 戻し、nTimeOut (ミリ秒) が経過した場合は False を戻します。

注: テスト条件が FALSE を戻すときに、nTimeOut がデフォルト値 (INFINITE) である場合は、 このメソッドはブロックされます。

例

```
// set up PS
ECLPS ps = new ECLPS('A');

// do the wait
int TimeOut = 5000;
BOOL waitOK = ps.WaitWhileCursor(23,1,TimeOut, TRUE);

// do the processing for when the screen goes away
```

WaitForString

WaitForString メソッドは、ECLPS オブジェクトに関連した接続の表示スペース で、指定されたストリングが現れるのを待ちます。オプションの行パラメーターおよび桁パラメーターが 使用される場合は、ストリングは指定された位置から開始しなければなりません。行と桁に、それぞれ 0 が渡された場合は、メソッドは PS 全体を探索します。

プロトタイプ

```
BOOL WaitForString( char* WaitString, int Row=0, int Col=0, long nTimeOut=INFINITE, BOOL bWaitForIR=TRUE,
BOOL bCaseSens=TRUE)
```

パラメーター

char* WaitString

待機の対象となるストリング。

int Row

カーソルの行の位置。負数の場合は、この値は PS の下部からの行位置を示します。デフォルトはゼロです。

int Col

カーソルの列の位置。負数の場合は、この値は PS の端からの桁位置を示します。デフォルトはゼロです。

long nTimeOut

待機する時間の最大長 (ミリ秒)。このパラメーターはオプションです。デフォルトは、Infinite (無期限) です。

BOOL bWaitForIR

この値が True である場合、待ち条件の基準を満たすと、関数は OIA が PS の入力受け入れ準備完了 を示すまで待機します。このパラメーターはオプションで、デフォルト値は TRUE です。

BOOL bCaseSens

この値が True の場合は、待ち条件は大/小文字の区別があるものとして検査されます。このパラメーターはオプションです。デフォルトは TRUE です。

戻り値

メソッドは、条件が合致していれば True を 戻し、nTimeOut (ミリ秒) が経過した場合は False を戻します。

注: テスト条件が FALSE を戻すときに、nTimeOut がデフォルト値 (INFINITE) である場合は、 このメソッドはブロックされます。

例

```
// set up PS
ECLPS ps = new ECLPS('A');

// do the wait
BOOL waitOK = ps.WaitForString("LOGON");

// do the processing for the screen
```

WaitWhileString

WaitWhileString メソッドは、指定されたストリングが ECLPS オブジェクトに関連した接続の 表示スペースにある間待機します。オプションの行パラメーターおよび桁パラメーターが 使用される場合は、ストリングは指定された位置から開始しなければなりません。行と桁に、それぞれ 0 が渡された場合は、メソッドは PS 全体を探索します。

プロトタイプ

```
BOOL WaitWhileString(char* WaitString, int Row=0, int Col=0, long nTimeOut=INFINITE, BOOL bWaitForIR=TRUE,
BOOL bCaseSens=TRUE)
```

パラメーター

char* WaitString

待機の対象となるストリング。

int Row

ストリングを開始する行位置。負数の場合は、この値は PS の下部からの行位置を示します。デフォルトはゼロです。

int Col

ストリングを開始する桁位置。負数の場合は、この値は PS の端からの桁位置を示します。デフォルトはゼロです。

long nTimeOut

待機する時間の最大長 (ミリ秒)。このパラメーターはオプションです。デフォルトは、Infinite (無期限) です。

BOOL bWaitForIR

この値が True である場合、待ち条件の基準を満たすと、関数は OIA が PS の入力受け入れ準備完了 を示すまで待機します。このパラメーターはオプションで、デフォルト値は TRUE です。

BOOL bCaseSens

この値が True の場合は、待ち条件は大/小文字の区別があるものとして検査されます。このパラメーターはオプションです。デフォルトは TRUE です。

戻り値

メソッドは、条件が合致していれば True を 戻し、nTimeOut (ミリ秒) が経過した場合は False を戻します。

注: テスト条件が FALSE を戻すときに、nTimeOut がデフォルト値 (INFINITE) である場合は、 このメソッドはブロックされます。

例

```
// set up PS
ECLPS ps = new ECLPS('A');

// do the wait
BOOL waitOK = ps.WaitWhileString("LOGON");

// do the processing for when the screen goes away
```

WaitForStringInRect

WaitForStringInRect メソッドは、指定された長方形における ECLPS オブジェクトに関連した 接続の表示スペースで、指定されたストリングが現れるのを待ちます。

プロトタイプ

```
BOOL WaitForStringInRect(char* WaitString, int sRow, int sCol, int eRow,int eCol, long nTimeOut=INFINITE, BOOL bWaitForIR=TRUE, BOOL bCaseSens=TRUE)
```

パラメーター

char* WaitString

待機の対象となるストリング。

int Row

長方形を開始する行位置。

int Col

長方形を開始する桁位置。

int eRow

長方形探索を終了する行位置。

int eCol

長方形探索を終了する桁位置。

long nTimeOut

待機する時間の最大長 (ミリ秒)。このパラメーターはオプションです。デフォルトは、Infinite (無期限) です。

BOOL bWaitForIR

この値が True である場合、待ち条件の基準を満たすと、関数は OIA が PS の入力受け入れ準備完了 を示すまで待機します。このパラメーターはオプションで、デフォルト値は TRUE です。

BOOL bCaseSens

この値が True の場合は、待ち条件は大/小文字の区別があるものとして検査されます。このパラメーターはオプションです。デフォルトは TRUE です。

戻り値

メソッドは、条件が合致していれば True を 戻し、nTimeOut (ミリ秒) が経過した場合は False を戻します。

注: テスト条件が FALSE を戻すときに、nTimeOut がデフォルト値 (INFINITE) である場合は、 このメソッドはブロックされます。

例

```
// set up PS
ECLPS ps = new ECLPS('A');

// do the wait
BOOL waitOK = ps.WaitForStringInRect("LOGON",1,1,23,80);

// do the processing for the screen
```

WaitWhileStringInRect

WaitWhileStringInRect メソッドは、指定されたストリングが指定長方形内の ECLPS オブジェクト に関連した接続の表示スペースにある間待機します。

プロトタイプ

```
BOOL WaitWhileStringInRect(char* WaitString, int sRow, int sCol, int eRow,int eCol, long nTimeOut=INFINITE,
BOOL bWaitForIR=TRUE, BOOL bCaseSens=TRUE)
```

パラメーター

char* WaitString

待機の対象となるストリング。

int Row

長方形を開始する行位置。

int Col

長方形を開始する桁位置。

int eRow

長方形探索を終了する行位置。

int eCol

長方形探索を終了する桁位置。

long nTimeOut

待機する時間の最大長 (ミリ秒)。このパラメーターはオプションです。デフォルトは、Infinite (無期限) です。

BOOL bWaitForIR

この値が True である場合、待ち条件の基準を満たすと、関数は OIA が PS の入力受け入れ準備完了 を示すまで待機します。このパラメーターはオプションで、デフォルト値は TRUE です。

BOOL bCaseSens

この値が True の場合は、待ち条件は大/小文字の区別があるものとして検査されます。このパラメーターはオプションです。デフォルトは TRUE です。

戻り値

メソッドは、条件が合致していれば True を 戻し、nTimeOut (ミリ秒) が経過した場合は False を戻します。

注: テスト条件が FALSE を戻すときに、nTimeOut がデフォルト値 (INFINITE) である場合は、 このメソッドはブロックされます。

例

```
// set up PS
ECLPS ps = new ECLPS('A');

// do the wait
BOOL waitOK = ps.WaitWhileStringInRect("LOGON",1,1,23,80);

// do the processing for when the screen goes away
```

WaitForAttrib

WaitForAttrib メソッドは、指定行/列位置にある ECLPS オブジェクトに関連した接続 の表示スペースで、指定された属性値が現れるまで待ちます。オプションの MaskData パラメーターを使用して、どの属性値を探索するのか を制御することができます。オプションのプレーン・パラメーターに より、4 つの PS プレーンの内から任意のプレーンを選択することが可能となります。

プロトタイプ

```
BOOL WaitForAttrib(int Row, int Col, unsigned char AttribDatum, unsigned char MskDatum= 0xFF, PS_PLANE plane = FieldPlane, long TimeOut = INFINITE, BOOL bWaitForIR = TRUE)
```

パラメーター

int Row

属性の行の位置。

int Col

属性の列の位置。

unsigned char AttribDatum

待機する属性値。この値は 1 バイトの 16 進数値です。

unsigned char MskDatum

属性をマスクするのに使用する 1 バイトの 16 進数の値。このパラメーターはオプションです。デフォルト値は 0xFF です。

PS_PLANE plane

取得する属性のプレーン。プレーンは、以下のような値が可能です。**TextPlane**、**ColorPlane**、**FieldPlane**、**ExfieldPlane**。その他の ECL プレーンの内容および形式については、ECL プレーン – 形式および内容 (ページ 433)を参照してください。

このパラメーターはオプションです。デフォルト値は、FieldPlane です。

long nTimeOut

待機する時間の最大長 (ミリ秒)。このパラメーターはオプションです。デフォルトは、Infinite (無期限) です。

BOOL bWaitForIR

この値が True である場合、待ち条件の基準を満たすと、関数は OIA が PS の入力受け入れ準備完了 を示すまで待機します。このパラメーターはオプションで、デフォルト値は TRUE です。

戻り値

メソッドは、条件が合致していれば True を 戻し、nTimeOut (ミリ秒) が経過した場合は False を戻します。

注: テスト条件が FALSE を戻すときに、nTimeOut がデフォルト値 (INFINITE) である場合は、 このメソッドはブロックされます。

例

```
// set up PS
ECLPS ps = new ECLPS('A');

// do the wait
BOOL waitOK = ps.WaitForAttrib(10, 16, 0xE0, 0xFF, FieldPlane, INFINITE, FALSE);

// do the processing for when the screen goes away
```

WaitWhileAttrib

WaitWhileAttrib メソッドは、指定行/列位置にある ECLPS オブジェクトに関連した接続 の表示スペースに、指定された属性値が表示されている間待ちます。オプションの MaskData パラメーターを使用して、どの属性値を探索するのか を制御することができます。オプションのプレーン・パラメーターに より、4 つの PS プレーンの内から任意のプレーンを選択することが可能となります。

プロトタイプ

```
BOOL WaitWhileAttrib(int Row, int Col, unsigned char AttribDatum, unsigned char MskDatum= 0xFF, PS_PLANE plane = FieldPlane, long TimeOut = INFINITE, BOOL bWaitForIR = TRUE)
```

パラメーター

int Row

属性の行の位置。

int Col

属性の桁位置 (符号なし)。

char AttribDatum

待機する属性値。この値は 1 バイトの 16 進数値です。

unsigned char MskDatum

属性をマスクするのに使用する 1 バイトの 16 進数の値。このパラメーターはオプションです。デフォルト値は 0xFF です。

PS_PLANE plane

取得する属性のプレーン。プレーンは、以下のような値が可能です。**TextPlane**、**ColorPlane**、**FieldPlane**、**ExfieldPlane**。その他の ECL プレーンの内容および形式については、ECL プレーン – 形式および内容 (ページ 433)を参照してください。

このパラメーターはオプションです。デフォルト値は、FieldPlane です。

long nTimeOut

待機する時間の最大長 (ミリ秒)。このパラメーターはオプションです。デフォルトは、Infinite (無期限) です。

BOOL bWaitForIR

この値が True である場合、待ち条件の基準を満たすと、関数は OIA が PS の入力受け入れ準備完了 を示すまで待機します。このパラメーターはオプションで、デフォルト値は TRUE です。

戻り値

メソッドは、条件が合致していれば True を 戻し、nTimeOut (ミリ秒) が経過した場合は False を戻します。

注: テスト条件が FALSE を戻すときに、nTimeOut がデフォルト値 (INFINITE) である場合は、 このメソッドはブロックされます。

例

```
// set up PS
ECLPS ps = new ECLPS('A');

// do the wait
BOOL waitOK = ps.WaitWhileAttrib(10, 16, 0xE0, 0xFF, FieldPlane, INFINITE, FALSE);

// do the processing for when the screen goes away
```

WaitForScreen

ECLScreenDesc パラメーターにより記述された画面が表示スペースに 現れるのを同期して待ちます。

プロトタイプ

BOOL WaitForScreen(ECLScreenDesc* screenDesc, long TimeOut = INFINITE)

パラメーター

ECLScreenDesc

画面を記述する screenDesc オブジェクト (ECLScreenDesc クラス (ページ 168)を参照)。

long nTimeOut

待機する時間の最大長 (ミリ秒)。このパラメーターはオプションです。デフォルトは、Infinite (無期限) です。

戻り値

メソッドは、条件が合致していれば True を 戻し、nTimeOut (ミリ秒) が経過した場合は False を戻します。

注: テスト条件が FALSE を戻すときに、nTimeOut がデフォルト値 (INFINITE) である場合は、 このメソッドはブロックされます。

例

```
// set up PS
ECLPS ps = new ECLPS('A');

// set up screen description
ECLScreenDesc eclSD = new ECLScreenDesc();
eclSD.AddCursorPos(23,1);
eclSD.AddString("LOGON");

// do the wait
int TimeOut = 5000;
BOOL waitOK = ps.WaitForScreen(eclSD, timeInt.intValue());

// do processing for the screen
```

WaitWhileScreen

ECLScreenDesc パラメーターにより記述された画面が表示スペースから 無くなるまで同期して待ちます。

プロトタイプ

BOOL WaitWhileScreen(ECLScreenDesc* screenDesc, long TimeOut = INFINITE)

パラメーター

ECLScreenDesc

画面を記述する screenDesc オブジェクト (ECLScreenDesc メソッド (ページ 169)を参照)。

long nTimeOut

待機する時間の最大長 (ミリ秒)。このパラメーターはオプションです。デフォルトは、Infinite (無期限) です。

戻り値

メソッドは、条件が合致していれば True を 戻し、nTimeOut (ミリ秒) が経過した場合は False を戻します。

注: テスト条件が FALSE を戻すときに、nTimeOut がデフォルト値 (INFINITE) である場合は、 このメソッドはブロックされます。

例

```
// set up PS
ECLPS ps = new ECLPS('A');
```

```
// set up screen description
ECLScreenDesc eclSD = new ECLScreenDesc();
eclSD.AddCursorPos(23,1);
eclSD.AddString("LOGON");

// do the wait
int TimeOut = 5000;
BOOL waitOK = ps.WaitWhileScreen(eclSD, timeInt.intValue());

// do processing for when the screen goes away
```

RegisterPSEvent

このメンバー関数は、PS 更新イベントの通知を受け取るためのアプリケーション・オブジェクト を登録します。アプリケーションでこの関数を使用するには、ECLPSNotify また は ECLPSListener のいずれかから派生したオブジェクトを作成しなければなりません。作成すると、そのオブジェクトを指すポインターは この登録関数に渡されます。通知あるいはリスナー・オブジェクトの数が、同時に登録されることがあります。複数のリスナーがイベントを受信する順序は、定義されず想定することはできません。

この関数の異なるプロトタイプを使用すれば、異なるタイプの更新イベント、また、 その更新について異なる詳細レベルの生成が可能になります。最も単純な更新イベント は、ECLPSNotify オブジェクトを使用して登録されます。このタイプの登録では、 各 PS 更新ごとにイベントを作成します。この更新についての情報は生成されません。詳細については、ECLPSNotify オブジェクトの説明を参照してください。

更新についてより多くの情報を必要とするアプリケーションでは、ECLPSListener オブジェクト を登録することができます。このオブジェクトの登録により、アプリケーションは あるタイプの更新 (例えば、キー・ストロークのようなローカル端末の機能) を無視したり、 更新された画面の領域を判別したりすることができます。詳細については、ECLPSListener オブジェクトの 説明を参照してください。ECLPSListener オブジェクトを 登録する場合は、アプリケーションはイベントの原因となる更新のタイプをオプションで 指定することができます。

この関数を使用して ECLPSNotify または ECLPSListener オブジェクトが登録されれば、 表示スペースに更新が行われるたびに、この NotifyEvent() メソッドが 呼び出されます。短時間内での PS に対する複数の更新は、単一のイベントに集約されることがあります。

アプリケーションは、これを破棄するのに先立って通知/リスナー・オブジェクトを登録抹消する 必要があります。ECLPS オブジェクトが破棄されると、このオブジェクトは自動的に 抹消されます。

プロトタイプ

void RegisterPSEvent(ECLPSNotify * notify) void RegisterPSEvent(ECLPSListener * listener) void RegisterPSEvent(ECLPSListener * listener, int type)

パラメーター

ECLPSNotify *

登録する ECLPSNotify オブジェクトに対するポインター。

ECLPSListener *

登録する ECLPSListener オブジェクトに対するポインター。

整数

以下のイベントの原因となる更新のタイプ。

- USER_EVENTS (ローカル端末の機能)
- HOST_EVENTS (ホスト更新)
- ALL_EVENTS (すべての更新)

戻り値

なし

StartMacro

StartMacro メソッドは、MacroName パラメーターにより指示された Z and I Emulator for Windows のマクロ・ファイルを実行します。

プロトタイプ

void StartMacro(String MacroName)

パラメーター

String MacroName

Z and I Emulator for Windows のユーザー・クラス・アプリケーション・データ・ディレクトリー (インストール時に指定) に入っているマクロ・ファイルの名前でファイル拡張子を持っていない。このメソッドは、 長いファイル名をサポートしません。

戻り値

なし

使用上の注意

マクロ名には、短いファイル名を使用する必要があります。このメソッドは、 長いファイル名をサポートしません。

例

以下の例は、マクロを開始する方法を示しています。

```
Dim PS as Object

Set PS = CreateObject("ZIEWin.autECLPS")
```

```
PS.StartMacro "mymacro"
```

UnregisterPSEvent

このメンバー関数は、RegisterPSEvent 関数を使用して通信イベント用に事前に 登録されているアプリケーション・オブジェクトの登録を抹消します。イベントを受信するための登録済みオブジェクトの場合、先にこの関数を呼び出して その登録を抹消しないかぎり、オブジェクトを破棄してはなりません。特定のオブジェクトが現在登録されていない場合は、アクションは取られず エラーは発生しません。

ECLPSNotify または ECLPSListener オブジェクトが登録されていない場合は、 この NotifyStop() メソッドが呼び出されます。

プロトタイプ

void UnregisterPSEvent(ECLPSNotify * notify) void UnregisterPSEvent(ECLPSListener * listener) void UnregisterPSEvent(ECLPSListener * listener, int type)

パラメーター

ECLPSNotify *

抹消する ECLPSNotify オブジェクトに対するポインター。

ECLPSListener *

抹消する ECLPSListener オブジェクトに対するポインター。

整数

登録済み更新のタイプ。

- USER_EVENTS (ローカル端末の機能)
- HOST_EVENTS (ホスト更新)
- ALL_EVENTS (すべての更新)

戻り値

なし

ECLPSEvent クラス

ECLPSEvent オブジェクトは、表示スペースが更新された ときに ECLListener オブジェクトに渡されます。このイベント・オブジェクトは、 表示スペースの更新イベントを表し、更新についての情報が入っています。

更新された表示スペースの領域を判別するのに、アプリケーションで 使用できる関数に 2 つのセットがあります。GetStart() と GetEnd() の両メソッドは、 線形位置を戻すことにより、表示スペース内の更新領域の開始位置と

終了位置を示します。リニア・アドレッシングは、左上端の文字が 1 で開始され、 左から右へ進み、折り返して行を進めます。対応する関数の セット (GetStartRow、GetStartCol、GetEndRow、 GetEndCol) は、行/列の座標で同じ情報を戻します。

更新領域には、開始文字から終了文字までのすべての PS 文字が 含まれます (両端の文字を含む)。開始位置と終了位置が同じ行にない場合は、更新領域はある行の終了から次行の最初の桁へ 折り返します。更新領域は (一般的に) 長方形でないことに注意してください。開始位置が終了位置より大きい場合は、更新領域は開始位置で開始され、画面の最後の文字から 最初の文字へ折り返し、終了位置へ続きます。

更新領域は、実際に変更になった表示スペース部分より多くの部分を 含むことがありますが、少なくとも変更された部分をカバーしていることが保証されている ことに注意してください。短時間に複数の PS 更新が起こると、これらの変更は単一のイベント に集約されることがあり、ここでは更新領域にはすべての更新の合計が含まれます。

派生

ECLBase > ECLEvent > ECLPSEvent

使用上の注意

アプリケーションは、このクラスを直接使用しません。アプリケーションは、 ECLListener から派生するオブジェクトを作成し、これが ECLPSEvent オブジェクト を ECLListener::NotifyEvent メソッド上で受け取ります。

ECLPSEvent メソッド

以下のセクションでは、ECLPSEvent クラスおよびそれから派生したすべてのクラスにおいて 有効なメソッドについて説明します。

ECLPS * GetPS() int GetType() ULONG GetStart() ULONG GetEnd() ULONG GetStartRow() ULONG GetStartCol() ULONG GetEndRow() ULONG GetEndCol()

GetPS

このメソッドは、このイベントを生成した ECLPS オブジェクトを戻します。

プロトタイプ

ECLPS * GetPS()

パラメーター

なし

戻り値

ECLPS *

このイベントを生成した ECLPS オブジェクトに対するポインター。

GetType

このメソッドは、このイベントを生成した表示スペース更新のタイプを 戻します。戻り値は、USER_EVENTS または HOST_EVENTS のいずれかです。ユーザー・イベントは、 ローカル端末の機能 (例えば、ユーザーまたはプログラミング API によって入力される キー・ストローク) として発生する PS 更新として定義されます。ホスト・イベントは、 ホスト・アウトバウンド・データ・ストリームから発生する PS 更新です。

プロトタイプ

int GetType()

パラメーター

なし

戻り値

整数

USER_EVENTS または HOST_EVENTS 定数を戻します。

GetStart

このメソッドは、更新領域の先頭の表示スペース内の 線形位置を戻します。この位置の行/列の座標は、 表示スペースに現在定義されている桁の数によって決まることに 注意してください。この値が GetEnd() によって戻された値よりも大きい場合には、 更新領域はこの位置から開始され、画面の最後から最初へ折り返して、 終了位置へ続きます。

プロトタイプ

ULONG GetStart()

パラメーター

なし

戻り値

ULONG

更新領域の先頭の線形位置。

GetEnd

このメソッドは、更新領域の最後の表示スペース内の 線形位置を戻します。この位置の行/列の座標は、 表示スペースに現在定義されている桁の数によって決まることに 注意してください。この値が GetStart() によって戻された値よりも小さい場合には、 更新領域は GetStart() 位置から開始され、画面の最後から最初へ折り返して、 この位置へ続きます。

プロトタイプ

ULONG GetEnd()

パラメーター

なし

戻り値

ULONG

更新領域の最後の線形位置。

GetStartRow

このメソッドは、更新領域の先頭の表示スペース内の 行番号を戻します。開始する行/列位置が終了する行/列位置よりも大きい場合には、 更新領域はこの位置から開始され、画面の最後から最初へ折り返して、 終了する位置へ続きます。

プロトタイプ

ULONG GetStartRow()

パラメーター

なし

戻り値

ULONG

更新領域の先頭の行番号。

GetStartCol

このメソッドは、更新領域の先頭の表示スペース内の 桁番号を戻します。開始する行/列位置が終了する行/列位置よりも 大きい場合には、更新領域は開始する行 /桁から開始され、 画面の最後から最初へ折り返して、 終了する位置へ続きます。

プロトタイプ

ULONG GetStartCol()

パラメーター

なし

戻り値

ULONG

更新領域の先頭の桁番号。

GetEndRow

このメソッドは、更新領域の最後の表示スペース内の 行番号を戻します。開始する行/列位置が終了する行/列位置よりも 大きい場合には、更新領域は開始する行/列から開始され、 画面の最後から最初へ折り返して、終了する行/列へ続きます。

プロトタイプ

ULONG GetEndRow()

パラメーター

なし

戻り値

ULONG

更新領域の最後の行番号。

GetEndCol

このメソッドは、更新領域の最後の表示スペース内の 桁番号を戻します。開始する行/列位置が終了する行/列位置よりも 大きい場合には、更新領域は開始する行/列から開始され、 画面の最後から最初へ折り返して、終了する行/列へ続きます。

プロトタイプ

ULONG GetEndCol()

パラメーター

なし

戻り値

ULONG

更新領域の最後の桁番号。

ECLPSListener クラス

ECLPSListener は、抽象基本クラスです。アプリケーションは、このクラスの インスタンスを直接作成することはできません。アプリケーションでこのクラスを 使用するには、ECLPSListener から派生した独自のクラスを定義しなければなりません。アプリケーションは、すべてのメソッドをこのクラス内で 実装する必要があります。

アプリケーションが表示スペースの更新の通知を受けられるように するには、ECLPSListener クラスを使用します。イベントは、ホスト画面が更新 される (例えば、表示スペースのデータがプレーンで変更される) たびに 生成されます。

このクラスは、ECLPSNotify クラスと同様に、PS 更新の通知を受信するために 使用されます。ただし、これは ECLPSNotify クラスよりも多くの更新の原因と 有効範囲についての情報を受け取る点で異なります。一般的に、このクラスの使用は、 各イベントごとにより多くの情報が生成されるため、処理時間とメモリーの点から より不経済であるといえます。ホスト画面のビジュアル表示を効率よく更新する 必要のあるアプリケーションの場合、このクラスは更新が起こるたびに 表示を再ドローするよりはより効率的である可能性があります。このクラスを使用すると、 アプリケーションは変更されたビジュアル表示の部分だけを更新することができます。

また、このクラスは ECLPSNotify とも異なり、すべてのメソッドは純粋に仮想であり、 そのためこれらのメソッドはアプリケーションによって実装される必要が あります (デフォルトの実装はありません)。

派生

ECLBase > ECLListener > ECLPSListener

使用上の注意

アプリケーションがこのクラスを使用する PS 更新の通知を受けるには、 次に示すステップを実行しなければなりません。

1. ECLPSListener から派生したクラスを定義します。
2. ECLPSListener から派生したクラスのすべてのメソッドを実装します。
3. 派生クラスのインスタンスを作成します。
4. そのインスタンスを ECLPS::RegisterPSEvent() メソッドで登録します。

登録が完了した後で、表示スペースの更新によって ECLPSListener から 派生したクラスの NotifyEvent() メソッドが呼び出されます。その結果アプリケーションは、メソッド呼び出しでシステムに提供された ECLPSEvent オブジェクト を使用して、PS 更新の原因および影響された画面の領域を判別することができます。

短時間に発生する複数の PS の更新は、単一のイベント通知に集約される ことがあるので注意してください。

アプリケーションは、派生したクラス用に自身のコンストラクター およびデストラクターを任意で提供することができます。これが便利なのは、 アプリケーションが特定のインスタンス固有データをそのクラス内に保管してから、 その情報をコンストラクター上のパラメーターとして渡す必要がある場合です。

イベントの登録時にエラーが検出された場合、ECLErr オブジェクトを 使用して NotifyError() メンバー関数が呼び出されます。エラーの後で、続いて イベントが生成されることも、生成されないこともあります。イベント生成が終了した とき (エラーか、あるいはその他の理由から) には、NotifyStop() メンバー関数が 呼び出されます。

ECLPSListener メソッド

以下のセクションでは、ECLPSListener クラスおよびそれから派生したすべてのクラス において有効なメソッドについて説明します。コンストラクターおよびデストラクター 以外のすべてのメソッドは、仮想メソッドであることに注意してください。

ECLPSListener() ECLPSListener() virtual void NotifyEvent(ECLPSEvent * event) = 0 virtual void NotifyError(ECLPS * PSObj, ECLErr ErrObj) = 0 virtual void NotifyStop(ECLPS * PSObj, int Reason) = 0

NotifyEvent

このメソッドは、純粋仮想 メンバー関数です (アプリケーションは ECLPSListener から派生したクラス内にこの関数を実装しなければなりません)。このメソッドは、 表示スペースが更新され、更新イベントを受け取るために このオブジェクトが登録されるたびに、呼び出されます。パラメーターとして渡される ECLPSEvent オブジェクトには、変更された画面の領域を 含むイベントについての情報が入っています。詳しくは、ECLPSEvent クラス (ページ 154)を参照してください。

複数の PS 更新は単一のイベントに集約されることがあり、結果として、 このメソッドに対してはただ 1 つの呼び出しとなります。ECLPSEvent オブジェクトに含まれる 変更された領域には、すべての変更の合計が入っています。

適切なパラメーターを ECLPS::RegisterPSEvent() メソッドに提供することにより、 イベントが PS 更新の特定タイプだけに制限されることがあります。例えば、 アプリケーションは、ホストからの更新のためだけに通知を受け、ローカル・キー・ストローク のためには通知を受けないよう選択することができます。

プロトタイプ

virtual void NotifyEvent(ECLPSEvent * event) = 0

パラメーター

ECLPSEvent *

PS 更新を表す ECLPSEvent オブジェクトを指すポインター。

戻り値

なし

NotifyError

このメソッドは、イベントの生成時に ECLPS オブジェクトがエラーを検出するたびに 呼び出されます。エラー・オブジェクトには、そのエラーについての情報が 含まれます (ECLErr クラス (ページ 59)を参照)。エラーの特性に応じて 、エラーの後で続いてイベントが生成されることがあります。エラーが原因でイベント生成が停止した 場合、NotifyStop() メソッドが呼び出されます。

これは、アプリケーションが実装しなければならない純粋仮想 メソッドです。

プロトタイプ

virtual void NotifyError(ECLPS * PSObj, ECLErr ErrObj) = 0

パラメーター

ECLPS *

このイベントを生成した ECLPS オブジェクトに対するポインター。

ECLErr

エラーを記述する ECLErr オブジェクト。

戻り値

なし

NotifyStop

イベント生成が何らかの理由 (例えば、エラー条件が原因か、 または ECLPS::UnregisterPSEvent の呼び出しなどが原因) で停止すると、 このメソッドが呼び出されます。

これは、アプリケーションが実装しなければならない純粋仮想 メソッドです。

現在、理由コード・パラメーターは未使用であり、ゼロになります。

プロトタイプ

virtual void NotifyStop(ECLPS * PSObj, int Reason) = 0

パラメーター

ECLPS *

このイベントを生成した ECLPS オブジェクトに対するポインター。

整数

理由イベントの生成が停止しました (現在は未使用でゼロです)。

戻り値

なし

ECLPSNotify クラス

ECLPSNotify は、抽象基本クラスです。アプリケーションは、このクラスの インスタンスを直接作成することはできません。アプリケーションでこのクラスを使用するには、ECLPSNotify から 派生した独自のクラスを定義しなければなりません。アプリケーションは、 その派生クラス内に NotifyEvent() メンバー関数を実装しなければなりません。また、 オプションで NotifyError() および NotifyStop() メンバー関数を実装することも できます。

アプリケーションが表示スペースに対する更新についての通知を 受けられるようにするには、ECLPSNotify クラスを使用します。イベントは、ホスト画面が更新 される (例えば、表示スペースのデータがプレーンで変更される) たびに 生成されます。

このクラスは、ECLPSListener クラスと同様に、PS 更新の通知を受け取るために 使用されます。ただし、これは更新の原因と有効範囲についての情報を受け取らない点で、 ECLPSNotify クラスとは異なります。一般的にこのクラスの使用は、各イベントごとに 情報を生成する必要がないため、処理時間とメモリー使用の観点からより効率的であるといえます。このクラスは、更新の通知だけを必要とし、イベントの原因が何か、あるいは 画面のどの部分が更新されたかの詳細については必要としないアプリケーションに、 使用することができます。

また、このクラスは ECLPSListener とも異なり、ここではデフォルトの実装 が NotifyError() および NotifyStop() メソッド用に用意されています。

派生

ECLBase > ECLNotify > ECLPSNotify

使用上の注意

アプリケーションがこのクラスを使用する PS 更新の通知を受けるには、 次に示すステップを実行しなければなりません。

1. ECLPSNotify から派生したクラスを定義します。
2. ECLPSNotify から派生したクラスの NotifyEvent メソッドを実装します。
3. オプションで、ECLPSNotify の他のメンバー関数を実装します。
4. 派生クラスのインスタンスを作成します。
5. そのインスタンスを ECLPS::RegisterPSEvent() メソッドで登録します。

登録が完了した後で、表示スペースの更新によって ECLPSNotify から 派生したクラスの NotifyEvent() メソッドが呼び出されます。

短時間に発生する複数の PS の更新は、単一のイベント通知に集約される ことがあるので注意してください。

アプリケーションは、派生したクラス用に自身のコンストラクター およびデストラクターを任意で提供することができます。これが便利なのは、 アプリケーションが特定のインスタンス固有データをそのクラス内に保管してから、 その情報をコンストラクター上のパラメーターとして渡す必要がある場合です。

イベントの登録時にエラーが検出された場合、ECLErr オブジェクトを 使用して NotifyError() メンバー関数が呼び出されます。エラーの後で、続いて イベントが生成されることも、生成されないこともあります。イベント生成が終了した とき (エラーか、あるいはその他の理由から) には、NotifyStop() メンバー関数が 呼び出されます。NotifyError() のデフォルトの実装によって、 ユーザーにメッセージ・ボックスが用意され、ECLErr オブジェクトから取り出された エラー・メッセージのテキストが示されます。

何らかの理由でイベント通知が停止したとき (エラーまたは ECLPS::UnregisterPSEvent 呼び出し) には、NotifyStop() メンバー関数が呼び出されます。デフォルトの NotifyStop() の 実装は、何も実行しません。

ECLPSNotify メソッド

以下のセクションでは、ECLPSNotify クラスおよびそれから派生したすべてのクラスにおいて 有効なメソッドについて説明します。

ECLPSNotify()=0 ~ECLPSNotify() virtual void NotifyEvent(ECLPS * PSObj) virtual void NotifyError(ECLPS * PSObj, ECLErr ErrObj) virtual void NotifyStop(ECLPS * PSObj, int Reason)

NotifyEvent

このメソッドは、純粋仮想 メンバー関数です (アプリケーションは ECLPSNotify から派生したクラス内にこの関数を実装しなければなりません)。このメソッドは、 表示スペースが更新され、更新イベントを受け取るために このオブジェクトが登録されるたびに、呼び出されます。

複数の PS 更新は単一のイベントに集約されることがあり、結果として、 このメソッドに対してはただ 1 つの呼び出しとなります。

プロトタイプ

virtual void NotifyEvent(ECLPS * PSObj)

パラメーター

ECLPS *

このイベントを生成した ECLPS オブジェクトに対するポインター。

戻り値

なし

NotifyError

このメソッドは、イベントの生成時に ECLPS オブジェクトがエラーを検出するたびに 呼び出されます。エラー・オブジェクトには、そのエラーについての情報が 含まれます (ECLErr クラス (ページ 59)を参照)。エラーの特性に応じて 、エラーの後で続いてイベントが生成されることがあります。エラーが原因でイベント生成が停止した 場合、NotifyStop() メソッドが呼び出されます。

アプリケーションは、この関数を実装するか、 または ECLPSNotify 基本クラスにそれを処理させるかを選ぶことができます。デフォルトの実装 は、ECLErr::GetMsgText() メソッドから提供される テキストを使用して、メッセージ・ボックスにエラーを表示します。アプリケーションが、 その派生クラス内にこの関数を実装すると、それによって この振る舞いがオーバーライドされます。

プロトタイプ

virtual void NotifyError(ECLPS * PSObj, ECLErr ErrObj) = 0

パラメーター

ECLPS *

このイベントを生成した ECLPS オブジェクトに対するポインター。

ECLErr

エラーを記述する ECLErr オブジェクト。

戻り値

なし

NotifyStop

イベント生成が何らかの理由 (例えば、エラー条件が原因か、 または ECLPS::UnregisterPSEvent の呼び出しなどが原因) で停止すると、 このメソッドが呼び出されます。

現在、理由コード・パラメーターは未使用であり、ゼロになります。

この関数のデフォルトの実装では、何も実行しません。

プロトタイプ

virtual void NotifyStop(ECLPS * PSObj, int Reason) = 0

パラメーター

ECLPS *

このイベントを生成した ECLPS オブジェクトに対するポインター。

整数

理由イベントの生成が停止しました (現在は未使用でゼロです)。

戻り値

なし

ECLRecoNotify クラス

ECLRecoNotify を使用して、 ECLScreenReco イベントを受け取り、処理するオブジェクトをインプリメントすることができます。イベントは、PS の中の画面で、 ECLScreenReco 内の ECLScreenDesc オブジェクトと一致するものがあるたびに生成されます。イベント生成が停止したり、イベント生成中にエラーが生じたりすると、 特殊イベントが生成されます。

アプリケーションが ECLScreenReco イベントの通知を受けるには、 次に示すステップを実行しなければなりません。

1. ECLRecoNotify クラスから派生するクラスを定義します。
2. NotifyEvent()、NotifyStop()、および NotifyError() メソッドをインプリメントする。
3. 新しいクラスのインスタンスを作成する。
4. そのインスタンスを ECLScreenReco::RegisterScreen() メソッドで登録します。

例については、ECLScreenReco クラス (ページ 178)を参照してください。

派生

ECLBase > ECLNotify > ECLRecoNotify

ECLRecoNotify メソッド

ECLRecoNotify に有効なメソッドを以下にリストします。

ECLRecoNotify() ~ECLRecoNotify() void NotifyEvent(ECLPS *ps, ECLScreenDesc *sd) void NotifyStop(ECLPS *ps, ECLScreenDesc *sd) void NotifyError(ECLPS *ps, ECLScreenDesc *sd, ECLErr e)

ECLRecoNotify コンストラクター

ECLRecoNotify の空インスタンスを作成します。

プロトタイプ

ECLRecoNotify()

パラメーター

なし

戻り値

なし

例

例については、ECLScreenReco クラス (ページ 178)を参照してください。

ECLRecoNotify デストラクター

ECLRecoNotify のインスタンスを破棄します。

プロトタイプ

~ECLRecoNotify()

パラメーター

なし

戻り値

なし

例

例については、ECLScreenReco クラス (ページ 178)を参照してください。

NotifyEvent

ECLRecoNotify オブジェクトで ECLScreenReco 上に登録された ECLScreenDesc が 表示スペースに現れるときに、呼び出されます。

プロトタイプ

void NotifyEvent(ECLPS *ps, ECLScreenDesc *sd)

パラメーター

ECLPS ps

ユーザーが登録した ECLPS オブジェクト。

ECLScreenDesc sd

ユーザーが登録した ECLScreenDesc。

戻り値

なし

例

例については、ECLScreenReco クラス (ページ 178)を参照してください。

NotifyStop

ECLScreenReco オブジェクトが、登録済み ECLScreenDesc オブジェクト用の ECLPS オブジェクト のモニターを停止したときに呼び出されます。

プロトタイプ

void NotifyStop(ECLPS *ps, ECLScreenDesc *sd)

パラメーター

ECLPS ps

ユーザーが登録した ECLPS オブジェクト。

ECLScreenDesc sd

ユーザーが登録した ECLScreenDesc。

戻り値

なし

例

例については、ECLScreenReco クラス (ページ 178)を参照してください。

NotifyError

ECLScreenReco オブジェクトでエラーが発生したときに呼び出されます。

プロトタイプ

void NotifyError(ECLPS *ps, ECLScreenDesc *sd, ECLErr e)

パラメーター

ECLPS ps

ユーザーが登録した ECLPS オブジェクト。

ECLScreenDesc sd

ユーザーが登録した ECLScreenDesc。

ECLErr e

エラー情報を含む ECLErr オブジェクト。

戻り値

なし

例

例については、ECLScreenReco クラス (ページ 178)を参照してください。

ECLScreenDesc クラス

ECLScreenDesc は、ホスト・アクセス・クラス・ライブラリーの画面認識テクノロジーの画面を記述するために使用されるクラスです。これは、カーソル位置はもちろんのこと、 これを説明する表示スペースの 4 つの主な プレーン (TEXT、 FIELD、EXFIELD、COLOR) すべてを使用します。

このオブジェクトで用意されているメソッドを使用して、プログラマーは指定された画面が ホスト・サイド・アプリケーションでどのように表示されるかを詳細に記述することが できます。ECLScreenDesc オブジェクトが作成されて設定されると、 それが ECLPS 上にある同期 WaitFor... メソッドに渡されるか、 または ECLScreenReco に渡されます。 これは、ECLScreenDesc オブジェクトと一致する画面が PS に表示される場合に、 非同期イベントを起動します。

派生

ECLBase > ECLScreenDesc

ECLScreenDesc メソッド

ECLScreenDesc に有効なメソッドを以下にリストします。

ECLScreenDesc() ~ECLScreenDesc() void AddAttrib(BYTE attrib, UINT pos, PS_PLANE plane=FieldPlane); void AddAttrib(BYTE attrib, UINT row, UINT col, PS_PLANE plane=FieldPlane); void AddCursorPos(uint row, uint col) void AddNumFields(uint num) void AddNumInputFields(uint num) void AddOIAInhibitStatus(OIAStatus type=NOTINHIBITED) void AddString(LPCSTR s, UINT row, UINT col, BOOL caseSensitive=TRUE) void AddStringInRect(char * str, int Top, int Left, int Bottom, int Right, BOOL caseSense=TRUE) void Clear()

ECLScreenDesc コンストラクター

ECLScreenDesc の空インスタンスを作成します。

プロトタイプ

ECLScreenDesc()

パラメーター

なし

戻り値

なし

例

```
// set up PS
ECLPS ps = new ECLPS('A');

// set up screen description
ECLScreenDesc eclSD = new ECLScreenDesc();
eclSD.AddCursorPos(23,1);
eclSD.AddString("LOGON");

// do the wait
int TimeOut = 5000;
BOOL waitOK = eclPS.WaitForScreen(eclSD, timeInt.intValue());
```

ECLScreenDesc デストラクター

ECLScreenDesc のインスタンスを破棄します。

プロトタイプ

~ ECLScreenDesc()

パラメーター

なし

戻り値

なし

例

```
// set up PS
ECLPS ps = new ECLPS('A');

// set up screen description
ECLScreenDesc eclSD = new ECLScreenDesc();
eclSD.AddCursorPos(23,1);
eclSD.AddString("LOGON");

// do the wait
int TimeOut = 5000;
BOOL waitOK = eclPS.WaitForScreen(eclSD, timeInt.intValue());
// destroy the descriptor
delete eclSD;
```

AddAttrib

画面記述の指定位置に属性値を追加します。

プロトタイプ

```
void AddAttrib(BYTE attrib, UINT pos, PS_PLANE plane=FieldPlane); void AddAttrib(BYTE attrib, UINT row, UINT col, PS_PLANE plane=FieldPlane);
```

パラメーター

BYTE attrib

追加する属性値。

int row

行位置。

int col

桁位置。

PS_PLANE plane

属性が常駐するプレーン。有効な値は、以下のとおりです。TextPlane、ColorPlane、FieldPlane、ExfieldPlane、GridPlane。**TextPlane**、**ColorPlane**、**FieldPlane**、**ExfieldPlane**。その他の ECL プレーンの内容および形式に ついては、ECL プレーン ー 形式および内容 (ページ 433)を参照してください。

戻り値

なし

例

```
// set up PS
ECLPS ps = new ECLPS('A');

// set up screen description
ECLScreenDesc eclSD = new ECLScreenDesc();
eclSD.AddAttrib(0xe8, 1, 1, ColorPlane);
eclSD.AddCursorPos(23,1);
eclSD.AddNumFields(45) ;
eclSD.AddNumInputFields(17) ;
AddOIAInhibitStatus(NOTINHIBITED) ;
eclSD.AddString("LOGON"., 23, 11, TRUE) ;
eclSD.AddStringInRect("PASSWORD", 23, 1, 24, 80, FALSE) ;

// do the wait
int TimeOut = 5000;
BOOL waitOK = eclPS.WaitForScreen(eclSD, timeInt.intValue());
```

AddCursorPos

指定位置に画面記述のためのカーソル位置をセットします。

プロトタイプ

void AddCursorPos(uint row, uint col)

パラメーター

uint row

行位置。

uint col

桁位置。

戻り値

なし

例

```
// set up PS
ECLPS ps = new ECLPS('A');

// set up screen description
ECLScreenDesc eclSD = new ECLScreenDesc();
eclSD.AddAttrib(0xe8, 1, 1, ColorPlane);
eclSD.AddCursorPos(23,1);
eclSD.AddNumFields(45) ;
eclSD.AddNumInputFields(17) ;
AddOIAInhibitStatus(NOTINHIBITED) ;
eclSD.AddString("LOGON"., 23, 11, TRUE) ;
eclSD.AddStringInRect("PASSWORD", 23, 1, 24, 80, FALSE) ;

// do the wait
int TimeOut = 5000;
BOOL waitOK = eclPS.WaitForScreen(eclSD, timeInt.intValue());
```

AddNumFields

画面記述に入力フィールド数を追加します。

プロトタイプ

void AddNumFields(uint num)

パラメーター

uint num

フィールドの数。

戻り値

なし

例

```
// set up PS
ECLPS ps = new ECLPS('A');

// set up screen description
ECLScreenDesc eclSD = new ECLScreenDesc();
eclSD.AddAttrib(0xe8, 1, 1, ColorPlane);
eclSD.AddCursorPos(23,1);
eclSD.AddNumFields(45) ;
eclSD.AddNumInputFields(17) ;
AddOIAInhibitStatus(NOTINHIBITED) ;
eclSD.AddString("LOGON"., 23, 11, TRUE) ;
eclSD.AddStringInRect("PASSWORD", 23, 1, 24, 80, FALSE) ;

// do the wait
int TimeOut = 5000;
BOOL waitOK = eclPS.WaitForScreen(eclSD, timeInt.intValue());
```

AddNumInputFields

画面記述に入力フィールド数を追加します。

プロトタイプ

void AddNumInputFields(uint num)

パラメーター

uint num

入力フィールドの数。

戻り値

なし

例

```
// set up PS
ECLPS ps = new ECLPS('A');

// set up screen description
ECLScreenDesc eclSD = new ECLScreenDesc();
eclSD.AddAttrib(0xe8, 1, 1, ColorPlane);
eclSD.AddCursorPos(23,1);
eclSD.AddNumFields(45) ;
eclSD.AddNumInputFields(17) ;
AddOIAInhibitStatus(NOTINHIBITED) ;
eclSD.AddString("LOGON"., 23, 11, TRUE) ;
eclSD.AddStringInRect("PASSWORD", 23, 1, 24, 80, FALSE) ;

// do the wait
int TimeOut = 5000;
BOOL waitOK = eclPS.WaitForScreen(eclSD, timeInt.intValue());
```

AddOIAInhibitStatus

画面記述のための OIA モニターのタイプをセットします。

プロトタイプ

void AddOIAInhibitStatus(OIAStatus type=NOTINHIBITED)

パラメーター

OIAStatus type

OIA 状況のタイプ。現行の有効値は、DONTCARE および NOTINHIBITED。

戻り値

なし

例

```
// set up PS
ECLPS ps = new ECLPS('A');

// set up screen description
ECLScreenDesc eclSD = new ECLScreenDesc();
eclSD.AddAttrib(0xe8, 1, 1, ColorPlane);
eclSD.AddCursorPos(23,1);
eclSD.AddNumFields(45) ;
eclSD.AddNumInputFields(17) ;
AddOIAInhibitStatus(NOTINHIBITED) ;
eclSD.AddString("LOGON"., 23, 11, TRUE) ;
eclSD.AddStringInRect("PASSWORD", 23, 1, 24, 80, FALSE) ;
```

```
// do the wait
int TimeOut = 5000;
BOOL waitOK = eclPS.WaitForScreen(eclSD, timeInt.intValue());
```

AddString

画面記述の指定された位置にストリングを追加します。行と桁が指定されない場合は、ストリングは PS のどこに現れるか分かり ません。

注: 負の値は、PS の下部からの絶対位置です。例えば、row=-2 は、 全体が 24 行の内の行 23 を示しています。

プロトタイプ

void AddString(LPCSTR s, UINT row, UINT col, BOOL caseSensitive=TRUE)

パラメーター

LPCSTR s

追加するストリング。

uint row

行位置。

uint col

桁位置。

BOOL caseSense

この値が True である場合は、ストリングは大/小文字の区別付きで 追加されます。このパラメーターはオプションです。デフォルトは TRUE です。

戻り値

なし

例

```
// set up PS
ECLPS ps = new ECLPS('A');

// set up screen description
ECLScreenDesc eclSD = new ECLScreenDesc();
eclSD.AddAttrib(0xe8, 1, 1, ColorPlane);
eclSD.AddCursorPos(23,1);
eclSD.AddNumFields(45) ;
```

```
eclSD.AddNumInputFields(17) ;
AddOIAInhibitStatus(NOTINHIBITED) ;
eclSD.AddString("LOGON"., 23, 11, TRUE) ;
eclSD.AddStringInRect("PASSWORD", 23, 1, 24, 80, FALSE) ;

// do the wait
int TimeOut = 5000;
BOOL waitOK = eclPS.WaitForScreen(eclSD, timeInt.intValue());
```

AddStringInRect

画面記述の指定長方形内にストリングを追加します。

プロトタイプ

void AddStringInRect(char * str, int Top, int Left, int Bottom, int Right, BOOL caseSense=TURE)

パラメーター

char * str

追加するストリング。

int Top

左上行位置。このパラメーターはオプションです。デフォルトは最初の行です。

int Left

左上桁位置。このパラメーターはオプションです。デフォルトは最初の桁です。

int Bottom

右下行位置。このパラメーターはオプションです。デフォルトは最後の行です。

int Right

右下桁位置。このパラメーターはオプションです。デフォルトは最後の桁です。

BOOL caseSense

この値が True である場合は、ストリングは大/小文字の区別付きで 追加されます。このパラメーターはオプションです。デフォルトは TRUE です。

戻り値

なし

例

```
// set up PS
ECLPS ps = new ECLPS('A');

// set up screen description
```

```
ECLScreenDesc eclSD = new ECLScreenDesc();
eclSD.AddAttrib(0xe8, 1, 1, ColorPlane);
eclSD.AddCursorPos(23,1);
eclSD.AddNumFields(45) ;
eclSD.AddNumInputFields(17) ;
AddOIAInhibitStatus(NOTINHIBITED) ;
eclSD.AddString("LOGON"., 23, 11, TRUE) ;
eclSD.AddStringInRect("PASSWORD", 23, 1, 24, 80, FALSE) ;

// do the wait
int TimeOut = 5000;
BOOL waitOK = eclPS.WaitForScreen(eclSD, timeInt.intValue());
```

クリア

画面記述からすべての記述要素を取り除きます。

プロトタイプ

void Clear()

パラメーター

なし

戻り値

なし

例

```
// set up PS
ECLPS ps = new ECLPS('A');

// set up screen description
ECLScreenDesc eclSD = new ECLScreenDesc();
eclSD.AddAttrib(0xe8, 1, 1, ColorPlane);
eclSD.AddCursorPos(23,1);
eclSD.AddNumFields(45) ;
eclSD.AddNumInputFields(17) ;
AddOIAInhibitStatus(NOTINHIBITED) ;
eclSD.AddString("LOGON"., 23, 11, TRUE) ;
eclSD.AddStringInRect("PASSWORD", 23, 1, 24, 80, FALSE) ;

// do the wait
int TimeOut = 5000;
BOOL waitOK = eclPS.WaitForScreen(eclSD, timeInt.intValue());

// do processing for the screen

eclSD.Clear() // start over for a new screen
```

ECLScreenReco クラス

ECLScreenReco クラスは、ホスト・アクセス・クラス・ライブラリーの画面認識システム にとってはエンジンに相当するものです。これには、画面の記述を追加したり除去したりするためのメソッドが含まれています。また、それらの画面を認識したり、 画面用のハンドラー・コードに非同期にコールバックしたりするための論理も含まれています。

ECLScreenReco クラスのオブジェクトは、固有の「認識セット」として考えてください。オブジェクトは、画面を監視する複数の ECLPS オブジェクト、 検索する複数の画面、 および任意の ECLPS オブジェクトの中に画面を表示するときに呼び出す複数のコールバック・ポイントを持つことができます。

ユーザーはアプリケーションの開始時に ECLScreenReco オブジェクトを設定するだけでよく、 モニターしたい画面が ECLPS に現れるときに、 使用するコードが ECLScreenReco によって呼び出されます。ユーザーは、画面をモニターする 際に何も実行する必要はありません。

以下は、共通の実装の例です。

```
class MyApp {
ECLPS myECLPS('A'); // My main HACL PS object
ECLScreenReco myScreenReco(); // My screen reco object
ECLScreenDesc myScreenDesc(); // My screen descriptor
MyRecoCallback myCallback(); // My GUI handler

MyApp() {
// Save the number of fields for below
ECLFieldList *fl = myECLPS.GetFieldList()
Fl->Refresh();
int numFields = fl->GetFieldCount();

// Set up my HACL screen description object. Say the screen
// is identified by a cursor position, a key word, and the
// number of fields
myScreenDesc.AddCursorPos(23,1);
myScreenDesc.AddString("LOGON");
myScreenDesc.AddNumFields(numFields);
```

```
// Set up HACL screen reco object, it will begin monitoring here
myScreenReco.AddPS(myECLPS);
myScreenReco.RegisterScreen(&myScreenDesc, &myCallback);
}

 MyApp() {
myScreenReco.UnregisterScreen(&myScreenDesc, &myCallback);
myScreenReco.RemovePS(&eclPS);
}

public void showMainGUI() {
// Show the main application GUI, this is just a simple example
}
```

```
// ECLRecoNotify-derived inner class (the "callback" code)
class MyRecoCallback public: ECLRecoNotify {
public: void NotifyEvent(ECLScreenDesc *sd, ECLPS *ps) {
// GUI code here for the specific screen
// Maybe fire a dialog that front ends the screen
}

public void NotifyError(ECLScreenDesc *sd, ECLPS *ps, ECLErr e) {
// Error handling
}
```

```
public void NotifyStop(ECLScreenDesc *sd, ECLPS *ps, int Reason) {
// Possible stop monitoring, not essential
}
}

}

int main() {
MyApp app = new MyApp();
app.showMainGUI();
}
```

派生

ECLBase > ECLScreenReco

ECLScreenReco メソッド

以下のメソッドは、ECLScreenReco に有効です。

ECLScreenReco() ~ECLScreenReco() AddPS(ECLPS*) IsMatch(ECLPS*, ECLScreenDesc*) RegisterScreen(ECLScreenDesc*, ECLRecoNotify*) RemovePS(ECLPS*) UnregisterScreen(ECLScreenDesc*)

ECLScreenReco コンストラクター

ECLScreenReco の空インスタンスを作成します。

プロトタイプ

ECLScreenReco()

パラメーター

なし

戻り値

なし

例

ECLScreenReco クラス (ページ 178)の共通実装の例を参照してください。

ECLScreenReco デストラクター

ECLScreenReco のインスタンスを破棄します。

プロトタイプ

~ECLScreenReco()

パラメーター

なし

戻り値

なし

例

ECLScreenReco クラス (ページ 178)の共通実装の例を参照してください。

AddPS

モニターすべき表示スペース・オブジェクトを追加します。

プロトタイプ

AddPS(ECLPS*)

パラメーター

ECLPS*

モニター対象の PS オブジェクト。

戻り値

なし

例

ECLScreenReco クラス (ページ 178)の共通実装の例を参照してください。

IsMatch

ECLPS オブジェクトおよび ECLScreenDesc オブジェクトを渡すことができるようにし、 画面記述が PS と一致しているかどうかを判別できるようにする、静的メンバー・メソッド。これは静的メソッドとして提供されるため、どのルーチンも ECLScreenReco オブジェクトを 作成せずに、これを呼び出すことができます。

プロトタイプ

IsMatch(ECLPS*, ECLScreenDesc*)

パラメーター

ECLPS*

比較対象の ECLPS オブジェクト。

ECLScreenDesc*

比較対象の ECLScreenDesc オブジェクト。

戻り値

PS 内の画面が一致する場合は True で、それ以外の場合は False。

例

```
// set up PS
ECLPS ps = new ECLPS('A');

// set up screen description
ECLScreenDesc eclSD = new ECLScreenDesc();
eclSD.AddAttrib(0xe8, 1, 1, ColorPlane);
eclSD.AddCursorPos(23,1);
eclSD.AddNumFields(45);
eclSD.AddNumInputFields(17);
AddOIAInhibitStatus(NOTINHIBITED);
eclSD.AddString("LOGON"., 23, 11, TRUE);
eclSD.AddStringInRect("PASSWORD", 23, 1, 24, 80, FALSE);
if(ECLScreenReco::IsMatch(ps,eclSD)) {
        // Handle Screen Match here . . .
}
```

RegisterScreen

指定された画面の記述のために画面認識オブジェクトに追加された、 すべての ECLPS オブジェクトのモニターを開始します。その画面が PS に現れる と、ECLRecoNotify オブジェクト上の NotifyEvent メソッドが呼び出されます。

プロトタイプ

RegisterScreen(ECLScreenDesc*, ECLRecoNotify*)

パラメーター

ECLScreenDesc*

登録対象の画面記述オブジェクト。

ECLRecoNotify*

画面記述のためのコールバック・コードを含むオブジェクト。

戻り値

なし

例

ECLScreenReco クラス (ページ 178)の共通実装の例を参照してください。

RemovePS

画面認識モニターから ECLPS オブジェクトを取り除きます。

プロトタイプ

RemovePS(ECLPS*)

パラメーター

ECLPS*

除去する ECLPS オブジェクト。

戻り値

なし

例

ECLScreenReco クラス (ページ 178)の共通実装の例を参照してください。

UnregisterScreen

画面認識モニターから、画面記述およびそのコールバック・コードを取り除きます。

プロトタイプ

UnregisterScreen(ECLScreenDesc*)

パラメーター

ECLScreenDesc*

除去する画面記述オブジェクト。

戻り値

なし

例

ECLScreenReco クラス (ページ 178)の共通実装の例を参照してください。

ECLSession クラス

ECLSession は、 汎用エミュレーター接続関連サービスを提供し、ホスト・アクセス・クラス・ライブラリー内のその他のオブジェクトのインスタンスを指すポインターを含んでいます。

派生

ECLBase > ECLConnection > ECLSession

プロパティー

なし

使用上の注意

ECLSession は ECLConnection から派生するため、ECLConnection オブジェクトに 含まれるすべての情報を取得できます。詳しくは、ECLConnection クラス (ページ 24)を参照してください。

ECLSession に含まれるオブジェクトは単独で機能しますが、 このオブジェクトを指すポインターは ECLSession クラスに存在し ます。ECLSession オブジェクトを作成すると、ECLPS、ECLOIA、ECLXfer、 および ECLWinMetrics オブジェクトも作成されます。

ECLSession メソッド

以下のセクションでは、ECLSession クラスにおいて有効なメソッド について説明します。

ECLSession(char Name) ECLSession(Long Handle) ~ECLSession() ECLPS *GetPS() ECLOIA *GetOIA() ECLXfer *GetXfer() ECLWinMetrics *GetWinMetrics() void RegisterUpdateEvent(UPDATETYPE Type, ECLUpdateNotify *UpdateNotifyClass, BOOL InitEvent) void UnregisterUpdateEvent(ECLUpdateNotify *UpdateNotifyClass,)

ECLSession コンストラクター

このメソッドは、接続名 (単一の A から Z、または a から z の英字) または接続ハンドル から ECLSession オブジェクトを作成します。Z and I Emulator for Windows 接続でオープンできるのは、1 つの名前につき 1 つしかありません。例えば、一度に 1 つの接続 A のみがオープンできます。

プロトタイプ

ECLSession(char Name)

ECLSession(long Handle)

パラメーター

char Name

1 文字の接続の短縮名 (A から Z、または a から z)。

long Handle

ECL 接続のハンドル。

戻り値

なし

例

```
//-------------------------------------------------------------------
// ECLSession::ECLSession      (Constructor)
//
// Build PS object from name.
//-------------------------------------------------------------------
void Sample73() {

ECLSession *Sess;       // Pointer to Session object for connection A
ECLPS      *PS;         // PS object pointer

try {
  Sess = new ECLSession('A');

  PS = Sess->GetPS();
  printf("Size of presentation space is %lu.\n", PS->GetSize());

  delete Sess;
}
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}
} // end sample
```

ECLSession デストラクター

このメソッドは、ECLSession オブジェクトを破棄します。

プロトタイプ

~ECLSession();

パラメーター

なし

戻り値

なし

例

```
//-------------------------------------------------------------------
// ECLSession::~ECLSession      (Destructor)
//
// Build PS object from name and then delete it.
//-------------------------------------------------------------------
void Sample74() {

ECLSession *Sess;       // Pointer to Session object for connection A
ECLPS      *PS;         // PS object pointer

try {
  Sess = new ECLSession('A');

  PS = Sess->GetPS();
  printf("Size of presentation space is %lu.\n", PS->GetSize());

  delete Sess;
}
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}
} // end sample
```

GetPS

このメソッドは、ECLSession オブジェクト に含まれている ECLPS オブジェクトを指すポインターを戻します。このメソッドは、ECLPS オブジェクト・メソッドにアクセスするのに使用します。詳しくは、『ECLPS クラス (ページ 109)』を参照してください。

プロトタイプ

ECLPS *GetPS()

パラメーター

なし

戻り値

ECLPS *

ECLPS オブジェクト・ポインター。

例

```
//-------------------------------------------------------------------
// ECLSession::GetPS
//
// Get PS object from session object and use it.
//-------------------------------------------------------------------
void Sample69() {

ECLSession *Sess;       // Pointer to Session object for connection A
ECLPS      *PS;         // PS object pointer

try {
  Sess = new ECLSession('A');

  PS = Sess->GetPS();
  printf("Size of presentation space is %lu.\n", PS->GetSize());

  delete Sess;
}
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}
} // end sample
```

GetOIA

このメソッドは、ECLSession オブジェクトに 含まれている ECLOIA オブジェクトを指すポインターを戻します。このメソッドは、ECLOIA メソッドにアクセスするのに使用します。詳しくは、『ECLOIA クラス (ページ 93)』を参照してください。

プロトタイプ

ECLOIA *GetOIA()

パラメーター

なし

戻り値

ECLOIA *

ECLOIA オブジェクト・ポインター。

例

```
//-------------------------------------------------------------------
// ECLSession::GetOIA
//
// Get OIA object from session object and use it.
//-------------------------------------------------------------------
void Sample70() {

ECLSession *Sess;       // Pointer to Session object for connection A
ECLOIA     *OIA;        // OIA object pointer

try {
  Sess = new ECLSession('A');

  OIA = Sess->GetOIA();
  if (OIA->InputInhibited() == NotInhibited)
    printf("Input is not inhibited.\n");
  else
    printf("Input is inhibited.\n");

  delete Sess;
}
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}
} // end sample
```

GetXfer

このメソッドは、ECLSession オブジェクト に含まれている ECLXfer オブジェクトを指すポインターを戻します。このメソッドは、ECLXfer メソッドにアクセスするのに使用します。詳しくは、ECLXfer クラス (ページ 219)を参照してください。

プロトタイプ

ECLXfer *GetXfer()

パラメーター

なし

戻り値

ECLXfer *

ECLXfer オブジェクト・ポインター。

例

```
//-------------------------------------------------------------------
// ECLSession::GetXfer
//
// Get OIA object from session object and use it.
//-------------------------------------------------------------------
void Sample71() {

ECLSession *Sess;       // Pointer to Session object for connection A
ECLXfer    *Xfer;       // Xfer object pointer

try {
  Sess = new ECLSession('A');

  Xfer = Sess->GetXfer();
  Xfer->SendFile("c:\\autoexec.bat", "AUTOEXEC BAT A", "(ASCII CRLF");

  delete Sess;
}
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}
} // end sample
```

GetWinMetrics

このメソッドは、ECLSession オブジェクトに 含まれている ECLWinMetrics オブジェクトを指すポインターを戻します。このメソッドは、ECLWinMetrics メソッドにアクセスするのに使用します。詳しくは、ECLWinMetrics クラス (ページ 196)を参照してください。

プロトタイプ

ECLWinMetrics *GetWinMetrics()

パラメーター

なし

戻り値

ECLWinMetrics *

ECLWinMetrics オブジェクト・ポインター。

例

```
//-------------------------------------------------------------------
// ECLSession::GetWinMetrics
//
// Get WinMetrics object from session object and use it.
//-------------------------------------------------------------------
void Sample72() {

ECLSession *Sess;       // Pointer to Session object for connection A
ECLWinMetrics *Metrics; // WinMetrics object pointer

try {
  Sess = new ECLSession('A');

  Metrics = Sess->GetWinMetrics();
  printf("Window height is %lu pixels.\n", Metrics->GetHeight());

  delete Sess;
}
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}
} // end sample
```

GetPageSettings

このメソッドは、ECLSession オブジェクトに 含まれている ECLPageSettings オブジェクトを指すポインターを戻します。このメソッドは、ECLPageSettings メソッドにアクセスするのに使用します。詳しくは、ECLPageSettings クラス (ページ 226)を参照してください。

プロトタイプ

ECLPageSettings *GetPageSettings() const;

パラメーター

なし

戻り値

ECLPageSettings *

ECLPageSettings オブジェクト・ポインター。

例

```
//-------------------------------------------------------------------------
// ECLSession::GetPageSettings
//
// Get PageSettings object from session object and use it.
//-------------------------------------------------------------------------
void Sample124() {
   ECLSession *Sess;       // Pointer to Session object for connection A
   ECLPageSettings *PgSet; // PageSettings object pointer

   try {
      Sess = new ECLSession('A');
      PgSet = Sess->GetPageSettings();
      printf("FaceName = %s\n", PgSet->GetFontFaceName());
      delete Sess;
   }
   catch (ECLErr Err) {
      printf("ECL Error: %s\n", Err.GetMsgText());
   }
} // end sample
```

GetPrinterSettings

このメソッドは、ECLSession オブジェクト に含まれている ECLPrinterSettings オブジェクトを指すポインターを戻します。このメソッドは、ECLPrinterSettings メソッドにアクセスするのに使用します。詳しくは、ECLPageSettings クラス (ページ 226)を参照してください。

プロトタイプ

ECLPrinterSettings *GetPrinterSettings() const;

パラメーター

なし

戻り値

ECLPrinterSettings *

ECLPrinterSettings オブジェクト・ポインター。

例

```
//-------------------------------------------------------------------
// ECLSession::GetPrinterSettings
//
// Get PrinterSettings object from session object and use it.
//-------------------------------------------------------------------
void Sample125() {
   ECLSession *Sess;        // Pointer to Session object for connection A
   ECLPrinterSettings *PrSet; // PrinterSettings object pointer

   try {
      Sess = new ECLSession('A');
      PrSet = Sess->GetPrinterSettings();
      if (PrSet->IsPDTMode())
         printf("PDTMode\n");
      else
         printf("Not PDTMode\n");
      delete Sess;
   }
   catch (ECLErr Err) {
      printf("ECL Error: %s\n", Err.GetMsgText());
   }
} // end sample
```

RegisterUpdateEvent

推奨されていません。RegisterPSEvent (ページ 152)の ECLPS::RegisterPSEvent を 参照してください。

UnregisterUpdateEvent

推奨されていません。UnregisterPSEvent (ページ 154)の ECLPS::UnregisterPSEvent を 参照してください。

ECLStartNotify クラス

ECLStartNotify は、抽象基本クラスです。アプリケーションは、このクラスの インスタンスを直接作成することはできません。アプリケーションでこのクラスを使用するには、ECLStartNotify から派生した 独自のクラスを定義しなければなりません。アプリケーションは、 その派生クラス内に NotifyEvent() メンバー関数を実装しなければなりません。また、 オプションで NotifyError() および NotifyStop() メンバー関数を実装することも できます。

アプリケーションが ZIEWin 接続の開始および停止の通知を受けられるようにするには、ECLStartNotify クラスを使用します。スタート・ストップ・イベントは、任意の方法 (ECLConnMgr スタート・ストップ・メソッドを含む) で ZIEWin 接続 (ウィンドウ) が開始または停止されるたびに生成されます。

アプリケーションがスタート・ストップ・イベントの通知を受けるには、 次に示すステップを実行しなければなりません。

1. ECLStartNotify から派生したクラスを定義します。
2. その派生クラスを採用し、NotifyEvent() メンバー関数を実装します。
3. オプションで、NotifyError() または NotifyStop() 関数 (あるいはその両方) を 実装します。
4. 派生クラスのインスタンスを作成します。
5. そのインスタンスを ECLConnMgr::RegisterStartEvent() 関数で登録します。

ここに示された例は、それがどのように行われるかを例示しています。上記のステップを完了すると、その後、接続が開始または停止されるたびに アプリケーション NotifyEvent() メンバー関数が呼び出されます。この関数には、接続ハンドルを提供する 2 つのパラメーターと BOOL スタート・ストップ標識が 渡されます。アプリケーションは、他の ECL 関数の呼び出しを 含め、NotifyEvent() プロシージャーで必要な任意の関数を実行できます。アプリケーションは、接続の停止を阻止することはできないことに注意してください。通知は、セッションが停止された後に行われます。

イベントの生成時にエラーが検出された場合、ECLErr オブジェクトを 使用して NotifyError() メンバー関数が呼び出されます。エラーの特性に応じて、 エラー後にイベントが続けて生成されるかどうかが決まります。イベント生成が終了するとき (エラーか、ECLConnMgr::UnregisterStartEvent の呼び出しか、 または ECLConnMgr オブジェクトの破棄のいずれかが原因で) には、NotifyStop() メンバー関数が呼び出されます。イベント通知が終了するときには、NotifyStop() メンバー関数が常に呼び出され、 アプリケーション・オブジェクトの登録が抹消されます。

アプリケーションが NotifyError() メンバー関数の実装を行わない場合、 デフォルトの実装が使われます (単純なメッセージ・ボックスがユーザーに対して表示されます)。アプリケーションがデフォルトの振る舞いをオーバーライドするには、 アプリケーションの派生クラス内に NotifyError() 関数を実装します。同様に、アプリケーションがこの関数を提供しない場合、 デフォルトの NotifyStop() 関数が使われます (デフォルトの振る舞いでは何も行われません)。

またアプリケーションは、派生したクラス用に自身のコンストラクターおよび デストラクターを任意で提供できることに注意してください。これが便利なのは、 アプリケーションが特定のインスタンス別データをそのクラス内に保管してから、 その情報をコンストラクター上のパラメーターとして渡したい場合です。例えば、アプリケーションにおいて、スタート・ストップ・イベントが発生したら アプリケーション・ウィンドウにメッセージをポストしたい場合があります。アプリケーションは、ウィンドウ・ハンドルをグローバル変数として定義する (このハンドルを、NotifyEvent() 関数に見えるようにするため) 代わりに、 ウィンドウ・ハンドルを受け取ってクラス・メンバーのデータ域に保管する クラス用のコンストラクターとして定義することができます。

アプリケーションは、イベントを受け取るために通知オブジェクトを 登録しているかぎり、そのオブジェクトを破棄してはなりません。

実装上の制約事項: 現在、 ECLConnMgr オブジェクトでは、1 つのスタート・ストップ・イベント通知に対して 1 つしか通知オブジェクトを登録できません。その ECLConnMgr オブジェクト用に、既に 通知オブジェクトが登録されている場合、ECLConnMgr::RegisterStartEvent から エラーがスローされます。

派生

ECLBase > ECLNotify > ECLStartNotify

例

```
//---------------------------------------------------------------------
// ECLStartNotify class
//
// This sample demonstrates the use of:
//
// ECLStartNotify::NotifyEvent
// ECLStartNotify::NotifyError
// ECLStartNotify::NotifyStop
// ECLConnMgr::RegisterStartEvent
// ECLConnMgr::UnregisterStartEvent
//---------------------------------------------------------------------

//.....................................................................
// Define a class derived from ECLStartNotify
//.....................................................................
class MyStartNotify: public ECLStartNotify
{
public:
  // Define my own constructor to store instance data
  MyStartNotify(HANDLE DataHandle);

  // We have to implement this function
  void NotifyEvent(ECLConnMgr *CMObj, long ConnHandle,
                   BOOL Started);

  // We will take the default behaviour for these so we
  // don't implement them in our class:
  // void NotifyError (ECLConnMgr *CMObj, long ConnHandle, ECLErr ErrObject);
  // void NotifyStop (ECLConnMgr *CMObj, int Reason);
```

```
private:
  // We will store our application data handle here
  HANDLE MyDataH;
};

 //.....................................................................
MyStartNotify::MyStartNotify(HANDLE DataHandle)   // Constructor
//.....................................................................
{
  MyDataH = DataHandle;  // Save data handle for later use
}

//.....................................................................
void MyStartNotify::NotifyEvent(ECLConnMgr *CMObj, long ConnHandle,
                   BOOL Started)
//.....................................................................
{
  // This function is called whenever a connection start or stops.

  if (Started)
    printf("Connection %c started.\n", CMObj->ConvertHandle2ShortName(ConnHandle));
  else
    printf("Connection %c stopped.\n", CMObj->ConvertHandle2ShortName(ConnHandle));

  return;
```

```
}

 //.............................................................
// Create the class and begin start/stop monitoring.
//.............................................................
void Sample75() {

ECLConnMgr    CMgr;     // Connection manager object
MyStartNotify *Event;   // Ptr to my event handling object
HANDLE InstData;      // Handle to application data block (for example)

try {
  Event = new MyStartNotify(InstData);  // Create event handler

  CMgr.RegisterStartEvent(Event);       // Register to get events
```

```
  // At this point, any connection start/stops will cause the
  // MyStartEvent::NotifyEvent() function to execute.  For
  // this sample, we put this thread to sleep during this
  // time.

  printf("Monitoring connection start/stops for 60 seconds...\n");
  Sleep(60000);

  // Now stop event generation.
  CMgr.UnregisterStartEvent(Event);
  printf("Start/stop monitoring ended.\n");

  delete Event;  // Don't delete until after unregister!
}
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}

} // end sample
```

ECLStartNotify メソッド

以下のセクションでは、ECLStartNotify クラスにおいて有効なメソッドについて説明します。

ECLStartNotfiy() ECLStartNotify() virtual int NotifyEvent (ECLConnMgr *CMObj, long ConnHandle, BOOL Started) = 0 virtual void NotifyError (ECLConnMgr *CMObj, long ConnHandle, ECLErr ErrObject) virtual void NotifyStop (ECLConnMgr *CMObj int Reason)

NotifyEvent

このメソッドは、純粋仮想メンバー関数です (アプリケーションは ECLStartNotify から派生したクラス内にこの関数を実装しなければなりません)。*must* 接続の開始または停止のときと、スタート・ストップ・イベントのためにオブ

ジェクトが登録されたときは、 常にこの関数が呼び出されます。接続が開始される場合は Started BOOL が True、停止されている場合は False になります。

プロトタイプ

virtual int NotifyEvent (ECLConnMgr *CMObj, long ConnHandle, BOOL Started) = 0

パラメーター

ECLConnMgr *CMObj

これは、イベントが発生した ECLConnMgr オブジェクトを指すポインターです。

long ConnHandle

これは、開始または停止した接続のハンドルです。

BOOL Started

接続が開始される場合は True、接続が停止される場合は False になります。

戻り値

なし

NotifyError

このメソッドは、ECLConnMgr オブジェクトがエラー・イベント生成を 検出するたびに呼び出されます。エラー・オブジェクトには、そのエラーについての情報が含まれます (ECLErr クラスの説明を参照)。エラーの特性に応じて、エラー後にイベントが続けて生成されることがあります。エラーが原因でイベント生成が停止した場合、NotifyStop() 関数が 呼び出されます。

ConnHandle には、エラーに関連した接続のハンドルが含まれます。エラーが特定の接続に関連していない場合、この値はゼロになります。

アプリケーションは、この関数を実装するか、 または ECLStartNotify 基本クラスにエラーを処理させるかを選ぶことができます。基本クラスは、ECLErr::GetMsgText() 関数から提供される テキストを使用して、メッセージ・ボックスにエラーを表示します。アプリケーションが、その派生クラス内にこの関数を実装すると、 それによって、基本クラス関数がオーバーライドされます。

プロトタイプ

virtual void NotifyError (ECLConnMgr *CMObj, long ConnHandle, ECLErr ErrObject)

パラメーター

ECLConnMgr *CMObj

これは、エラーが発生した ECLConnMgr オブジェクトを指すポインターです。

long ConnHandle

これは、エラーに関連した接続のハンドルまたはゼロです。

ECLErr ErrObject

これは、エラーを記述した ECLErr オブジェクトです。

戻り値

なし

NotifyStop

イベント生成が何らかの理由 (例えば、エラー条件が原因か、 または ECLConnMgr::UnregisterStartEvent の呼び出しなどが原因) で停止すると、 このメソッドが呼び出されます。

プロトタイプ

virtual void NotifyStop (ECLConnMgr *CMObj int Reason)

パラメーター

ECLConnMgr *CMObj

これは、通知を停止した ECLConnMgr オブジェクトを指すポインターです。

int Reason

これは、未使用のゼロです。

戻り値

なし

ECLUpdateNotify クラス

推奨されていません。ECLPSListener クラス (ページ 159)および ECLOIA クラス (ページ 93)の クラスの説明を参照してください。

ECLWinMetrics クラス

ECLWinMetrics クラスは、Z and I Emulator for Windows の接続ウィンドウの処理を行います。このクラスは、ウィンドウ長方形および位置の操作 (例えば、 SetWindowRect、GetXpos、または SetWidth) を、ウィンドウの状態の操作と 同じように実行できます。

派生

ECLBase > ECLConnection > ECLWinMetrics

プロパティー

なし

使用上の注意

ECLWinMetrics は ECLConnection から派生するため、ECLConnection オブジェクトに 含まれるすべての情報を取得できます。詳しくは、ECLConnection クラス (ページ 24)を参照してください。

ECLWinMetrics オブジェクトは、作成時に識別された接続用に作成されます。ECLWinMetrics オブジェクトを作成するには、通常は ECLConnection オブジェクトから 取得される接続 ID (単一の A から Z、または a から z の英字) または接続ハンドルを渡します。1 つの名前またはハンドルに対して、一度に 1 つの Z and I Emulator for Windows 接続しかオープンできません。

注: ECLSession クラス内に、ECLWinMetrics オブジェクトを指す ポインターがあります。接続ウィンドウだけを操作したい場合 は、ECLWinMetrics を単独で作成します。他にも処理したいものがある場合は、ECLSession オブジェクトを作成することができます。

ECLWinMetrics メソッド

次に示すメソッドは、ECLWinMetrics クラスに適用されます。

ECLWinMetrics(char Name) ECLWinMetrics(long Handle) ~ECLWinMetrics() const char *GetWindowTitle() void SetWindowTitle(char *NewTitle) long GetXpos() void SetXpos(long NewXpos) long GetYpos() void SetYpos(long NewYpos) long GetWidth() void SetWidth(long NewWidth) long GetHeight() void SetHeight(long NewHeight) void GetWindowRect(Long *left, Long *top, Long *right, Long *bottom) void SetWindowRect(Long left, Long top, Long right, Long bottom) BOOL IsVisible() void SetVisible(BOOL SetFlag) BOOL Active() void SetActive(BOOL SetFlag) BOOL IsMinimized() void SetMinimized() BOOL IsMaximized() void SetMaximized() BOOL IsRestored() void SetRestored()

ECLWinMetrics コンストラクター

このメソッドは、接続名または接続ハンドル から ECLWinMetrics オブジェクトを作成します。Z and I Emulator for Windows 接続でオープンできるのは、1 つの名前につき 1 つしかありません。例えば、一度に 1 つの接続 A のみをオープンできます。

プロトタイプ

ECLWinMetrics(char Name)

ECLWinMetrics(long Handle)

パラメーター

char Name

1 文字の接続の短縮名 (A から Z、または a から z)。

long Handle

ECL 接続のハンドル。

戻り値

なし

例

```
//-------------------------------------------------------------------
// ECLWinMetrics::ECLWinMetrics   (Constructor)
//
// Build WinMetrics object from name.
//-------------------------------------------------------------------
void Sample77() {

ECLWinMetrics *Metrics;    // Ptr to object

try {
  Metrics = new ECLWinMetrics('A');  // Create for connection A

  printf("Window of connection A is %lu pixels wide.\n",
    Metrics->GetWidth());

  delete Metrics;
}
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}

} // end sample
```

ECLWinMetrics デストラクター

このメソッドは、ECLWinMetrics オブジェクトを破棄します。

プロトタイプ

~ECLWinMetrics()

パラメーター

なし

戻り値

なし

例

```
//---------------------------------------------------------------------
// ECLWinMetrics::ECLWinMetrics   (Destructor)
//
// Build WinMetrics object from name.
//---------------------------------------------------------------------
void Sample78() {

ECLWinMetrics *Metrics;    // Ptr to object

try {
  Metrics = new ECLWinMetrics('A');  // Create for connection A

  printf("Window of connection A is %lu pixels wide.\n",
    Metrics->GetWidth());

  delete Metrics;
}
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}

} // end sample
```

GetWindowTitle

GetWindowTitle メソッドは、ECLWinMetrics オブジェクト に関連した接続用のタイトル・バーに現在あるタイトルの入ったヌル終了ストリングを 指すポインターを戻します。戻されたストリングは、時間が経過しても持続すると想定してはなりません。そのストリングのコピーを作成するか、または必要があるたびに このメソッドを呼び出さなければなりません。

プロトタイプ

const char *GetWindowTitle()

パラメーター

なし

戻り値

タイトルの入ったヌル終了ストリングを指すポインター。

例

```
//-------------------------------------------------------------------
// ECLWinMetrics::GetWindowTitle
//
// Display current window title of connection A.
//-------------------------------------------------------------------
void Sample79() {

ECLWinMetrics *Metrics;    // Ptr to object

try {
  Metrics = new ECLWinMetrics('A');  // Create for connection A

  printf("Title of connection A is: %s\n",
    Metrics->GetWindowTitle());

  delete Metrics;
}
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}

} // end sample
```

SetWindowTitle

SetWindowTitle メソッドは、ECLWinMetrics オブジェクト に関連した接続用のタイトル・バーに現在あるタイトルを、 入力パラメーターに入れて渡されたタイトルに変更します。タイトルをデフォルトのタイトルにリセットするために、null ストリングを 使用することができます。

プロトタイプ

void SetWindowTitle(char *NewTitle)

パラメーター

char *NewTitle

ヌル終了のタイトル・ストリング。

戻り値

なし

例

```
//-------------------------------------------------------------------
// ECLWinMetrics::SetWindowTitle
//
// Change current window title of connection A.
//-------------------------------------------------------------------
void Sample80() {

ECLWinMetrics *Metrics;    // Ptr to object

try {
  Metrics = new ECLWinMetrics('A');  // Create for connection A

  // Get current title
  printf("Title of connection A is: %s\n", Metrics->GetWindowTitle());

  // Set new title
  Metrics->SetWindowTitle("New Title");
  printf("New title is: %s\n", Metrics->GetWindowTitle());

  // Reset back to original title
  Metrics->SetWindowTitle("");
  printf("Returned title to: %s\n", Metrics->GetWindowTitle());

  delete Metrics;
}
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}

} // end sample
```

使用上の注意

NewTitle がヌル・ストリングの場合、SetWindowTitle は ウィンドウ・タイトルを元の設定値に復元します。

GetXpos

GetXpos メソッドは、接続ウィンドウ長方形の左上の角の *x* 位置を戻します。

プロトタイプ

long GetXpos()

パラメーター

なし

戻り値

long

接続ウィンドウの *x* 位置。

例

```
//-------------------------------------------------------------------
// ECLWinMetrics::GetXpos
//
// Move window 10 pixels.
//-------------------------------------------------------------------
void Sample81() {

ECLWinMetrics *Metrics;    // Ptr to object
long X, Y;

try {
  Metrics = new ECLWinMetrics('A');  // Create for connection A

  if (Metrics->IsMinimized() || Metrics->IsMaximized()) {
    printf("Cannot move minimized or maximized window.\n");
  }
  else {
    X = Metrics->GetXpos();
    Y = Metrics->GetYpos();
    Metrics->SetXpos(X+10);
    Metrics->SetYpos(Y+10);
  }

  delete Metrics;
}
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}

} // end sample
```

SetXpos

SetXpos メソッドは、接続ウィンドウ長方形の左上の角の *x* 位置を設定します。

プロトタイプ

void SetXpos(long NewXpos)

パラメーター

long NewXpos

ウィンドウ長方形の新しい *x* 座標。

戻り値

なし

例

```
//-------------------------------------------------------------------
// ECLWinMetrics::SetXpos
//
// Move window 10 pixels.
//-------------------------------------------------------------------
void Sample83() {

ECLWinMetrics *Metrics;    // Ptr to object
long X, Y;

try {
  Metrics = new ECLWinMetrics('A');  // Create for connection A

  if (Metrics->IsMinimized() || Metrics->IsMaximized()) {
    printf("Cannot move minimized or maximized window.\n");
  }
  else {
    X = Metrics->GetXpos();
    Y = Metrics->GetYpos();
    Metrics->SetXpos(X+10);
    Metrics->SetYpos(Y+10);
  }

  delete Metrics;
}
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}

} // end sample
```

GetYpos

GetYpos メソッドは、接続ウィンドウ長方形の左上の角の *y* 位置を戻します。

プロトタイプ

long GetYpos()

パラメーター

なし

戻り値

long

接続ウィンドウの *y* 座標。

例

```
a//--------------------------------------------------------------------
// ECLWinMetrics::GetYpos
//
// Move window 10 pixels.
//--------------------------------------------------------------------
void Sample82() {

ECLWinMetrics *Metrics;    // Ptr to object
long X, Y;

try {
  Metrics = new ECLWinMetrics('A');  // Create for connection A

  if (Metrics->IsMinimized() || Metrics->IsMaximized()) {
    printf("Cannot move minimized or maximized window.\n");
  }
  else {
    X = Metrics->GetXpos();
    Y = Metrics->GetYpos();
    Metrics->SetXpos(X+10);
    Metrics->SetYpos(Y+10);
  }

  delete Metrics;
}
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}

} // end sample
```

SetYpos

SetYpos メソッドは、接続ウィンドウ長方形の左上の角の *y* 位置を設定します。

プロトタイプ

void SetYpos(long NewYpos)

パラメーター

long NewYpos

ウィンドウ長方形の新しい *y* 座標。

戻り値

なし

例

```
//-------------------------------------------------------------------
// ECLWinMetrics::SetYpos
//
// Move window 10 pixels.
//-------------------------------------------------------------------
void Sample84() {

ECLWinMetrics *Metrics;    // Ptr to object
long X, Y;

try {
  Metrics = new ECLWinMetrics('A');  // Create for connection A

  if (Metrics->IsMinimized() || Metrics->IsMaximized()) {
    printf("Cannot move minimized or maximized window.\n");
  }
  else {
    X = Metrics->GetXpos();
    Y = Metrics->GetYpos();
    Metrics->SetXpos(X+10);
    Metrics->SetYpos(Y+10);
  }

  delete Metrics;
}
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}

} // end sample
```

GetWidth

このメソッドは、接続ウィンドウ長方形の幅を戻します。

プロトタイプ

long GetWidth()

パラメーター

なし

戻り値

long

接続ウィンドウの幅。

例

```
//---------------------------------------------------------------------
// ECLWinMetrics::GetWidth
//
// Make window 1/2 its current size.  Depending on display settings
// (Appearance->Display Setup menu) it may snap to a font that is
// not exactly the 1/2 size we specify.
//---------------------------------------------------------------------
void Sample85() {

ECLWinMetrics *Metrics;    // Ptr to object
long X, Y;

try {
  Metrics = new ECLWinMetrics('A');  // Create for connection A

  if (Metrics->IsMinimized() || Metrics->IsMaximized()) {
    printf("Cannot size minimized or maximized window.\n");
  }
  else {
    X = Metrics->GetWidth();
    Y = Metrics->GetHeight();
    Metrics->SetWidth(X/2);
    Metrics->SetHeight(Y/2);
  }

  delete Metrics;
}
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}

} // end sample
```

SetWidth

SetWidth メソッドは、接続ウィンドウ長方形の幅を設定します。

プロトタイプ

void SetWidth(long NewWidth)

パラメーター

long NewWidth

ウィンドウ長方形の新しい幅。

戻り値

なし

例

```
//---------------------------------------------------------------------
// ECLWinMetrics::SetWidth
//
// Make window 1/2 its current size.  Depending on display settings
// (Appearance->Display Setup menu) it may snap to a font that is
// not exactly the 1/2 size we specify.
//---------------------------------------------------------------------
void Sample87() {

ECLWinMetrics *Metrics;    // Ptr to object
long X, Y;

try {
  Metrics = new ECLWinMetrics('A');  // Create for connection A

  if (Metrics->IsMinimized() || Metrics->IsMaximized()) {
    printf("Cannot size minimized or maximized window.\n");
  }
  else {
    X = Metrics->GetWidth();
    Y = Metrics->GetHeight();
    Metrics->SetWidth(X/2);
    Metrics->SetHeight(Y/2);
  }

  delete Metrics;
}
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}

} // end sample
```

GetHeight

GetHeight メソッドは、接続ウィンドウ長方形の高さを戻します。

プロトタイプ

long GetHeight()

パラメーター

なし

戻り値

long

接続ウィンドウの高さ。

例

```
//-------------------------------------------------------------------
// ECLWinMetrics::GetHeight
//
// Make window 1/2 its current size.  Depending on display settings
// (Appearance->Display Setup menu) it may snap to a font that is
// not exactly the 1/2 size we specify.
//-------------------------------------------------------------------
void Sample86() {

ECLWinMetrics *Metrics;    // Ptr to object
long X, Y;

try {
  Metrics = new ECLWinMetrics('A');  // Create for connection A

  if (Metrics->IsMinimized() || Metrics->IsMaximized()) {
    printf("Cannot size minimized or maximized window.\n");
  }
  else {
    X = Metrics->GetWidth();
    Y = Metrics->GetHeight();
    Metrics->SetWidth(X/2);
    Metrics->SetHeight(Y/2);
  }

  delete Metrics;
}
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}

} // end sample
```

SetHeight

このメソッドは、接続ウィンドウ長方形の高さを設定します。

プロトタイプ

void SetHeight(Long NewHeight)

パラメーター

long NewHeight

ウィンドウ長方形の新しい高さ。

戻り値

なし

例

以下の例は、SetHeight メソッドを使用して接続ウィンドウ長方形の 高さを設定する方法を示します。

```
ECLWinMetrics   *pWM;
ECLConnList  ConnList();

// Create using connection handle of first connection in the list of
// active connections
try {
  if ( ConnList.Count() != 0 ) {
    pWM = new ECLWinMetrics(ConnList.GetFirstSession()->GetHandle());

    // Set the height
    pWM->SetHeight(6081);
  }
}
catch (ECLErr ErrObj) {
  // Just report the error text in a message box
  MessageBox( NULL, ErrObj.GetMsgText(), "Error!", MB_OK );
}
```

GetWindowRect

このメソッドは、接続ウィンドウ長方形の境界点を戻します。

プロトタイプ

void GetWindowRect(Long *left, Long *top, Long *right, Long *bottom)

パラメーター

long *left

この出力パラメーターは、ウィンドウ長方形の左の座標に設定されます。

long *top

この出力パラメーターは、ウィンドウ長方形の上端の座標に設定されます。

long *right

この出力パラメーターは、ウィンドウ長方形の右の座標に設定されます。

long *bottom

この出力パラメーターは、ウィンドウ長方形の下端の座標に設定されます。

戻り値

なし

例

```
//-------------------------------------------------------------------
// ECLWinMetrics::GetWindowRect
//
// Make window 1/2 its current size.  Depending on display settings
// (Appearance->Display Setup menu) it may snap to a font that is
// not exactly the 1/2 size we specify.  Also move the window.
//-------------------------------------------------------------------
void Sample88() {

ECLWinMetrics *Metrics;    // Ptr to object
long X, Y, Width, Height;

try {
  Metrics = new ECLWinMetrics('A');  // Create for connection A

  if (Metrics->IsMinimized() || Metrics->IsMaximized()) {
    printf("Cannot size/move minimized or maximized window.\n");
  }
  else {
    Metrics->GetWindowRect(&X, &Y, &Width, &Height);
    Metrics->SetWindowRect(X+10, Y+10,         // Move window
                           Width/2, Height/2); // Size window
  }

  delete Metrics;
}
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}

} // end sample
```

SetWindowRect

このメソッドは、接続ウィンドウ長方形の境界点を設定します。

プロトタイプ

void SetWindowRect(long left, long top, long right, long bottom)

パラメーター

long left

ウィンドウ長方形の左の座標。

long top

ウィンドウ長方形の上端の座標。

long right

ウィンドウ長方形の右の座標。

long bottom

ウィンドウ長方形の下端の座標。

戻り値

なし

例

```
//---------------------------------------------------------------------
// ECLWinMetrics::SetWindowRect
//
// Make window 1/2 its current size.  Depending on display settings
// (Appearance->Display Setup menu) it may snap to a font that is
// not exactly the 1/2 size we specify.  Also move the window.
//---------------------------------------------------------------------
void Sample89() {

ECLWinMetrics *Metrics;    // Ptr to object
long X, Y, Width, Height;

try {
  Metrics = new ECLWinMetrics('A');  // Create for connection A

  if (Metrics->IsMinimized() || Metrics->IsMaximized()) {
    printf("Cannot size/move minimized or maximized window.\n");
  }
  else {
    Metrics->GetWindowRect(&X, &Y, &Width, &Height);
    Metrics->SetWindowRect(X+10, Y+10,         // Move window
                           Width/2, Height/2); // Size window
  }

  delete Metrics;
}
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
```

```
}

} // end sample
```

IsVisible

このメソッドは、接続ウィンドウの可視状態を戻します。

プロトタイプ

BOOL IsVisible()

パラメーター

なし

戻り値

可視状態。ウィンドウが可視の場合は True 値、ウィンドウが不可視の場合は False 値になります。

例

```
//---------------------------------------------------------------------
// ECLWinMetrics::IsVisible
//
// Get current state of window, and then toggle it.
//---------------------------------------------------------------------
void Sample90() {

ECLWinMetrics Metrics('A');      // Window metrics class
BOOL  CurrState;

CurrState = Metrics.IsVisible(); // Get state
Metrics.SetVisible(!CurrState);  // Set state

} // end sample
```

SetVisible

このメソッドは、接続ウィンドウの可視状態を設定します。

プロトタイプ

void SetVisible(BOOL SetFlag)

パラメーター

BOOL SetFlag

可視の場合は True、不可視の場合は False。

戻り値

なし

例

```
//---------------------------------------------------------------------
// ECLWinMetrics::SetVisible
//
// Get current state of window, and then toggle it.
//---------------------------------------------------------------------
void Sample91() {

ECLWinMetrics Metrics('A');      // Window metrics class
BOOL  CurrState;

CurrState = Metrics.IsVisible(); // Get state
Metrics.SetVisible(!CurrState);  // Set state

} // end sample

//---------------------------------------------------------------------
```

IsActive

このメソッドは、接続ウィンドウのフォーカス状態を戻します。

プロトタイプ

BOOL Active()

パラメーター

なし

戻り値

BOOL

フォーカス状態。アクティブの場合は True、非アクティブの場合は False。

例

```

// ECLWinMetrics::IsActive
```

```
//
// Get current state of window, and then toggle it.
//---------------------------------------------------------------------
void Sample92() {

ECLWinMetrics Metrics('A');      // Window metrics class
BOOL  CurrState;

CurrState = Metrics.IsActive();  // Get state
Metrics.SetActive(!CurrState);   // Set state

} // end sample
```

SetActive

このメソッドは、接続ウィンドウのフォーカス状態を設定します。

プロトタイプ

void SetActive(BOOL SetFlag)

パラメーター

Bool SetFlag

新しい状態。アクティブの場合は True、非アクティブの場合は False。

戻り値

なし

例

以下に、SetActive メソッドの例を示します。

```
ECLWinMetrics *pWM;
ECLConnList  ConnList();

// Create using connection handle of first connection in the list of
// active connections
try {
  if ( ConnList.Count() != 0 ) {
    pWM = new ECLWinMetrics(ConnList.GetFirstSession()->GetHandle());

    // Set to inactive if active
    if ( pWM->Active() )
   pWM->SetActive(FALSE);
  }
}
catch (ECLErr ErrObj) {
  // Just report the error text in a message box
```

```
  MessageBox( NULL, ErrObj.GetMsgText(), "Error!", MB_OK );
}
```

IsMinimized

このメソッドは、接続ウィンドウの最小化状態を戻します。

プロトタイプ

BOOL IsMinimized()

パラメーター

なし

戻り値

BOOL

最小化状態。ウィンドウが最小化の場合は True 値、 ウィンドウが最小化でない場合は False 値になります。

例

```
//---------------------------------------------------------------------
// ECLWinMetrics::IsMinimized
//
// Get current state of window, and then toggle it.
//---------------------------------------------------------------------
void Sample93() {

ECLWinMetrics Metrics('A');      // Window metrics class
BOOL  CurrState;

CurrState = Metrics.IsMinimized();  // Get state
if (!CurrState)
  Metrics.SetMinimized();           // Set state
else
  Metrics.SetRestored();

} // end sample
```

SetMinimized

このメソッドは、接続ウィンドウを最小化に設定します。

プロトタイプ

void SetMinimized()

パラメーター

なし

戻り値

なし

例

```
//---------------------------------------------------------------------
// ECLWinMetrics::SetMinimized
//
// Get current state of window, and then toggle it.
//---------------------------------------------------------------------
void Sample94() {

ECLWinMetrics Metrics('A');      // Window metrics class
BOOL  CurrState;

CurrState = Metrics.IsMinimized();  // Get state
if (!CurrState)
  Metrics.SetMinimized();           // Set state
else
  Metrics.SetRestored();

} // end sample
```

IsMaximized

このメソッドは、接続ウィンドウの最大化状態を戻します。

プロトタイプ

BOOL IsMaximized()

パラメーター

なし

戻り値

BOOL

最大化状態。ウィンドウが最大化の場合は True 値、 ウィンドウが最大化でない場合は False 値になります。

例

```
// ECLWinMetrics::IsMaximized
//
// Get current state of window, and then toggle it.
//-------------------------------------------------------------------
void Sample97() {

ECLWinMetrics Metrics('A');      // Window metrics class
BOOL  CurrState;

CurrState = Metrics.IsMaximized();  // Get state
if (!CurrState)
  Metrics.SetMaximized();           // Set state
else
  Metrics.SetMinimized();

} // end sample
```

SetMaximized

このメソッドは、接続ウィンドウを最大化に設定します。

プロトタイプ

void SetMaximized()

パラメーター

なし

戻り値

なし

例

```
//-------------------------------------------------------------------
// ECLWinMetrics::SetMaximized
//
// Get current state of window, and then toggle it.
//-------------------------------------------------------------------
```

```
void Sample98() {

ECLWinMetrics Metrics('A');      // Window metrics class
BOOL  CurrState;

CurrState = Metrics.IsMaximized();  // Get state
if (!CurrState)
  Metrics.SetMaximized();           // Set state
else
  Metrics.SetMinimized();

} // end sample
```

IsRestored

このメソッドは、接続ウィンドウの復元状態を戻します。

プロトタイプ

BOOL IsRestored()

パラメーター

なし

戻り値

BOOL

復元状態。ウィンドウが復元されている場合は True 値、 ウィンドウが復元されていない場合は False 値になります。

例

```
//---------------------------------------------------------------------
// ECLWinMetrics::IsRestored
//
// Get current state of window, and then toggle it.
//---------------------------------------------------------------------
void Sample95() {

ECLWinMetrics Metrics('A');      // Window metrics class
BOOL  CurrState;

CurrState = Metrics.IsRestored();  // Get state
if (!CurrState)
  Metrics.SetRestored();           // Set state
else
  Metrics.SetMinimized();

} // end sample
```

```
```

SetRestored

SetRestored メソッドは、接続ウィンドウを 復元済みに設定します。

プロトタイプ

void SetRestored()

パラメーター

なし

戻り値

なし

例

```
//----------------------------------------------------------------------
// ECLWinMetrics::SetRestored
//
// Get current state of window, and then toggle it.
//----------------------------------------------------------------------
void Sample96() {

ECLWinMetrics Metrics('A');      // Window metrics class
BOOL  CurrState;

CurrState = Metrics.IsRestored();  // Get state
if (!CurrState)
  Metrics.SetRestored();           // Set state
else
  Metrics.SetMinimized();

} // end sample

//----------------------------------------------------------------------
```

ECLXfer クラス

ECLXfer は、ファイル転送サービスを提供します。

派生

ECLBase > ECLConnection > ECLXfer

プロパティー

なし

使用上の注意

ECLXfer は ECLConnection から派生するため、ECLConnection オブジェクトに 含まれるすべての情報を取得できます。詳しくは、ECLConnection クラス (ページ 24)を参照してください。

ECLXfer オブジェクトは、作成時に識別された接続用に作成されます。ECLXfer オブジェクトを作成するには、 通常は ECLConnList オブジェクトから取得される接続 ID (単一の A から Z、または a から z の英字) または接続ハンドルを 渡します。1 つの名前またはハンドルに対して、一度に 1 つの Z and I Emulator for Windows 接続しかオープンできません。

注: ECLSession クラス内に、ECLXfer オブジェクトを指すポインターがあります。接続ウィンドウだけを操作したい場合は、ECLXfer オブジェクトを単独で作成します。他にも処理したいものがある場合は、ECLSession オブジェクトを作成することができます。

ECLXfer メソッド

以下のセクションでは、ECLXfer クラスで有効なメソッドを説明します。

ECLXfer(char Name) ECLXfer(long Handle) ~ECLXfer() int SendFile(char *PCFile, char *HostFile, char *Options) int ReceiveFile(char *PCFile, char *HostFile, char *Options)

ECLXfer コンストラクター

このメソッドは、接続名 (単一の A から Z、または a から z の英字) または接続ハンドル から ECLXfer オブジェクトを作成します。Z and I Emulator for Windows 接続は、1 つの ID につき 1 つしかありません。例えば、一度に 1 つの接続 A のみをオープンできます。

プロトタイプ

ECLXfer(char Name)

ECLXfer(long Handle)

パラメーター

char Name

1 文字の接続の短縮名 (A から Z、または a から z)。

long Handle

ECL 接続のハンドル。

戻り値

なし

例

```
//---------------------------------------------------------------------
// ECLXfer::ECLXfer          (Constructor)
//
// Build ECLXfer object from a connection name.
//---------------------------------------------------------------------
void Sample99() {

ECLXfer  *Xfer;             // Pointer to Xfer object

try {
  Xfer = new ECLXfer('A');  // Create object for connection A
  printf("Created ECLXfer for connection %c.\n", Xfer->GetName());

  delete Xfer;              // Delete Xfer object
}
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}

} // end sample
```

ECLXfer デストラクター

このメソッドは、ECLXfer オブジェクトを破棄します。

プロトタイプ

~ECLXfer();

パラメーター

なし

戻り値

なし

例

```
//-------------------------------------------------------------------
// ECLXfer::~ECLXfer          (Destructor)
//
// Build ECLXfer object from a connection name.
//-------------------------------------------------------------------
void Sample100() {

ECLXfer  *Xfer;             // Pointer to Xfer object

try {
  Xfer = new ECLXfer('A');  // Create object for connection A
  printf("Created ECLXfer for connection %c.\n", Xfer->GetName());

  delete Xfer;              // Delete Xfer object
}
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}

} // end sample
```

SendFile

このメソッドは、 ファイルをワークステーションからホストに送信します。

プロトタイプ

int SendFile(char *PCFile, char *HostFile, char *Options)

パラメーター

char *PCFile

ホストに送るワークステーション・ファイル名を含んだストリングを指すポインター。

char *HostFile

ホストで作成または更新するホスト・ファイル名を含んだストリングを指すポインター。

char *Options

転送時に使用するオプションを含んだストリングを指すポインター。

戻り値

整数

SendFile EHLLAPI 関数に関して「*Emulator Programming*」に記載されている、 EHLLAPI 戻りコード。

例

```
//-------------------------------------------------------------------
// ECLXfer::SendFile
//
// Send a file to a VM/CMS host with ASCII translation.
//-------------------------------------------------------------------
void Sample101() {

ECLXfer  *Xfer;             // Pointer to Xfer object
int Rc;

try {
  Xfer = new ECLXfer('A');  // Create object for connection A

  printf("Sending file...\n");
  Rc = Xfer->SendFile("c:\\autoexec.bat", "autoexec bat a", "(ASCII CRLF QUIET");
  switch (Rc) {
  case 2:
    printf("File transfer failed, error in parameters.\n", Rc);
    break;
  case 3:
    printf("File transfer sucessfull.\n");
    break;
  case 4:
    printf("File transfer sucessfull, some records were segmented.\n");
    break;
  case 5:
    printf("File transfer failed, workstation file not found.\n");
    break;
  case 27:
    printf("File transfer cancelled or timed out.\n");
    break;
  default:
    printf("File transfer failed, code %u.\n", Rc);
    break;
  } // case

  delete Xfer;              // Delete Xfer object
}
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}

} // end sample
```

使用上の注意

ファイル転送オプションは、ホストに応じて異なります。以下に示すのは、VM/CMS ホストの有効なホスト・オプションの一部です。

ASCII
CRLF
APPEND
LRECL
RECFM
CLEAR/NOCLEAR
PROGRESS
QUIET

サポートされるホストおよび関連ファイル転送オプションのリストについては、「*Emulator Programming*」を 参照してください。

ReceiveFile

このメソッドは、ホストからファイルを受け取って、 ファイルをワークステーションに送ります。

プロトタイプ

int ReceiveFile(char *PCFile, char *HostFile, char *Options)

パラメーター

char *PCFile

ホストに送るワークステーション・ファイル名を含んだストリングを指すポインター。

char *HostFile

ホストで作成または更新するホスト・ファイル名を含んだストリングを指すポインター。

char *Options

転送時に使用するオプションを含んだストリングを指すポインター。

戻り値

整数

ReceiveFile EHLLAPI 関数に関して「*Emulator Programming*」に記載されている、 EHLLAPI 戻りコード。

例

```
//-------------------------------------------------------------------
// ECLXfer::ReceiveFile
//
// Receive file from a VM/CMS host with ASCII translation.
//-------------------------------------------------------------------
void Sample102() {

ECLXfer  *Xfer;             // Pointer to Xfer object
int Rc;

try {
  Xfer = new ECLXfer('A');  // Create object for connection A

  printf("Receiving file...\n");
  Rc = Xfer->ReceiveFile("c:\\temp.txt", "temp text a", "(ASCII CRLF QUIET");
  switch (Rc) {
  case 2:
    printf("File transfer failed, error in parameters.\n", Rc);
    break;
  case 3:
    printf("File transfer sucessfull.\n");
    break;
  case 4:
    printf("File transfer sucessfull, some records were segmented.\n");
    break;
  case 27:
    printf("File transfer cancelled or timed out.\n");
    break;
  default:
    printf("File transfer failed, code %u.\n", Rc);
    break;
  } // case

  delete Xfer;              // Delete Xfer object
}
catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}

} // end sample
```

使用上の注意

ファイル転送オプションは、ホストに応じて異なります。以下に示すのは、VM/CMS ホストの有効なホスト・オプションの一部です。

ASCII
CRLF
APPEND
LRECL
RECFM

CLEAR/NOCLEAR
PROGRESS
QUIET

サポートされるホストおよび関連ファイル転送オプションのリストについては、「*Emulator Programming*」を 参照してください。

ECLPageSettings クラス

ECLPageSettings クラスは、セッションのページ設定での操作を実行します。ここでは、「CPI」、「LPI」、および「書体名」などの**「ファイル」**→**「ページ設定」**ダイアログ設定を検索および構成できます。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。

派生

ECLBase > ECLConnection > ECLPageSettings

プロパティー

なし

制約事項

各メソッドに関連する接続は、メソッドを成功させるために 特定の状態になければなりません。制約事項を満たしていないと、 該当する例外が引き起こされます。

ECLPageSettings クラスのメソッドが呼び出されるときには、 以下の制約事項が適用されます。制約事項を満たしていないと、 例外がスローされます。

- 接続の**「ページ設定」**および**「プリンター設定」**ダイアログが使用中ではない。
- 接続が印刷中ではない。
- 関連する接続が PDT モードではない。

特定のメソッドに対して追加の制約事項が適用される場合があります。

使用上の注意

ECLPageSettings は ECLConnection から派生するため、ECLConnection オブジェクトに含まれるすべての情報を取得できます。詳しくは、ECLConnection クラス (ページ 24)を参照してください。

ECLPageSettings オブジェクトは、作成時に識別された接続用に作成されます。ECLPageSettings オブジェクトを作成するには、接続 ID (**A** から **Z** の単一の英字) または接続ハンドル (通常は ECLConnection オブジェクトから取得)

を渡します。1 つの名前またはハンドルに対して、一度に 1 つの Z and I Emulator for Windows 接続しかオープンできません。

ECLSession クラスは、このオブジェクトのインスタンスを作成します。ECLSession が提供する他のサービスをアプリケーションで必要としない場合、 このオブジェクトを単独で作成しても構いません。それ以外の場合、ECLSession オブジェクトを作成することを検討し、ECLSession によって作成されたオブジェクトを使用してください。詳しくは、ECLSession クラス (ページ 183)を参照してください。

各メソッドは、ECLPageSettings オブジェクトに関連する接続の特定の接続タイプのみをサポートします。サポートされる接続タイプについては、各メソッドのセクションで説明します。メソッドが、サポートされない接続上で呼び出されると、 例外がスローされます。メソッド GetConnType を使用して接続タイプを判別してください。

CPI、LPI、および FontSize はプロパティー FaceName によって決定されます。そのため、FaceName を設定する前に CPI、LPI、および FontSize を設定し、 それらの値が FaceName プロパティーに対して有効ではない場合、異なる CPI、LPI、または FontSize 値が接続で再構成される場合があります。CPI、LPI、または FontSize を設定する前に FaceName 値を設定する必要があります。あるいは、FaceName を設定するたびに CPI、LPI、および FontSize を照会して、希望する値を使用していることを確認できます。

ECLPageSettings メソッド

以下のセクションで、ECLPageSettings クラスに有効なメソッドについて説明します。

ECLPageSettings(char Name) ECLPageSettings(long Handle) ~ECLPageSettings() void SetCPI(ULONG CPI=FONT_CPI) ULONG GetCPI() const BOOL IsFontCPI() void SetLPI(ULONG LPI=FONT_LPI) ULONG GetLPI() const BOOL IsFontLPI() void SetFontFaceName(const char *const FaceName) const char *GetFontFaceName() const void SetFontSize(ULONG FontSize) ULONG GetFontSize() void SetMaxLinesPerPage(ULONG MPL) ULONG GetMaxLinesPerPage() const void SetMaxCharsPerLine(ULONG MPP) ULONG GetMaxCharsPerLine() const void RestoreDefaults(ULONG Tabs=PAGE_TEXT) const

接続タイプ

ECLPageSettings メソッドに有効な接続タイプは、以下のとおりです。

接続タイプ	**ストリング値**
3270 表示装置	HOSTTYPE_3270DISPLAY
5250 ディスプレイ	HOSTTYPE_5250DISPLAY
3270 印刷装置	HOSTTYPE_3270PRINTER
VT (ASCII) エミュレーション	HOSTTYPE_VT

ECLPageSettings コンストラクター

このメソッドは、接続名またはハンドルを 使用して ECLPageSettings オブジェクトを作成します。

プロトタイプ

ECLPageSettings(char Name)

ECLPageSettings(long Handle)

パラメーター

char Name

1 文字の接続の短縮名。有効値は A から Z です。

long Handle

ECL 接続のハンドル。

戻り値

なし

例

以下の例は、接続名および接続ハンドルを使用して ECLPageSettings オブジェクトを作成する方法を示します。

```
void Sample108() {

   ECLPageSettings *PgSet1, *PgSet2; // Pointer to ECLPageSettings objects
   ECLConnList ConnList; // Connection list object

   try {
      // Create ECLPageSettings object for connection 'A'
      PgSet1 = new ECLPageSettings('A');
      // Create ECLPageSettings object for first connection in conn list
      ECLConnection *Connection = ConnList.GetFirstConnection();
      if (Connection != NULL) {
         PgSet2 = new ECLPageSettings(Connection->GetHandle());
         printf("PgSet#1 is for connection %c, PgSet #2 is for connection %c.\n",
                 PgSet1->GetName(), PgSet2->GetName());
         delete PgSet1;
         delete PgSet2;
      }
      else
         printf("No connections to create PageSettings object.\n");
   } catch (ECLErr Err) {
      printf("ECL Error: %s\n", Err.GetMsgText());
   }
} // end sample
```

SetCPI

このメソッドは、接続の CPI (字/インチ) 値を設定します。このメソッドを引数なしで呼び出すと、 接続には「フォント CPI」が設定されます。

プロトタイプ

void SetCPI(ULONG CPI=FONT_CPI);

パラメーター

ULONG CPI

1 インチ当たりの文字数。このパラメーターはオプションです。デフォルト値は FONT_CPI です。

戻り値

なし

例

```
void Sample109() {

   ECLPageSettings PgSet('A');

   PgSet.SetCPI(10);
   ULONG cpi = PgSet.GetCPI();
   printf("CPI = %ld\n", cpi);
   if (PgSet.IsFontCPI())
      printf("FontCPI\n");
   else
      printf("Not FontCPI\n");
} // end sample
```

GetCPI

このメソッドは、接続の CPI (字/インチ) 値を戻します。関連する接続で**「フォント CPI」**が選択されていても、このメソッドは関連する接続のフォントに選択されている CPI の値を戻します。

接続に**「フォント CPI」**が構成されている場合、このメソッドは定数 FONT_CPI を戻しません。IsFontCPI メソッドを使用して、接続に**「フォント CPI」**が設定されているかどうかを判別します。

プロトタイプ

ULONG GetCPI() const;

パラメーター

なし

戻り値

ULONG CPI

1 インチ当たりの文字数。

例

```
void Sample109() {

   ECLPageSettings PgSet('A');

   PgSet.SetCPI(10);
   ULONG cpi = PgSet.GetCPI();
   printf("CPI = %ld\n", cpi);
   if (PgSet.IsFontCPI())
      printf("FontCPI\n");
   else
      printf("Not FontCPI\n");
} // end sample
```

IsFontCPI

このメソッドは、接続に**「フォント CPI」**が設定されているかどうかの標識を戻します。

プロトタイプ

BOOL IsFontCPI();

パラメーター

なし

戻り値

BOOL

以下の値を指定できます。

- 接続に**「フォント CPI」**が設定されている場合、TRUE。
- 接続に**「フォント CPI」**が設定されていない場合、FALSE。

例

```
void Sample109() {

   ECLPageSettings PgSet('A');

   PgSet.SetCPI(10);
   ULONG cpi = PgSet.GetCPI();
   printf("CPI = %ld\n", cpi);
   if (PgSet.IsFontCPI())
      printf("FontCPI\n");
   else
      printf("Not FontCPI\n");
} // end sample
```

SetLPI

このメソッドは、接続の LPI (行/インチ) 値を設定します。このメソッドを引数なしで呼び出すと、 接続には「フォント LPI」が設定されます。

プロトタイプ

void SetLPI(ULONG LPI=FONT_LPI);

パラメーター

ULONG LPI

1 インチ当たりの行数。このパラメーターはオプションです。デフォルト値は FONT_LPI です。

戻り値

なし

例

```
void Sample110() {

   ECLPageSettings PgSet('A');

   PgSet.SetLPI(10);
   ULONG lpi = PgSet.GetLPI();
   printf("LPI = %ld\n", lpi);
   if (PgSet.IsFontLPI())
      printf("FontLPI\n");
   else
      printf("Not FontLPI\n");
} // end sample
```

GetLPI

このメソッドは、接続の LPI (行/インチ) 値を戻します。関連する接続で**「フォント LPI」**が選択されていても、このメソッドは関連する接続のフォントに選択されている LPI の値を戻します。

接続に**「フォント LPI」**が構成されている場合、このメソッドは定数 FONT_LPI を戻しません。IsFontLPI メソッドを使用して、接続に**「フォント LPI」**が設定されているかどうかを判別します。

プロトタイプ

ULONG GetLPI() const;

パラメーター

なし

戻り値

ULONG LPI

1 インチ当たりの行数。

例

```
void Sample110() {

   ECLPageSettings PgSet('A');

   PgSet.SetLPI(10);
   ULONG lpi = PgSet.GetLPI();
   printf("LPI = %ld\n", lpi);
   if (PgSet.IsFontLPI())
      printf("FontLPI\n");
   else
      printf("Not FontLPI\n");
} // end sample
```

IsFontLPI

このメソッドは、関連する接続に**「フォント LPI」**が設定されているかどうかの標識を戻します。

プロトタイプ

BOOL IsFontLPI();

パラメーター

なし

戻り値

BOOL

以下の値を指定できます。

- 接続に**「フォント LPI」**が設定されている場合、TRUE。
- 接続に**「フォント LPI」**が設定されていない場合、FALSE。

例

```
void Sample110() {

   ECLPageSettings PgSet('A');

   PgSet.SetLPI(10);
   ULONG lpi = PgSet.GetLPI();
   printf("LPI = %ld\n", lpi);
   if (PgSet.IsFontLPI())
      printf("FontLPI\n");
   else
      printf("Not FontLPI\n");
} // end sample
```

SetFontFaceName

このメソッドは、接続のフォント書体を設定します。

プロトタイプ

void SetFontFaceName(const char *const FaceName);

パラメーター

char *FaceName

フォント書体名を含むヌル終了ストリング。

戻り値

なし

例

```
void Sample111() {

   ECLPageSettings PgSet('A');
   const char *Face;

   PgSet.SetFontFaceName("Courier New");
   Face = PgSet.GetFontFaceName();
   printf("FaceName = %s\n", Face);
} // end sample
```

GetFontFaceName

このメソッドは、ヌル終了ストリングを指すポインターを戻します。ストリングには、ECLPageSettings オブジェクトに関連する接続のページ設定で現在選択されているフォントの書体名が含まれます。このメソッドは、毎回同じストリングを戻さない場合があります。

ストリングは、オブジェクトの存続期間内のみ有効です。そのストリングのコピーを作成するか、または必要があるたびに このメソッドを呼び出さなければなりません。

プロトタイプ

const char *GetFontFaceName() const;

パラメーター

なし

戻り値

char *

フォントの書体名を含むヌル終了ストリングを指すポインター。

例

```
void Sample111() {

   ECLPageSettings PgSet('A');
   const char *Face;

   PgSet.SetFontFaceName("Courier New");
   Face = PgSet.GetFontFaceName();
   printf("FaceName = %s\n", Face);
} // end sample
```

SetFontSize

このメソッドは、フォントのサイズを設定します。

プロトタイプ

void SetFontSize(ULONG FontSize);

パラメーター

ULONG FontSize

接続に設定するフォントのサイズ。

戻り値

なし

SetMaxLinesPerPage

このメソッドは、1 ページに印刷可能な最大行数を設定します。

プロトタイプ

void SetMaxLinesPerPage(ULONG MPL);

パラメーター

ULONG MPL

ページ当たり最大行数 (最大印刷行数)。有効な値は 1 から 255 の範囲内です。

戻り値

なし

例

```
void Sample113() {

   ECLPageSettings PgSet('A');

   PgSet.SetMaxLinesPerPage(40);
   ULONG MPL = PgSet.GetMaxLinesPerPage();
   printf("MaxLinesPerPage = %ld\n", MPL);
} // end sample
```

GetMaxLinesPerPage

このメソッドは、1 ページに印刷可能な最大行数を戻します。

プロトタイプ

ULONG GetMaxLinesPerPage() const;

パラメーター

なし

戻り値

ULONG

ページ当たり最大行数 (最大印刷行数)。

例

```
void Sample113() {

   ECLPageSettings PgSet('A');

   PgSet.SetMaxLinesPerPage(40);
   ULONG MPL = PgSet.GetMaxLinesPerPage();
   printf("MaxLinesPerPage = %ld\n", MPL);
} // end sample
```

SetMaxCharsPerLine

このメソッドは、1 行に印刷可能な最大文字数を設定します。

プロトタイプ

void SetMaxCharsPerLine(ULONG MPP);

パラメーター

ULONG MPP

1 行に印刷可能な最大文字数 (最大印刷位置)。有効な値は 1 から 255 の範囲内です。

戻り値

なし

例

```
void Sample114() {

   ECLPageSettings PgSet('A');

   PgSet.SetMaxCharsPerLine(50);
   ULONG MPP = PgSet.GetMaxCharsPerLine();
   printf("MaxCharsPerLine=%ld\n", MPP);
} // end sample
```

GetMaxCharsPerLine

このメソッドは、1 行に印刷可能な最大文字数を戻します。

プロトタイプ

ULONG GetMaxCharsPerLine() const;

パラメーター

なし

戻り値

ULONG

1 行に印刷可能な最大文字数 (最大印刷位置)。

例

```
void Sample114() {

   ECLPageSettings PgSet('A');

   PgSet.SetMaxCharsPerLine(50);
   ULONG MPP = PgSet.GetMaxCharsPerLine();
   printf("MaxCharsPerLine=%ld\n", MPP);
} // end sample
```

RestoreDefaults

このメソッドは、「ページ設定」パネルの「nFlags」フィールドに指定されたプロパティー・ページのシステム・デフォルト値を復元します。これは、接続の**「ページ設定」**ダイアログのプロパティー・ページの**「デフォルト」**ボタンをクリックすることと同じです。

プロトタイプ

void RestoreDefaults(ULONG Flags=PAGE_TEXT) const;

パラメーター

ULONG Flags

このパラメーターはオプションです。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。

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。これは、現在サポートされる唯一のプロパティー・ページです。

戻り値

なし

例

```
void Sample115() {

   ECLPageSettings PgSet('A');

   PgSet.RestoreDefaults(PAGE_TEXT);
} // end sample
```

ECLPrinterSettings クラス

ECLPrinterSettings クラスは、Z and I Emulator for Windows 接続のプリンター設定での操作を実行します。ここでは、プリンターおよび PDT モードなどの**「ファイル」→「プリンター設定」**ダイアログの設定を検索および構成できます。

派生

ECLBase > ECLConnection > ECLPrinterSettings

プロパティー

なし

制約事項

各メソッドに関連する接続は、メソッドを成功させるために 特定の状態になければなりません。制約事項を満たしていないと、 該当する例外が引き起こされます。

ECLPrinterSettings クラスのメソッドが呼び出されるときには、 以下の制約事項が適用されます。制約事項を満たしていないと、 例外がスローされます。

- 接続の**「ページ設定」**および**「プリンター設定」**ダイアログが使用中ではない。
- 接続が印刷中ではない。

特定のメソッドに対して追加の制約事項が適用される場合があります。

使用上の注意

ECLPrinterSettings は ECLConnection から派生するため、ECLConnection オブジェクトに含まれるすべての情報を取得できます。詳しくは、ECLConnection クラス (ページ 24)を参照してください。

ECLPrinterSettings オブジェクトは、作成時に識別された接続用に作成されます。ECLPrinterSettings オブジェクトを作成するには、接続 ID (**A** から **Z** の単一の英字) または接続ハンドル (通常は ECLConnection オブジェクトから取得) のいずれかを渡します。1 つの名前またはハンドルに対して、一度に 1 つの Z and I Emulator for Windows 接続しかオープンできません。

ECLSession クラスは、このオブジェクトのインスタンスを作成します。ECLSession が提供する他のサービスをアプリケーションで必要としない場合、 このオブジェクトを単独で作成しても構いません。それ以外の場合、ECLSession オブジェクトを作成することを検討し、ECLSession によって作成されたオブジェクトを使用してください。詳しくは、ECLSession クラス (ページ 183)を参照してください。

ECLPrinterSettings メソッド

以下のセクションで、ECLPrinterSettings クラスに有効なメソッドを説明します。

ECLPrinterSettings(char Name) ECLPrinterSettings(long Handle) ~ECLPrinterSettings() void SetPDTMode(BOOL PDTMode=TRUE, const char*const PDTFile = NULL) const char *GetPDTFile() const BOOL IsPDTMode() const ECLPrinterSettings::PrintMode GetPrintMode() const void SetPrtToDskAppend(const char *const FileName = NULL) const char *GetPrtToDskAppendFile() void SetPrtToDskSeparate(const char *const FileName = NULL) const char *GetPrtToDskSeparateFile() void SetSpecificPrinter(const char *const PrinterName) void SetWinDefaultPrinter() const char*GetPrinterName() void SetPromptDialog(BOOL Prompt=TRUE) BOOL IsPromptDialogEnabled()

ECLPrinterSettings コンストラクター

このメソッドは、接続名またはハンドル使用して ECLPrinterSettings オブジェクトを作成します。

プロトタイプ

ECLPrinterSettings(char Name)

ECLPrinterSettings(long Handle)

パラメーター

char Name

1 文字の接続の短縮名。有効値は A から Z です。

long Handle

ECL 接続のハンドル。

戻り値

なし

例

以下の例は、接続名および接続ハンドルを使用して ECLPrinterSettings オブジェクトを作成する方法を示します。

```
void Sample116() {
   ECLPrinterSettings *PrSet1, *PrSet2; // Pointer to ECLPrinterSettings objects
   ECLConnList ConnList; // Connection list object

   try {
      // Create ECLPrinterSettings object for connection 'A'
      PrSet1 = new ECLPrinterSettings('A');
      // Create ECLPrinterSettings object for first connection in conn list
      ECLConnection *Connection = ConnList.GetFirstConnection();
      if (Connection != NULL) {
         PrSet2 = new ECLPrinterSettings(Connection->GetHandle());
         printf("PrSet#1 is for connection %c, PrSet #2 is for connection %c.\n",
            PrSet1->GetName(), PrSet2->GetName());
      delete PrSet1;
      delete PrSet2;
   } else
      printf("No connections to create PageSettings object.\n");
   }
   catch (ECLErr Err) {
      printf("ECL Error: %s\n", Err.GetMsgText());
   }
} // end sample
```

SetPDTMode

このメソッドは、所定の PDT ファイルによって PDT モードに接続を設定するか、非 PDT モード (GDI モード) に接続を設定します。

注: このメソッドが PDTMode を False に設定して呼び出される場合には、 関連する接続の PrintMode が既に SpecificPrinter または WinDefaultPrinter である必要があります。

プロトタイプ

void SetPDTMode(BOOL PDTMode=TRUE, const char *const PDTFile = NULL);

パラメーター

BOOL PDTMode

このパラメーターはオプションです。以下の値を指定できます。

- PDT モードに接続を設定するには **TRUE**。これはデフォルト値です。
- 非 PDT モードに接続を設定するには **FALSE**。

char *PDTFile

PDT ファイルの名前を含むヌル終了ストリング。

このパラメーターはオプションです。これは、PDTMode が TRUE の場合にのみ使用されます。PDTMode が FALSE の場合、このパラメーターは無視されます。

以下の値を指定できます。

- NULL

  接続に構成されている PDT ファイルが使用されます。接続にまだ PDT ファイルが構成されていない場合、このメソッドは例外をスローして失敗します。これはデフォルト値です。
- パスなしのファイル名

  Z and I Emulator for Windows のインストール・パスの PDFPDT サブフォルダー内の PDTFile が使用されます。
- ファイルの完全修飾パス名

  PDTFile が存在しない場合、このメソッドは例外をスローして失敗します。

戻り値

なし

例

```
void Sample117() {

   ECLPrinterSettings PrSet('A');

   try {
      PrSet.SetPDTMode(TRUE, "epson.pdt");
      const char *PDTFile = PrSet.GetPDTFile();
      printf("PDT File = %s\n", PDTFile);
      if (PrSet.IsPDTMode())
         printf("PDTMode\n");
      else
         printf("Not PDTMode\n");
      PrSet.SetPDTMode(FALSE);
      PrSet.SetPDTMode(TRUE);
   }
   catch (ECLErr Err) {
      printf("ECL Error: %s\n", Err.GetMsgText());
   }
} // end sample
```

GetPDTFile

このメソッドは、接続に構成されている PDT ファイルを戻します。このメソッドは、毎回同じストリングを戻さない場合があります。

ストリングは、オブジェクトの存続期間内のみ有効です。そのストリングのコピーを作成するか、または必要があるたびに このメソッドを呼び出さなければなりません。

プロトタイプ

const char *GetPDTFile() const;

パラメーター

なし

戻り値

char *

以下の値を指定できます。

- 接続の PDT ファイルの完全修飾パス名を含むヌル終了ストリング。
- 接続に PDT ファイルが構成されていない場合は **NULL**。

例

```
void Sample117() {

   ECLPrinterSettings PrSet('A');

   try {
      PrSet.SetPDTMode(TRUE, "epson.pdt");
      const char *PDTFile = PrSet.GetPDTFile();
      printf("PDT File = %s\n", PDTFile);
      if (PrSet.IsPDTMode())
         printf("PDTMode\n");
      else
         printf("Not PDTMode\n");
      PrSet.SetPDTMode(FALSE);
      PrSet.SetPDTMode(TRUE);
   }
   catch (ECLErr Err) {
      printf("ECL Error: %s\n", Err.GetMsgText());
   }
} // end sample
```

IsPDTMode

このメソッドは、接続の PDT モードの状態を戻します。

プロトタイプ

BOOL IsPDTMode() const;

パラメーター

なし

戻り値

BOOL

以下の値を指定できます。

- 接続が PDT モードの場合、TRUE。
- 接続が PDT モードでない場合、FALSE。

例

```
void Sample117() {

   ECLPrinterSettings PrSet('A');

   try {
      PrSet.SetPDTMode(TRUE, "epson.pdt");
      const char *PDTFile = PrSet.GetPDTFile();
      printf("PDT File = %s\n", PDTFile);
      if (PrSet.IsPDTMode())
         printf("PDTMode\n");
      else
         printf("Not PDTMode\n");
      PrSet.SetPDTMode(FALSE);
      PrSet.SetPDTMode(TRUE);
   }
   catch (ECLErr Err) {
      printf("ECL Error: %s\n", Err.GetMsgText());
   }
} // end sample
```

GetPrintMode

このメソッドは、接続の PrintMode を示す列挙値を戻します。列挙データ型 ECLPrinterSettings::PrintMode は ECLPRSET.HPP に定義されます。

PrintMode は、以下のいずれかになります。

- **PrtToDskAppend (ディスクへの印刷 - 付加**モード)

  これは、ホスト・セッションの**「プリンター設定」→「プリンター」→「ディスクへの印刷」**ダイアログで**「付加」**オプションを選択することと同じです。

- **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」**→**「プリンター」**→**「ディスクへの印刷」**ダイアログで**「別個」**オプションを選択することと同じです。
- **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」**ダイアログで**「Windows デフォルト・プリンターを使用」**オプションを選択することと同じです。
- **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」**ダイアログで、**「Windows デフォルト・プリンターを使用」**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。

プロトタイプ

ECLPrinterSettings::PrintMode GetPrintMode() const;

パラメーター

なし

戻り値

ECLPrinterSettings::PrintMode

ECLPRSET.HPP に定義されている PrintMode 値の 1 つ。

例

```
void Sample118() {

   ECLPrinterSettings PrSet('A');

   ECLPrinterSettings::PrintMode PrtMode;
   PrtMode = PrSet.GetPrintMode();
   switch (PrtMode) {
   case ECLPrinterSettings::PrtToDskAppend:
      printf("PrtToDskAppend mode\n");
      break;
   case ECLPrinterSettings::PrtToDskSeparate:
      printf("PrtToDskSeparate mode\n");
      break;
   case ECLPrinterSettings::SpecificPrinter:
      printf("SpecificPrinter mode\n");
      break;
   case ECLPrinterSettings::WinDefaultPrinter:
      printf("WinDefaultPrinter mode\n");
      break;
   }
} // end sample
```

SetPrtToDskAppend

このメソッドは、PrintMode を**ディスクへの印刷 - コピー追加**モードに設定し、このモードに該当するファイルを設定します。

注:

1. 関連する接続は PDT モードにする必要があります。
2. このファイルを設定するフォルダーには書き込みアクセス権限が必要です。権限がない場合、 このメソッドは例外をスローして失敗します。
3. ファイルが存在する場合には、それが使用されます。それ以外の場合、 印刷の完了時に作成されます。

プロトタイプ

```
void SetPrtToDskAppend(const char *const FileName = NULL);
```

パラメーター

char *FileName

ディスクへの印刷 - コピー追加ファイルの名前を含むヌル終了ストリング。このパラメーターはオプションです。
以下の値を指定できます。

- NULL

  接続でこの PrintMode に現在構成されているファイルが使用されます。接続にまだファイルが構成されていないと、このメソッドは例外をスローして失敗します。これはデフォルト値です。
- 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を見つけます。
- ファイルの完全修飾パス名

  パス内にディレクトリーが存在している必要があります。 ない場合、メソッドは例外をスローして失敗します。ファイルがパス内に存在する必要はありません。

戻り値

なし

例

```
void Sample119() {

   ECLPrinterSettings PrSet('A');

   try {
      PrSet.SetPrtToDskAppend("dskapp.txt");
      const char *DskAppFile = PrSet.GetPrtToDskAppendFile();
      printf("Print to Disk-Append File = %s\n", DskAppFile);
   }
   catch (ECLErr Err) {
      printf("ECL Error: %s\n", Err.GetMsgText());
   }
} // end sample
```

GetPrtToDskAppendFile

このメソッドは、**ディスクへの印刷 - コピー追加**モードに構成されたファイルを戻します。このファイルは、**ディスクへの印刷 - コピー追加**ファイルと呼ばれます。このメソッドは、毎回同じストリングを戻さない場合があります。

ストリングは、オブジェクトの存続期間内のみ有効です。そのストリングのコピーを作成するか、または必要があるたびに このメソッドを呼び出さなければなりません。

プロトタイプ

const char *GetPrtToDskAppendFile();

パラメーター

なし

戻り値

char *

以下の値を指定できます。

- 接続の**ディスクへの印刷 - コピー追加**ファイルの完全修飾パス名を含むヌル終了ストリング。
- 接続に**ディスクへの印刷 - コピー追加**ファイルが構成されていない場合、NULL。

例

```
void Sample119() {

   ECLPrinterSettings PrSet('A');

   try {
      PrSet.SetPrtToDskAppend("dskapp.txt");
      const char *DskAppFile = PrSet.GetPrtToDskAppendFile();
```

```
      printf("Print to Disk-Append File = %s\n", DskAppFile);
   }
   catch (ECLErr Err) {
      printf("ECL Error: %s\n", Err.GetMsgText());
   }
} // end sample
```

SetPrtToDskSeparate

このメソッドは、接続を**ディスクへの印刷 - 別個**モードに設定し、このモードに該当するファイルを設定します。

注:

1. 関連する接続は PDT モードにする必要があります。
2. このファイルを設定するフォルダーには書き込みアクセス権限が必要です。権限がない場合、 このメソッドは例外をスローして失敗します。
3. ファイル名には拡張子を入れないでください。拡張子が入っていると、このメソッドは例外をスローして失敗します。

プロトタイプ

void SetPrtToDskSeparate(const char *const FileName = NULL);

パラメーター

char *FileName

ディスクへの印刷 - 別個ファイルの名前を含むヌル終了ストリング。このパラメーターはオプションです。
以下の値を指定できます。

- NULL

  接続でこの PrintMode に現在構成されているファイルが使用されます。接続にまだファイルが構成されていないと、このメソッドは例外をスローして失敗します。これはデフォルト値です。
- 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を見つけます。
- ファイルの完全修飾パス名

  パス内にディレクトリーが存在している必要があります。 ない場合、メソッドは例外をスローして失敗します。ファイルがパス内に存在する必要はありません。

戻り値

なし

例

```
void Sample120() {

   ECLPrinterSettings PrSet('A');

   try {
      PrSet.SetPrtToDskSeparate("dsksep");
      const char *DskSepFile = PrSet.GetPrtToDskSeparateFile();
      printf("Print to Disk-Separate File = %s\n", DskSepFile);
   }
   catch (ECLErr Err) {
      printf("ECL Error: %s\n", Err.GetMsgText());
   }
} // end sample
```

GetPrtToDskSeparateFile

このメソッドは、**ディスクへの印刷 - 別個**モードに構成されたファイルを戻します。このファイルは、**ディスクへの印刷 - 別個**ファイルと呼ばれます。このメソッドは、毎回同じストリングを戻さない場合があります。

ストリングは、オブジェクトの存続期間内のみ有効です。そのストリングのコピーを作成するか、または必要があるたびに このメソッドを呼び出さなければなりません。

プロトタイプ

const char *GetPrtToDskSeparateFile();

パラメーター

なし

戻り値

char *

以下の値を指定できます。

- **ディスクへの印刷 - 別個**ファイルの完全修飾パス名を含むヌル終了ストリング。
- 接続に**ディスクへの印刷 - 別個**ファイルが構成されていない場合、NULL。

例

```
void Sample120() {
```

```
   ECLPrinterSettings PrSet('A');

   try {
      PrSet.SetPrtToDskSeparate("dsksep");
      const char *DskSepFile = PrSet.GetPrtToDskSeparateFile();
      printf("Print to Disk-Separate File = %s\n", DskSepFile);
   }
   catch (ECLErr Err) {
      printf("ECL Error: %s\n", Err.GetMsgText());
   }
} // end sample
```

SetSpecificPrinter

このメソッドは、Printer パラメーターに指定されたプリンターによって 接続を SpecificPrinter モードに設定します。

プロトタイプ

void SetSpecificPrinter(const char *const Printer);

パラメーター

char *Printer

プリンター名およびポート名を含むヌル終了ストリング。プリンターが存在しない場合、このメソッドは例外をスローして失敗します。
値の形式は次のようにする必要があります。

```
<Printer name> on <Port Name>
```

例:

- `HP LaserJet 4050 Series PCL 6 on LPT1`

戻り値

なし

例

```
void Sample121() {

   ECLPrinterSettings PrSet('A');

   try {
      PrSet.SetSpecificPrinter("HCL InfoPrint 40 PS on Network Port");
      const char *Printer = PrSet.GetPrinterName();
      printf("Printer = %s\n", Printer);
   }
   catch (ECLErr Err) {
```

```
      printf("ECL Error: %s\n", Err.GetMsgText());
   }
} // end sample
```

SetWinDefaultPrinter

このメソッドは、接続を WinDefaultPrinter モードに設定します。これにより、接続は Windows® のデフォルト・プリンターを使用するようになります。マシンに Windows のデフォルト・プリンターが構成されていない場合、このメソッドは例外をスローして失敗します。

プロトタイプ

void SetWinDefaultPrinter();

パラメーター

なし

戻り値

なし

例

```
void Sample122() {

   ECLPrinterSettings PrSet('A');

   try {
      PrSet.SetWinDefaultPrinter();
      const char *Printer = PrSet.GetPrinterName();
   printf("Windows Default Printer = %s\n", Printer);
   }
   catch (ECLErr Err) {
      printf("ECL Error: %s\n", Err.GetMsgText());
   }
} // end sample
```

GetPrinterName

このメソッドは、NULL または接続に構成されているプリンターの名前を戻します。このメソッドは、毎回同じストリングを戻さない場合があります。

ストリングは、オブジェクトの存続期間内のみ有効です。そのストリングのコピーを作成するか、または必要があるたびに このメソッドを呼び出さなければなりません。

プリンター名の形式は次のとおりです。

```
<Printer name> on <Port Name>
```

例:

- `HP LaserJet 4050 Series PCL 6 on LPT1`

プロトタイプ

const char *GetPrinterName();

パラメーター

なし

戻り値

char *

以下の値を指定できます。

- 接続の PrintMode が SpecificPrinter の場合、特定のプリンターの名前を含むヌル終了ストリング。
- 接続の PrintMode が WinDefaultPrinter の場合、Windows のデフォルト・プリンターの名前を含むヌル終了ストリング。
- 接続にプリンターが構成されていない場合、あるいは接続の PrintMode が PrtToDskAppend または PrtToDskSeparate の場合、NULL。

例

```
void Sample122() {

   ECLPrinterSettings PrSet('A');

   try {
      PrSet.SetWinDefaultPrinter();
      const char *Printer = PrSet.GetPrinterName();
   printf("Windows Default Printer = %s\n", Printer);
   }
   catch (ECLErr Err) {
      printf("ECL Error: %s\n", Err.GetMsgText());
   }
} // end sample
```

SetPromptDialog

このメソッドは、印刷前に「プリンター設定」ダイアログを表示するオプションを設定またはリセットします。

プロトタイプ

void SetPromptDialog(BOOL bPrompt=TRUE);

パラメーター

BOOL bPrompt

このパラメーターはオプションです。以下の値を指定できます。

- 印刷前に「プリンター設定」ダイアログを表示するには TRUE。これはデフォルト値です。
- 印刷前に「プリンター設定」ダイアログを表示しないようにするには FALSE。

戻り値

なし

例

```
void Sample123() {

   ECLPrinterSettings PrSet('A');

   try {
      PrSet.SetPromptDialog();
      if (PrSet.IsPromptDialogEnabled())
         printf("Prompt Dialog before Printing - Enabled\n");
      else
         printf("Prompt Dialog before Printing - Disabled\n");
   }
   catch (ECLErr Err) {
      printf("ECL Error: %s\n", Err.GetMsgText());
   }
} // end sample
```

IsPromptDialogEnabled

このメソッドは、印刷前に「プリンター設定」ダイアログを表示するかどうかを検査します。

プロトタイプ

BOOL IsPromptDialogEnabled();

パラメーター

なし

戻り値

BOOL

以下の値を指定できます。

- 印刷前に「プリンター設定」ダイアログを表示する場合、TRUE。
- 印刷前に「プリンター設定」ダイアログを表示しない場合、FALSE。

例

```
void Sample123() {

   ECLPrinterSettings PrSet('A');

   try {
      PrSet.SetPromptDialog();
      if (PrSet.IsPromptDialogEnabled())
         printf("Prompt Dialog before Printing - Enabled\n");
      else
         printf("Prompt Dialog before Printing - Disabled\n");
   }
   catch (ECLErr Err) {
      printf("ECL Error: %s\n", Err.GetMsgText());
   }
} // end sample
```

第 3 章. ホスト・アクセス・クラス・ライブラリーの自動化オブジェクト

ホスト・アクセス・クラス・ライブラリーの自動化オブジェクトを使用すると、Z and I Emulator for Windows 製品が Microsoft® の COM ベースのオートメーション技術 (以前は OLE オートメーションと呼ばれていた) をサポートできるようになります。ECL 自動化オブジェクトは、一連の自動化サーバーであり、これを使用することによって、自動化コントローラー (Microsoft® Visual Basic® など) が、プログラマチックに Z and I Emulator for Windows のデータおよび機能にアクセスすることができます。

この例として、キーを Z and I Emulator for Windows の表示スペースに送信することを考慮します。これは、「Z and I Emulator for Windows」ウィンドウでキーを入力することにより実行できますが、適切な Z and I Emulator for Windows の自動化サーバー (この場合、autECLPS) を使用して自動処理することもできます。Visual Basic® を使用して、autECLPS オブジェクトを作成し、表示スペースに配置されるストリングを使用して、そのオブジェクトの SendKeys メソッドを呼び出します。

言い替えると、自動化プロトコルを制御できるアプリケーション (自動化コントローラー) は、Z and I Emulator for Windows の操作の一部 (自動化サーバー) を制御することができます。Z and I Emulator for Windows は、ECL 自動化オブジェクトを使用する Visual Basic® スクリプトをサポートします。詳細については、Z and I Emulator for Windows のマクロやスクリプトのサポートについて参照してください。

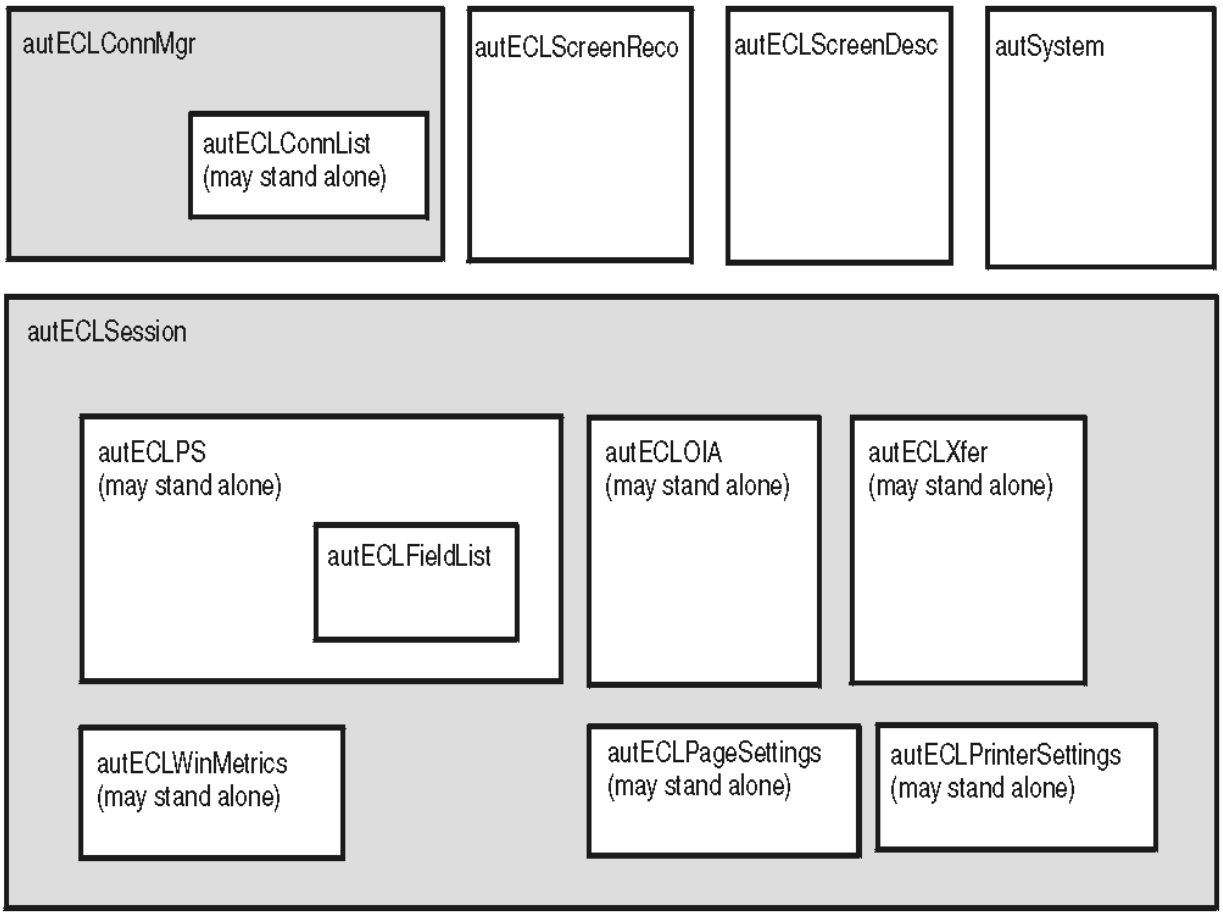
Z and I Emulator for Windows は、これを実行するためにいくつかの自動化サーバーを提供しています。これらのサーバーは、直感的に理解できる身近なオブジェクトとして、そして Z and I Emulator for Windows の操作容易性を制御するメソッドとプロパティーを持つものとして実装されています。各オブジェクトは、自動化ホスト・アクセス・クラス・ライブラリーで あることを表す autECL で始まります。オブジェクトは、以下のとおりです。

- autECLConnList クラス (ページ 256) ページの autECLConnList (接続リスト) は、 所定のシステムの Z and I Emulator for Windows 接続を列挙します。これは、autECLConnMgr に含まれていますが、autECLConnMgr から独立して 作成することもできます。
- autECLConnMgr クラス (ページ 264) ページの autECLConnMgr (接続マネージャー) は、所定のシステムで Z and I Emulator for Windows 接続を管理するメソッドおよびプロパティーを提供します。この場合の接続とは、Z and I Emulator for Windows のウィンドウのことです。
- autECLFieldList クラス (ページ 270) ページの autECLFieldList (フィールド・リスト) は、 エミュレーター表示スペースでのフィールドの操作を実行します。
- autECLOIA クラス (ページ 281) ページの autECLOIA (オペレーター情報域) は、 オペレーター情報域を照会および操作するメソッドおよびプロパティーを提供します。これは、autECLSession に含まれていますが、autECLSession から独立して 作成することもできます。
- autECLPS クラス (ページ 299) ページの autECLPS (表示スペース) は、 関連する Z and I Emulator for Windows 接続について表示スペースを照会および操作するメソッドとプロパティーを提供します。ここには、表示スペース内のすべてのフィールドのリストが含まれています。これは、autECLSession に含まれていますが、autECLSession から独立して 作成することもできます。
- autECLScreenDesc クラス (ページ 339) ページの autECLScreenDesc (画面記述) は、画面を記述するためのメソッドおよびプロパティーを提供します。これは、autECLPS オブジェクトまたは autECLScreenReco オブジェクトで画面を待機する ために使用することができます。

- autECLScreenReco クラス (ページ 347) ページの autECLScreenReco (画面認識) は、HACL 画面認識システムのエンジンを提供します。
- autECLSession クラス (ページ 353) ページの autECLSession (セッション) は、一般セッション関連の機能および情報を提供します。便利なように、autECLPS、autECLOIA、autECLXfer、autECLWinMetrics、autECLPageSettings、 および autECLPrinterSettings オブジェクトも含まれています。
- autECLWinMetrics クラス (ページ 366) ページの autECLWinMetrics (ウィンドウ・メトリック) は、このオブジェクトに関連する Z and I Emulator for Windows セッションのウィンドウ・メトリックを照会するメソッドを提供します。例えば、このオブジェクトを使用すると、Z and I Emulator for Windowsのウィンドウを最小化したり最大化したりすることができます。これは、autECLSession に含まれていますが、autECLSession から独立して 作成することもできます。
- autECLXfer クラス (ページ 383) ページの autECLXfer (ファイル転送) は、このファイル転送オブジェクトに関連するZ and I Emulator for Windows接続でのホストおよびワークステーション間のファイルを転送するメソッドおよびプロパティーを提供します。これは、autECLSession に含まれていますが、autECLsessionから独立して 作成することもできます。
- autECLPageSettings クラス (ページ 397) ページの autECLPageSettings (ページ設定) は、セッションの「ページ設定」ダイアログの「CPI」、「LPI」、 および「書体名」などの一般に使用される設定を照会および操作するメソッドおよびプロパティーを提供します。これは、autECLSession に含まれていますが、autECLSession から独立して 作成することもできます。
- autECLPrinterSettings クラス (ページ 408) ページの autECLPrinterSettings (プリンター設定) は、 セッションの「プリンター設定」ダイアログの「プリンター」や「PDT モード」などの設定を照会および操作するメソッドおよびプロパティーを提供します。これは、autECLSession に含まれていますが、autECLSession から独立して 作成することもできます。

図 3 : ホスト・アクセス・クラス・ライブラリーの自動化オブジェクト (ページ 256) に、autECL オブジェクトを図形的に表現します。

図 3. ホスト・アクセス・クラス・ライブラリーの自動化オブジェクト



この章は、それぞれのオブジェクトのメソッドおよびプロパティーを詳細に説明しており、 自動化オブジェクトを使用する潜在的可能性のあるすべてのユーザーを対象にしています。オブジェクトは、Visual Basic® などのスクリプト・アプリケーションを介して使用する場合が一般的であるため、例ではすべて Visual Basic® 形式を使用しています。

autSystem クラス

autSystem クラスは、一部のプログラム言語で使用する場合に役立つ 2 つのユーティリティー関数を提供します。詳しくは、autSystem クラス (ページ 395)を参照してください。

autECLConnList クラス

autECLConnList には、 すべての開始された接続についての情報が含まれています。レジストリーでのその名前は ZIEWin.autECLConnList です。

autECLConnList オブジェクトには、ホストへの接続についての情報の集合が 含まれています。集合の各要素は、単一の接続 (エミュレーター・ウィンドウ) を 示しています。このリスト内の接続は、どのような状態 (例えば、停止または切断) にあるものでも構いません。すべての開始済みの接続が、このリストに示されます。リスト要素には、接続の状態が入っています。

autECLConnList オブジェクトは、現在の接続の 静的スナップショットを提供します。このリストは、接続の開始および停止に連動して更新されるわけではありません。Refresh メソッドは、autECLConnList オブジェクトの作成時に自動的に呼び出されます。autECLConnList オブジェクトをその作成後すぐ後に使用すると、 接続のリストは最新になります。しかし、作成後しばらくたった場合は、確実に現在のデータを取得するためには、 他のメソッドにアクセスする前に autECLConnList オブジェクトで Refresh メソッドを 呼び出さなければなりません。一度 Refresh を呼び出せば、集合の内容をすべて調べることができます。

プロパティー

このセクションでは、autECLConnList オブジェクトの プロパティーについて説明します。

タイプ	名前	属性
Long (長形式)	カウント	読み取り専用

以下の表は、集合要素プロパティーを示しています。 これは、リスト内の各項目に有効です。

タイプ	名前	属性
ストリング	名前	読み取り専用
長形式	ハンドル	読み取り専用
ストリング	ConnType	読み取り専用
長形式	CodePage	読み取り専用
ブール値	開始済み	読み取り専用
ブール値	CommStarted	読み取り専用
ブール値	APIEnabled	読み取り専用
ブール値	作動可能	読み取り専用

カウント

これは、Refresh メソッドへの最後の呼び出しに 関する autECLConnList 集合にある接続の数です。Count プロパティーは、Long データ型で読み取り専用 です。以下の例では、Count プロパティーを使用しています。

```
Dim autECLConnList as Object
Dim Num as Long

Set autECLConnList = CreateObject("ZIEWin.autECLConnList")

autECLConnList.Refresh
Num = autECLConnList.Count
```

名前

この集合要素プロパティーは、接続の接続名ストリングです。Z and I Emulator for Windows は、短い文字 ID (A から Z、または a から z) のみをストリングで戻します。Z and I Emulator for Windows 接続でオープンできるのは、1 つの名前につき 1 つしかありません。例えば、一度に 1 つの接続 A のみをオープンできます。Name は、String データ型で読み取り専用です。以下の例では、Name 集合要素プロパティーを使用しています。

```
Dim  Str as String
Dim autECLConnList as Object
Dim Num as Long

Set autECLConnList = CreateObject("ZIEWin.autECLConnList")

autECLConnList.Refresh
Str = autECLConnList(1).Name
```

ハンドル

この集合要素プロパティーは、接続のハンドルです。特定の 1 つのハンドルに対して 1 つしか、Z and I Emulator for Windows 接続をオープンできません。Handle は、Long データ型で読み取り専用です。以下の例では、Handle 集合要素プロパティーを使用しています。

```
Dim autECLConnList as Object
Dim Hand as Long

Set autECLConnList = CreateObject("ZIEWin.autECLConnList")

autECLConnList.Refresh
Hand = autECLConnList(1).Handle
```

ConnType

この集合要素プロパティーは、接続タイプです。このタイプは、時間の経過とともに変更する場合が あります。ConnType は、String データ型で読み取り専用です。以下の例で、ConnType プロパティーを示します。

```
Dim  Type as String
Dim autECLConnList as Object

Set autECLConnList = CreateObject("ZIEWin.autECLConnList")

autECLConnList.Refresh
Type = autECLConnList(1).ConnType
```

ConnType プロパティーの接続タイプは、以下のとおりです。

戻されるストリング	意味
DISP3270	3270 表示装置
DISP5250	5250 ディスプレイ
PRNT3270	3270 印刷装置
PRNT5250	5250 プリンター

ASCII	VT エミュレーション

CodePage

この集合要素プロパティーは、接続のコード・ページです。このコード・ページは、時間の経過とともに変更される場合があります。CodePage は、Long データ型で読み取り専用です。以下の例で、CodePage プロパティーを示します。

```
Dim  CodePage as Long
Dim autECLConnList as Object

Set autECLConnList = CreateObject("ZIEWin.autECLConnList")

autECLConnList.Refresh
CodePage = autECLConnList(1).CodePage
```

開始済み

この集合要素プロパティーで、 エミュレーター・ウィンドウが開始されたかどうかを示します。ウィンドウがオープンしている場合、値は True です。 その他の場合は False です。Started は、Boolean データ型で読み取り専用です。以下の例で、Started プロパティーを示します。

```
Dim autECLConnList as Object

Set autECLConnList = CreateObject("ZIEWin.autECLConnList")
autECLConnList.Refresh

' This code segment checks to see if is started.
' The results are sent to a text box called Result.
If Not autECLConnList(1).Started Then
  Result.Text = "No"
Else
  Result.Text = "Yes"
End If
```

CommStarted

この集合要素プロパティーは、ホストへの接続の状況を示しています。ホストが接続されている場合、値は True です。 その他の場合は False です。CommStarted は、Boolean データ型で読み取り専用です。以下の例で、CommStarted プロパティーを示します。

```
Dim autECLConnList as Object

Set autECLConnList = CreateObject("ZIEWin.autECLConnList")
autECLConnList.Refresh

' This code segment checks to see if communications are connected
' The results are sent to a text box called CommConn.
If Not autECLConnList(1).CommStarted Then
    CommConn.Text = "No"
```

```
Else
    CommConn.Text = "Yes"
End If
```

APIEnabled

この集合プロパティーで、エミュレーターが API 使用可能か どうかを示します。API 設定 (Z and I Emulator for Windows のウィンドウでは、**「ファイル」->「API の設定」**を選択) の状態に応じて、接続が可能な場合と、接続できない場合があります。エミュレーターが使用可能の場合には、True です。その他の場合には、False です。APIEnabled は、Boolean データ型で読み取り専用です。以下の例で、APIEnabled プロパティーを示します。

```
Dim autECLConnList as Object

Set autECLConnList = CreateObject("ZIEWin.autECLConnList")
autECLConnList.Refresh

' This code segment checks to see if API is enabled.
' The results are sent to a text box called Result.
If Not autECLConnList(1).APIEnabled Then
  Result.Text = "No"
Else
  Result.Text = "Yes"
End If
```

作動可能

この集合要素プロパティーで、エミュレーター・ウィンドウが 開始されていて、API 使用可能で、そして接続されているかどうかを示します。このプロパティーは、3 つのすべてのプロパティーを確認します。エミュレーターが準備できている場合には、値は True です。 その他の場合には、False です。Ready は、Boolean データ型で読み取り専用です。以下の例で、Ready プロパティーを示します。

```
Dim autECLConnList as Object

Set autECLConnList = CreateObject("ZIEWin.autECLConnList")
autECLConnList.Refresh

' This code segment checks to see if X is ready.
' The results are sent to a text box called Result.
If Not autECLConnList(1).Ready Then
  Result.Text = "No"
Else
  Result.Text = "Yes"
End If
```

autECLConnList メソッド

以下のセクションで、autECLConnList オブジェクトに 有効なメソッドを説明します。

void Refresh() Object FindConnectionByHandle(Long Hand) Object FindConnectionByName(String Name)

集合要素メソッド

以下の集合要素メソッドは、リスト内の各項目に有効です。

void StartCommunication() void StopCommunication()

最新表示

Refresh メソッドは、 開始されたすべての接続のスナップショットを取得します。

注: 確実に現在のデータを取得するには、autECLConnList 集合に アクセスする前にこのメソッドを呼び出さなければなりません。

プロトタイプ

void Refresh()

パラメーター

なし

戻り値

なし

例

以下の例は、開始されたすべての接続のスナップショットを取得するため に Refresh メソッドを使用する方法を示しています。

```
Dim autECLPSObj as Object
Dim autECLConnList as Object

Set autECLPSObj = CreateObject("ZIEWin.autECLPS")
Set autECLConnList = CreateObject("ZIEWin.autECLConnList")

' Initialize the connection
autECLConnList.Refresh
autECLPSObj.SetConnectionByHandle(autECLConnList(1).Handle)
```

FindConnectionByHandle

このメソッドは、**Hand** パラメーターで渡されるハンドルの autECLConnList オブジェクトでの要素を見つけます。このメソッドは通常、与えられた接続がシステムで活動状態であるか を確認するために使用されます。

プロトタイプ

Object FindConnectionByHandle(Long Hand)

パラメーター

Long Hand

リストで検索するハンドル。

戻り値

オブジェクト

集合要素ディスパッチ・オブジェクト。

例

以下の例は、接続ハンドルにより要素を見つける方法を示しています。

```
Dim Hand as Long
Dim autECLConnList as Object
Dim ConnObj as Object

Set autECLConnList = CreateObject("ZIEWin.autECLConnList")

' Initialize the collection
autECLConnList.Refresh
' Assume Hand obtained earlier
Set ConnObj = autECLConnList.FindConnectionByHandle(Hand)
Hand = ConnObj.Handle
```

FindConnectionByName

このメソッドは、**Name** パラメーターで渡される名前の autECLConnList オブジェクトでの要素を見つけます。このメソッドは通常、与えられた接続がシステムで活動状態であるか を確認するために使用されます。

プロトタイプ

Object FindConnectionByName(String Name)

パラメーター

String 型、Name 型

リストで検索する名前。

戻り値

オブジェクト

集合要素ディスパッチ・オブジェクト。

例

以下の例は、接続名により autECLConnList オブジェクト内の要素を 見つける方法を示しています。

```
Dim Hand as Long
Dim autECLConnList as Object
Dim ConnObj as Object

Set autECLConnList = CreateObject("ZIEWin.autECLConnList")

' Initialize the collection
autECLConnList.Refresh
' Assume Hand obtained earlier
Set ConnObj = autECLConnList.FindConnectionByName("A")
Hand = ConnObj.Handle
```

StartCommunication

StartCommunication 集合要素メソッドは、ホスト・データ・ストリームに ZIEWin エミュレーターを接続します。これは、ZIEWin エミュレーター**「通信」**メニューを表示して**「接続」**を選択した場合と同じ結果になります。

プロトタイプ

void StartCommunication()

パラメーター

なし

戻り値

なし

例

以下の例は、ZIEWin エミュレーター・セッションをホストへ接続する方法を示しています。

```
Dim autECLConnList as Object

Set autECLConnList = CreateObject("ZIEWin.autECLConnList")

'Start the first session
autECLConnList.Refresh
autECLConnList(1).StartCommunication()
```

StopCommunication

StopCommunication 集合要素メソッドは、ホスト・データ・ストリームから ZIEWin エミュレーターを切断します。これは、ZIEWin エミュレーター**「通信」**メニューを表示して**「切断」**を選択した場合と同じ結果になります。

プロトタイプ

void StopCommunication()

パラメーター

なし

戻り値

なし

例

以下の例は、ホストから ZIEWin エミュレーター・セッションを切断する方法を示したものです。

```
Dim autECLConnList as Object

Set autECLConnList = CreateObject("ZIEWin.autECLConnList")

'Start the first session
autECLConnList.Refresh
autECLConnList(1).StartCommunication()
'
'Interact programmatically with host
'
autECLConnList.Refresh
'Stop the first session
autECLConnList(1).StartCommunication()
```

autECLConnMgr クラス

autECLConnMgr は、マシン上のすべての Z and I Emulator for Windows 接続を管理します。これには、接続の開始および停止など、接続の管理に関連するメソッドが含まれています。また、autECLConnList オブジェクトを作成して、システムで認知されている すべての接続のリストを列挙します (autECLConnList クラス (ページ 256)を参照)。レジストリーでのその名前は ZIEWin.autECLConnMgr です。

プロパティー

このセクションでは、autECLConnMgr オブジェクトの プロパティーについて説明します。

タイプ	名前	属性
autECLConnList オブジェクト	autECLConnList	読み取り専用

autECLConnList

autECLConnMgr オブジェクトには、autECLConnList オブジェクトが 含まれています。このメソッドおよびプロパティーの詳細については、『autECLConnList クラス (ページ 256)』を参照してください。プロパティーには、autECLConnList の値があります。 これは、autECLConnList ディスパッチ・オブジェクトです。以下の例は、このプロパティーを示しています。

```
Dim Mgr as Object
Dim Num as Long

Set Mgr = CreateObject("ZIEWin.autECLConnMgr ")

Mgr.autECLConnList.Refresh
Num = Mgr.autECLConnList.Count
```

autECLConnMgr メソッド

以下のセクションで、autECLConnMgr オブジェクトに有効な メソッドを説明します。

void RegisterStartEvent() void UnregisterStartEvent() void StartConnection(String ConfigParms) void StopConnection(Variant Connection, [optional] String StopParms)

RegisterStartEvent

このメソッドは、セッションで開始イベントの通知を受け取るため の autECLConnMgr オブジェクトを登録します。

プロトタイプ

void RegisterStartEvent()

パラメーター

なし

戻り値

なし

例

例については、イベント処理の例 (ページ 270)を参照してください。

UnregisterStartEvent

開始イベントの処理を終了します。

プロトタイプ

void UnregisterStartEvent()

パラメーター

なし

戻り値

なし

例

例については、イベント処理の例 (ページ 270)を参照してください。

StartConnection

このメンバー関数は、新しい Z and I Emulator for Windows のエミュレーター・ウィンドウを開始します。ConfigParms ストリングには、使用上の注意 (ページ 266)で 説明されているとおりの接続構成情報が入っています。

プロトタイプ

void StartConnection(String ConfigParms)

パラメーター

String ConfigParms

構成ストリング。

戻り値

なし

使用上の注意

構成ストリングは、インストール・システムによって異なります。異なるインストール・システム上の autECL オブジェクトには、 構成ストリングに異なる形式または情報が必要とされる場合があります。新しいエミュレーターはこの呼び出しの応答として開始されますが、 ホストに接続される場合もあれば、接続されない場合もあります。

Z and I Emulator for Windows の場合、構成ストリングの形式は次のとおりです。

```
PROFILE=['']<filename>['] [CONNNAME=<c>] [WINSTATE=<MAX|MIN|RESTORE|HIDE>]
```

オプションのパラメーターは、大括弧 [] で囲みます。パラメーターは、少なくとも 1 つのブランクで区切ります。パラメーターは、大文字、小文字、または混合のいずれでも指定可能で、 順序も任意です。各パラメーターの意味は、次のとおりです。

- PROFILE=<filename>: 構成情報が含まれている Z and I Emulator for Windows ワークステーション・プロファイル (.WS ファイル) の名前を指定します。このパラメーターは、オプションではありません。 プロファイル名を入力しなければなりません。ファイル名にブランクが含まれる場合、名前を単一引用符で囲まなければなりません。<ファイル名> の値は、拡張子のないプロファイル名、.WS 拡張子の付いたプロファイル名、または完全修飾プロファイル名パスのどれでも構いません。
- CONNNAME=<c> は、新しい接続の簡易 ID を指定します。この値は、単一の英字 (A から Z、または a から z) でなければなりません。この値を指定しないと、使用可能な次の接続 ID が自動的に割り当てられます。
- WINSTATE=<MAX|MIN|RESTORE|HIDE> は、エミュレーター・ウィンドウの初期状態を指定します。このパラメーターを指定しない場合のデフォルト値は、RESTORE です。

例

以下の例は、新しい Z and I Emulator for Windows のエミュレーター・ウィンドウを開始する方法を示します。

```
Dim Mgr as Object
Dim Obj as Object
Dim Hand as Long

Set Mgr = CreateObject("ZIEWin.autECLConnMgr ")
Mgr.StartConnection("profile=coax connname=e")
```

StopConnection

StopConnection メソッドは、 接続ハンドルにより識別されるエミュレーター・ウィンドウを停止 (終了) します。StopParms ストリングの内容の詳細については、使用上の注意 (ページ 268)を 参照してください。

プロトタイプ

void StopConnection(Variant Connection, [optional] String StopParms)

パラメーター

Variant Connection

接続名またはハンドル。このバリアントに対して有効な型は short、long、BSTR、short by reference、 long by reference、および BSTR by reference です。

String StopParms

停止パラメーター・ストリング。ストリングの形式については、使用上の注意を参照。このパラメーターはオプションです。

戻り値

なし

使用上の注意

停止パラメーター・ストリングは、インストール・システムに応じて異なります。異なるインストール・システム上の autECL オブジェクトには、異なる形式または内容の パラメーター・ストリングが必要とされる場合があります。Z and I Emulator for Windows の場合、ストリングの形式は次のとおりです。

```
[SAVEPROFILE=<YES|NO|DEFAULT>]
```

オプションのパラメーターは、大括弧 [] で囲みます。パラメーターは、少なくとも 1 つのブランクで区切ります。パラメーターは、大文字、小文字、または混合のいずれでも指定可能で、 順序も任意です。各パラメーターの意味は、次のとおりです。

- SAVEPROFILE=<YES|NO|DEFAULT> は、現在の構成のワークステーション・プロファイル (.WS ファイル) への保管を制御します。これによって、これまでに加えた構成変更によってプロファイルを更新することができます。NO を指定した場合、接続が停止されるときプロファイルは 更新されません。YES を指定した場合、接続が停止されるとき現行構成 (変更されている ことがある) を使用してプロファイルは更新されます。DEFAULT を指定した場合には、更新オプションは**「ファイル」->「終了時に変更を保管」**エミュレーター・メニュー・オプションによって制御されます。このパラメーターを指定しない場合には、DEFAULT が使用されます。

例

以下の例は、接続ハンドルにより識別されるエミュレーター・ウィンドウを 停止する方法を示します。

```
Dim Mgr as Object
Dim Hand as Long

Set Mgr = CreateObject("ZIEWin.autECLConnMgr ")

' Assume we've got connections open and the Hand parm was obtained earlier
Mgr.StopConnection Hand, "saveprofile=no"
'or
Mgr.StopConnection "B", "saveprofile=no"
```

autECLConnMgr イベント

以下のイベントは autECLConnMgr に有効です。

void NotifyStartEvent(By Val Handle As Variant, By Val Started As Boolean) NotifyStartError(By Val ConnHandle As Variant) void NotifyStartStop(Long Reason)

NotifyStartEvent

セッションが開始あるいは停止しました。

プロトタイプ

void NotifyStartEvent(By Val Handle As Variant、By Val Started As Boolean)

注: Visual Basic では、このサブルーチンが正しく作成されます。

パラメーター

By Val Handle As Variant

開始または停止したセッションのハンドルです。

By Val Started As Boolean

セッションが開始済みの場合は True で、それ以外の場合は False です。

例

例については、イベント処理の例 (ページ 270)を参照してください。

NotifyStartError

このイベントは、イベント処理でエラーが発生したときに起こります。

プロトタイプ

NotifyStartError(By Val ConnHandle As Variant)

注: Visual Basic では、このサブルーチンが正しく作成されます。

パラメーター

なし

例

例については、イベント処理の例 (ページ 270)を参照してください。

NotifyStartStop

このイベントは、イベント処理が停止したときに起こります。

プロトタイプ

void NotifyStartStop(Long Reason)

パラメーター

Long Reason

停止の理由コード。現在は、これは常に 0 です。

イベント処理の例

以下は、開始イベントの実装方法についての簡単な例です。

```
Option Explicit
Private WithEvents mCmgr As autECLConnMgr 'AutConnMgr added as reference
dim mSess as object

sub main()
'Create Objects
Set mCmgr = New autECLConnMgr
Set mSess = CreateObject("ZIEWin.autECLSession")
mCmgr.RegisterStartEvent 'register for PS Updates

' Display your form or whatever here  (this should be a blocking call, otherwise sub just ends
call DisplayGUI()
mCmgr.UnregisterStartEvent
set mCmgr = Nothing
set mSess = Nothing
End Sub

'This sub will get called when a session is started or stopped
Private Sub mCmgr_NotifyStartEvent(Handle as long, bStarted as Boolean)
' do your processing here
if (bStarted) then
mSess.SetConnectionByHandle Handle
end if
End Sub

'This event occurs if an error happens
Private Sub mCmgr_NotifyStartError()
'Do any error processing here
End Sub

Private Sub mCmgr_NotifyStartStop(Reason As Long)
'Do any stop processing here
End Sub
```

autECLFieldList クラス

autECLFieldList は、 エミュレーター表示スペースでのフィールドの操作を実行します。このオブジェクトは、単体として独立してはいません。これは autECLPS に含まれ、autECLPS オブジェクトを介してのみアクセスされます。autECLPS は、独立している場合も、autECLSession に含まれる場合もあります。

autECLFieldList には、表示スペースの すべてのフィールドの集合が含まれます。集合の各要素には、集合要素プロパティー (ページ 271)に示されている要素が含まれます。

autECLFieldList オブジェクトは、Refresh メソッドが 呼び出されたときに表示スペースにあった 静的スナップショットを提供します。

注: 確実に現在のフィールド・データを取得するには、 要素にアクセスする前に autECLFieldList オブジェクトで Refresh メソッド を呼び出さなければなりません。一度 Refresh を呼び出せば、集合の内容をすべて調べることができます。

プロパティー

このセクションでは、autECLFieldList オブジェクトの プロパティーおよび集合要素プロパティーについて説明します。

タイプ	名前	属性
Long (長形式)	カウント	読み取り専用

以下のプロパティーは集合要素プロパティーであり、 リストの各項目に有効です。

タイプ	名前	属性
長形式	StartRow	読み取り専用
長形式	StartCol	読み取り専用
長形式	EndRow	読み取り専用
長形式	EndCol	読み取り専用
長形式	長さ	読み取り専用
ブール値	変更日	読み取り専用
ブール値	保護されています	読み取り専用
ブール値	数字	読み取り専用
ブール値	HighIntensity	読み取り専用
ブール値	PenDetectable	読み取り専用
ブール値	表示	読み取り専用

カウント

このプロパティーは、Refresh メソッドへの最後の 呼び出しでの autECLFieldList 集合にあるフィールドの 数です。Count は、Long データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
Dim NumFields as long
Dim autECLPSObj as Object
Dim autECLConnList as Object
Set autECLPSObj = CreateObject("ZIEWin.autECLPS")
Set autECLConnList = CreateObject("ZIEWin.autECLConnList")

' Initialize the connection
autECLConnList.Refresh
autECLPSObj.SetConnectionByHandle(autECLConnList(1).Handle)

' Build the list and get the number of fields
autECLPSObj.autECLFieldList.Refresh(1)
NumFields = autECLPSObj.autECLFieldList.Count
```

StartRow

この集合要素プロパティーは、autECLFieldList 集合で 与えられたフィールドの最初の文字の行位置です。StartRow は、Long データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
Dim StartRow as Long
Dim StartCol as Long
Dim autECLPSObj as Object
Dim autECLConnList as Object
Set autECLPSObj = CreateObject("ZIEWin.autECLPS")
Set autECLConnList = CreateObject("ZIEWin.autECLConnList")

' Initialize the connection
autECLConnList.Refresh
autECLPSObj.SetConnectionByHandle(autECLConnList(1).Handle)

' Build the list and get the number of fields
autECLPSObj.autECLFieldList.Refresh(1)
If (Not autECLPSObj.autECLFieldList.Count = 0 ) Then
  StartRow = autECLPSObj.autECLFieldList(1).StartRow
  StartCol = autECLPSObj.autECLFieldList(1).StartCol
Endif
```

StartCol

この集合要素プロパティーは、autECLFieldList 集合で 与えられたフィールドの最初の文字の桁位置です。StartCol は Long データ型 で読み取り専用です。以下の例は、このプロパティーを示しています。

```
Dim StartRow as Long
Dim StartCol as Long
Dim autECLPSObj as Object
Dim autECLConnList as Object
Set autECLPSObj = CreateObject("ZIEWin.autECLPS")
Set autECLConnList = CreateObject("ZIEWin.autECLConnList")
```

```
' Initialize the connection
autECLConnList.Refresh
autECLPSObj.SetConnectionByHandle(autECLConnList(1).Handle)

' Build the list and get the number of fields
autECLPSObj.autECLFieldList.Refresh(1)
If (Not autECLPSObj.autECLFieldList.Count = 0 ) Then
  StartRow = autECLPSObj.autECLFieldList(1).StartRow
  StartCol = autECLPSObj.autECLFieldList(1).StartCol
Endif
```

EndRow

この集合要素プロパティーは、autECLFieldList 集合で 与えられたフィールドの最後の文字の行位置です。EndRow は Long データ型で 読み取り専用です。以下の例は、このプロパティーを示しています。

```
Dim EndRow as Long
Dim EndCol as Long
Dim autECLPSObj as Object
Dim autECLConnList as Object
Set autECLPSObj = CreateObject("ZIEWin.autECLPS")
Set autECLConnList = CreateObject("ZIEWin.autECLConnList")

' Initialize the connection
autECLConnList.Refresh
autECLPSObj.SetConnectionByHandle(autECLConnList(1).Handle)

' Build the list and get the number of fields
autECLPSObj.autECLFieldList.Refresh(1)
If (Not autECLPSObj.autECLFieldList.Count = 0 ) Then
  EndRow = autECLPSObj.autECLFieldList(1).EndRow
  EndCol = autECLPSObj.autECLFieldList(1).EndCol
Endif
```

EndCol

この集合要素プロパティーは、autECLFieldList 集合で 与えられたフィールドの最後の文字の桁位置です。EndCol は、Long データ型で 読み取り専用です。以下の例は、このプロパティーを示しています。

```
Dim EndRow as Long
Dim EndCol as Long
Dim autECLPSObj as Object
Dim autECLConnList as Object
Set autECLPSObj = CreateObject("ZIEWin.autECLPS")
Set autECLConnList = CreateObject("ZIEWin.autECLConnList")

' Initialize the connection
autECLConnList.Refresh
autECLPSObj.SetConnectionByHandle(autECLConnList(1).Handle)

' Build the list and get the number of fields
autECLPSObj.autECLFieldList.Refresh(1)
If (Not autECLPSObj.autECLFieldList.Count = 0 ) Then
  EndRow = autECLPSObj.autECLFieldList(1).EndRow
```

```
  EndCol = autECLPSObj.autECLFieldList(1).EndCol
Endif
```

長さ

この集合要素プロパティーは、autECLFieldList 集合で 与えられたフィールドの長さです。Length は、Long データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
Dim Len as Long
Dim autECLPSObj as Object
Dim autECLConnList as Object
Set autECLPSObj = CreateObject("ZIEWin.autECLPS")
Set autECLConnList = CreateObject("ZIEWin.autECLConnList")

' Initialize the connection
autECLConnList.Refresh
autECLPSObj.SetConnectionByHandle(autECLConnList(1).Handle)

' Build the list and get the number of fields
autECLPSObj.autECLFieldList.Refresh(1)
If (Not autECLPSObj.autECLFieldList.Count = 0 ) Then
  Len = autECLPSObj.autECLFieldList(1).Length
Endif
```

変更日

この集合要素プロパティーは、autECLFieldList 集合内の与えられたフィールドに 修正された属性があるかどうかを示しています。Modified は、Boolean データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
Dim autECLPSObj as Object
Dim autECLConnList as Object
Set autECLPSObj = CreateObject("ZIEWin.autECLPS")
Set autECLConnList = CreateObject("ZIEWin.autECLConnList")

' Initialize the connection
autECLConnList.Refresh
autECLPSObj.SetConnectionByHandle(autECLConnList(1).Handle)

' Build the list and get the number of fields
autECLPSObj.autECLFieldList.Refresh(1)
If (Not autECLPSObj.autECLFieldList.Count = 0 ) Then
  If ( autECLPSObj.autECLFieldList(1).Modified ) Then
    ' do whatever
  Endif
Endif
```

保護されています

この集合要素プロパティーは、autECLFieldList 集合内の与えられたフィールドに 保護属性があるかどうかを示しています。Protected は、Boolean データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
Dim autECLPSObj as Object
Dim autECLConnList as Object
Set autECLPSObj = CreateObject("ZIEWin.autECLPS")
Set autECLConnList = CreateObject("ZIEWin.autECLConnList")

' Initialize the connection
autECLConnList.Refresh
autECLPSObj.SetConnectionByHandle(autECLConnList(1).Handle)

' Build the list and get the number of fields
autECLPSObj.autECLFieldList.Refresh(1)
If (Not autECLPSObj.autECLFieldList.Count = 0 ) Then
  If ( autECLPSObj.autECLFieldList(1).Protected ) Then
    ' do whatever
  Endif
Endif
```

数字

この集合要素プロパティーは、autECLFieldList 集合内の与えられたフィールドに 数字入力専用の属性があるかどうかを示します。Numeric は、Boolean データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
Dim autECLPSObj as Object
Dim autECLConnList as Object
Set autECLPSObj = CreateObject("ZIEWin.autECLPS")
Set autECLConnList = CreateObject("ZIEWin.autECLConnList")

' Initialize the connection
autECLConnList.Refresh
autECLPSObj.SetConnectionByHandle(autECLConnList(1).Handle)

' Build the list and get the number of fields
autECLPSObj.autECLFieldList.Refresh(1)
If (Not autECLPSObj.autECLFieldList.Count = 0 ) Then
  If ( autECLPSObj.autECLFieldList(1).Numeric ) Then
    ' do whatever
  Endif
Endif
```

HighIntensity

この集合要素プロパティーは、autECLFieldList 集合内の与えられたフィールドに 高輝度属性があるかどうかを示しています。HighIntensity は、Boolean データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
Dim autECLPSObj as Object
Dim autECLConnList as Object
Set autECLPSObj = CreateObject("ZIEWin.autECLPS")
Set autECLConnList = CreateObject("ZIEWin.autECLConnList")

' Initialize the connection
autECLConnList.Refresh
```

```
autECLPSObj.SetConnectionByHandle(autECLConnList(1).Handle)

' Build the list and get the number of fields
autECLPSObj.autECLFieldList.Refresh(1)
If (Not autECLPSObj.autECLFieldList.Count = 0 ) Then
  If ( autECLPSObj.autECLFieldList(1).HighIntensity ) Then
    ' do whatever
  Endif
Endif
```

PenDetectable

この集合要素プロパティーは、autECLFieldList 集合内の 与えられたフィールドにペン検出可能属性があるかどうかを 示しています。PenDetectable は、Boolean データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
Dim autECLPSObj as Object
Dim autECLConnList as Object
Set autECLPSObj = CreateObject("ZIEWin.autECLPS")
Set autECLConnList = CreateObject("ZIEWin.autECLConnList")

' Initialize the connection
autECLConnList.Refresh
autECLPSObj.SetConnectionByHandle(autECLConnList(1).Handle)

' Build the list and get the number of fields
autECLPSObj.autECLFieldList.Refresh(1)
If (Not autECLPSObj.autECLFieldList.Count = 0 ) Then
  If ( autECLPSObj.autECLFieldList(1).PenDetectable ) Then
    ' do whatever
  Endif
Endif
```

表示

この集合要素プロパティーは、autECLFieldList 集合内の与えられたフィールドに 表示属性があるかどうかを示します。Display は、Boolean データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
Dim autECLPSObj as Object
Dim autECLConnList as Object
Set autECLPSObj = CreateObject("ZIEWin.autECLPS")
Set autECLConnList = CreateObject("ZIEWin.autECLConnList")

' Initialize the connection
autECLConnList.Refresh
autECLPSObj.SetConnectionByHandle(autECLConnList(1).Handle)

' Build the list and get the number of fields
autECLPSObj.autECLFieldList.Refresh(1)
If (Not autECLPSObj.autECLFieldList.Count = 0 ) Then
  If ( autECLPSObj.autECLFieldList(1).Display ) Then
    ' do whatever
```

```
  Endif
Endif
```

autECLFieldList メソッド

以下のセクションで、autECLFieldList オブジェクトに 有効なメソッドを説明します。

void Refresh() Object FindFieldByRowCol(Long Row, Long Col) Object FindFieldByText(String text, [optional] Long Direction, [optional] Long StartRow, [optional] Long StartCol)

集合要素メソッド

以下の集合要素メソッドは、リスト内の各項目に有効です。

String GetText() void SetText(String Text)

最新表示

Refresh メソッドは、 すべてのフィールドのスナップショットを取得します。

注: 確実に最新のフィールド・データを取得するには、 フィールド集合にアクセスする前に Refresh メソッドを呼び出さなければなりません。

プロトタイプ

void Refresh()

パラメーター

なし

戻り値

なし

例

以下の例は、所定の表示スペースについて、 すべてのフィールドのスナップショットを取得する方法を示しています。

```
Dim NumFields as long
Dim autECLPSObj as Object
Dim autECLConnList as Object
Set autECLPSObj = CreateObject("ZIEWin.autECLPS")
Set autECLConnList = CreateObject("ZIEWin.autECLConnList")
```

```
' Initialize the connection
autECLConnList.Refresh
autECLPSObj.SetConnectionByHandle(autECLConnList(1).Handle)

' Build the list and get the number of fields
autECLPSObj.autECLFieldList.Refresh()
NumFields = autECLPSObj.autECLFieldList.Count
```

FindFieldByRowCol

このメソッドは、所定の行および桁の 座標を含むフィールドを autECLFieldList オブジェクトで検索します。戻される値は、autECLFieldList 集合の集合要素オブジェクトです。

プロトタイプ

Object FindFieldByRowCol(Long Row, Long Col)

パラメーター

Long Row

検索するフィールド行。

Long Col

検索するフィールド桁。

戻り値

オブジェクト

autECLFieldList 集合項目のディスパッチ・オブジェクト。

例

以下の例は、所定の行および桁の座標を含む フィールドを autECLFieldList オブジェクトで検索する方法を示しています。

```
Dim autECLPSObj as Object
Dim autECLConnList as Object
Dim FieldElement as Object

Set autECLPSObj = CreateObject("ZIEWin.autECLPS")
Set autECLConnList = CreateObject("ZIEWin.autECLConnList)

' Initialize the connection
autECLConnList.Refresh
autECLPSObj.SetConnectionByHandle(autECLConnList(1).Handle)

' Build the list and search for field at row 2 col 1
autECLPSObj.autECLFieldList.Refresh(1)
```

```
Set FieldElement = autECLPSObj.autECLFieldList.FindFieldByRowCol( 2, 1 )
FieldElement.SetText("HCL")
```

FindFieldByText

このメソッドは、**テキスト**として渡されたストリングを含むフィールドを autECLFieldList オブジェクトで検索します。戻される値は、autECLFieldList 集合の集合要素オブジェクトです。

プロトタイプ

Object FindFieldByText(String Text, [optional] Long Direction, [optional] Long StartRow, [optional] Long StartCol)

パラメーター

String Text

検索するテキスト・ストリング。

Long StartRow

検索を開始する表示スペース内の行位置。

Long StartCol

検索を開始する表示スペース内の桁位置。

Long Direction

検索の方向。前方検索は **1**、後方検索は **2** を指定します。

戻り値

オブジェクト

autECLFieldList 集合項目のディスパッチ・オブジェクト。

例

以下の例は、テキストとして渡されたストリングを含むフィールド を autECLFieldList オブジェクトで検索する方法を示しています。

```
Dim autECLPSObj as Object
Dim autECLConnList as Object
Dim FieldElement as Object

Set autECLPSObj = CreateObject("ZIEWin.autECLPS")
Set autECLConnList = CreateObject("ZIEWin.autECLConnList")

' Initialize the connection
autECLConnList.Refresh
autECLPSObj.SetConnectionByHandle(autECLConnList(1).Handle)

' Build the list and search for field with text
autECLPSObj.autECLFieldList.Refresh(1)
```

```
set FieldElement = autECLPSObj.autECLFieldList.FindFieldByText "HCL"

' Or... search starting at row 2 col 1
set FieldElement = autECLPSObj.autECLFieldList.FindFieldByText "HCL", 2, 1
' Or... search starting at row 2 col 1 going backwards
set FieldElement = autECLPSObj.autECLFieldList.FindFieldByText "HCL", 2, 2, 1

FieldElement.SetText("Hello.")
```

GetText

集合要素メソッド GetText は、autECLFieldList 項目内の 所定のフィールドの文字を検索します。

プロトタイプ

String GetText()

パラメーター

なし

戻り値

ストリング

フィールド・テキスト

例

以下の例は、GetText メソッドを使用する方法を示しています。

```
Dim autECLPSObj as Object
Dim TextStr as String

' Initialize the connection
Set autECLPSObj = CreateObject("ZIEWin.autECLPS")
autECLPSObj.SetConnectionByName("A")

autECLPSObj.autECLFieldList.Refresh()
TextStr = autECLPSObj.autECLFieldList(1).GetText()
```

SetText

このメソッドは、autECLFieldList 項目内の所定のフィールドに、 テキストとして渡された文字ストリングを入れます。このテキストがフィールドの長さを超える場合、テキストは 切り捨てられます。

プロトタイプ

void SetText(String Text)

パラメーター

String text

フィールドに設定するストリング。

戻り値

なし

例

以下の例は、autECLFieldList 項目内のフィールドに、 テキストとして渡された文字ストリングを入れる方法を示しています。

```
Dim NumFields as Long
Dim autECLPSObj as Object
Dim autECLConnList as Object
Set autECLPSObj = CreateObject("ZIEWin.autECLPS")
Set autECLConnList = CreateObject("ZIEWin.autECLConnList")

' Initialize the connection
autECLConnList.Refresh
autECLPSObj.SetConnectionByHandle(autECLConnList(1).Handle)

' Build the list and set the first field with some text
autECLPSObj.autECLFieldList.Refresh(1)
autECLPSObj.autECLFieldList(1).SetText("HCL is a cool company")
```

autECLOIA クラス

autECLOIA オブジェクトは、 ホスト・オペレーター情報域から状況を検索します。レジストリーでのその名前は ZIEWin.autECLOIA です。

最初に、作成したオブジェクトの接続を設定しなければなりません。SetConnectionByName または SetConnectionByHandle を使用して、オブジェクトを初期化します。接続は一度しか設定できません。接続が設定された後は、接続設定メソッドをさらに呼び出すと例外を引き起こします。また、接続を設定せずにプロパティーまたはメソッドにアクセスしようとしても、 例外が引き起こされます。

注: autECLSession オブジェクト内の autECLOIA オブジェクト は、autECLSession オブジェクトにより設定されます。

以下の例は、Visual Basic で autECLOIA オブジェクトを作成し 設定する方法を示します。

```
DIM  autECLOIA as Object

Set autECLOIA = CreateObject("ZIEWin.autECLOIA")
autECLOIA.SetConnectionByName("A")
```

プロパティー

このセクションでは、autECLOIA オブジェクトの プロパティーを説明します。

タイプ	名前	属性
ブール値	英数字	読み取り専用
ブール値	APL	読み取り専用
ブール値	UpperShift	読み取り専用
ブール値	数字	読み取り専用
ブール値	CapsLock	読み取り専用
ブール値	InsertMode	読み取り専用
ブール値	CommErrorReminder	読み取り専用
ブール値	MessageWaiting	読み取り専用
長形式	InputInhibited	読み取り専用
ストリング	名前	読み取り専用
長形式	ハンドル	読み取り専用
ストリング	ConnType	読み取り専用
長形式	CodePage	読み取り専用
ブール値	開始済み	読み取り専用
ブール値	CommStarted	読み取り専用
ブール値	APIEnabled	読み取り専用
ブール値	作動可能	読み取り専用
ブール値	NumLock	読み取り専用

英数字

このプロパティーは、オペレーター情報域を照会してカーソル位置のフィールドが 英数字かどうかを判別します。Alphanumeric は、Boolean データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM  autECLOIA as Object
DIM  autECLConnList as Object

Set autECLOIA = CreateObject("ZIEWin.autECLOIA")
Set autECLConnList = CreateObject("ZIEWin.autECLConnList")

' Initialize the connection
autECLConnList.Refresh
autECLOIA.SetConnectionByHandle(autECLConnList(1).Handle)

If autECLOIA.Alphanumeric Then...
```

APL

このプロパティーは、オペレーター情報域を照会して、キーボードが APL モードかどうかを 判別します。APL は、Boolean データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM  autECLOIA as Object
DIM  autECLConnList as Object

Set autECLOIA = CreateObject("ZIEWin.autECLOIA")
Set autECLConnList = CreateObject("ZIEWin.autECLConnList")

' Initialize the connection
autECLConnList.Refresh
autECLOIA.SetConnectionByHandle(autECLConnList(1).Handle)

' Check if the keyboard is in APL mode
if autECLOIA.APL Then...
```

カタカナ

このプロパティーは、オペレーター情報域を照会してカタカナ文字が使用可能かどうかを 判別します。Katakana は、Boolean データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM  autECLOIA as Object
DIM  autECLConnList as Object

Set autECLOIA = CreateObject("ZIEWin.autECLOIA")
Set autECLConnList = CreateObject("ZIEWin.autECLConnList")

' Initialize the connection
autECLConnList.Refresh
autECLOIA.SetConnectionByHandle(autECLConnList(1).Handle)

' Check if Katakana characters are available
if autECLOIA.Katakana Then...
```

Hiragana

このプロパティーは、オペレーター情報域を照会して、ひらがな文字が使用可能かどうかを 判別します。Hiragana は、Boolean データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM  autECLOIA as Object
DIM  autECLConnList as Object

Set autECLOIA = CreateObject("ZIEWin.autECLOIA")
Set autECLConnList = CreateObject("ZIEWin.autECLConnList")

' Initialize the connection
autECLConnList.Refresh
autECLOIA.SetConnectionByHandle(autECLConnList(1).Handle)

' Check if Hiragana characters are available
if autECLOIA.Hiragana Then...
```

UpperShift

このプロパティーは、オペレーター情報域を照会して、キーボードが上段シフト・モードかどうかを 判別します。Uppershift は、Boolean データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM  autECLOIA as Object
DIM  autECLConnList as Object

Set autECLOIA = CreateObject("ZIEWin.autECLOIA")
Set autECLConnList = CreateObject("ZIEWin.autECLConnList")

' Initialize the connection
autECLConnList.Refresh
autECLOIA.SetConnectionByHandle(autECLConnList(1).Handle)

' Check if the keyboard is in uppershift mode
If autECLOIA.UpperShift then...
```

数字

このプロパティーは、オペレーター情報域を照会して、カーソル位置のフィールドが数字かどうかを 判別します。Numeric は、Boolean データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM  autECLOIA as Object
DIM  autECLConnList as Object

Set autECLOIA = CreateObject("ZIEWin.autECLOIA")
Set autECLConnList = CreateObject("ZIEWin.autECLConnList")

' Initialize the connection
autECLConnList.Refresh
autECLOIA.SetConnectionByHandle(autECLConnList(1).Handle)

' Check if the cursor location is a numeric field
If autECLOIA.Numeric Then...
```

CapsLock

このプロパティーは、オペレーター情報域を照会して、キーボード CapsLock キーがオンかどうかを 判別します。CapsLock は、Boolean データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM  autECLOIA as Object
DIM  autECLConnList as Object

Set autECLOIA = CreateObject("ZIEWin.autECLOIA")
Set autECLConnList = CreateObject("ZIEWin.autECLConnList")

' Initialize the connection
autECLConnList.Refresh
autECLOIA.SetConnectionByHandle(autECLConnList(1).Handle)
```

```
' Check if the caps lock
If autECLOIA.CapsLock Then...
```

InsertMode

このプロパティーは、オペレーター情報域を照会して、キーボードが挿入モードかどうかを 判別します。InsertMode は、Boolean データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM  autECLOIA as Object
DIM  autECLConnList as Object

Set autECLOIA = CreateObject("ZIEWin.autECLOIA")
Set autECLConnList = CreateObject("ZIEWin.autECLConnList")

' Initialize the connection
autECLConnList.Refresh
autECLOIA.SetConnectionByHandle(autECLConnList(1).Handle)

' Check if in insert mode
If autECLOIA.InsertMode Then...
```

CommErrorReminder

このプロパティーは、オペレーター情報域を照会して、通信エラー状況メッセージ条件が存在するかどうかを 判別します。CommErrorReminder は、Boolean データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM  autECLOIA as Object
DIM  autECLConnList as Object

Set autECLOIA = CreateObject("ZIEWin.autECLOIA")
Set autECLConnList = CreateObject("ZIEWin.autECLConnList")

' Initialize the connection
autECLConnList.Refresh
autECLOIA.SetConnectionByHandle(autECLConnList(1).Handle)

' Check if comm error
If autECLOIA.CommErrorReminder Then...
```

MessageWaiting

このプロパティーは、オペレーター情報域を照会して、 メッセージ待ち標識がオンかどうかを判別します。これは、5250 接続にのみ起こります。MessageWaiting は、Boolean データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM  autECLOIA as Object
DIM  autECLConnList as Object
Set autECLOIA = CreateObject("ZIEWin.autECLOIA")
Set autECLConnList = CreateObject("ZIEWin.autECLConnList")
```

```
' Initialize the connection
autECLConnList.Refresh
autECLOIA.SetConnectionByHandle(autECLConnList(1).Handle)

' Check if message waiting
If autECLOIA.MessageWaiting Then...
```

InputInhibited

このプロパティーは、オペレーター情報域を照会して、キーボード入力が使用禁止かどうかを 判別します。InputInhibited は、Long データ型で読み取り専用です。以下の表は、InputInhibited に有効な値を示しています。

名前	値
使用可能	0
システム待機	1
通信チェック	2
プログラム・チェック	3
マシン・チェック	4
その他の禁止	5

以下の例は、このプロパティーを示しています。

```
DIM  autECLOIA as Object
DIM  autECLConnList as Object
Set autECLOIA = CreateObject("ZIEWin.autECLOIA")
Set autECLConnList = CreateObject("ZIEWin.autECLConnList")

' Initialize the connection
autECLConnList.Refresh
autECLOIA.SetConnectionByHandle(autECLConnList(1).Handle)

' Check if input inhibited
If not autECLOIA.InputInhibited = 0 Then...
```

名前

このプロパティーは、autECLOIA が設定された接続の接続名ストリングです。Z and I Emulator for Windows は、短い文字 ID (A から Z、または a から z) のみをストリングで戻します。Z and I Emulator for Windows 接続でオープンできるのは、1 つの名前につき 1 つしかありません。例えば、一度に 1 つの接続 A のみをオープンできます。Name は、String データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM  Name as String
DIM  Obj as Object
Set Obj = CreateObject("ZIEWin.autECLOIA")

' Initialize the connection
Obj.SetConnectionByName("A")
```

```
' Save the name
Name = Obj.Name
```

ハンドル

これは、autECLOIA オブジェクトが設定された接続のハンドルです。特定の 1 つのハンドルに対して 1 つしか、Z and I Emulator for Windows 接続をオープンできません。例えば、一度に 1 つの接続 A のみをオープンできます。Handle は、Long データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM  Obj as Object
Set Obj = CreateObject("ZIEWin.autECLOIA")

' Initialize the connection
Obj.SetConnectionByName("A")

' Save the handle
Hand = Obj.Handle
```

ConnType

これは、autECLOIA が設定された接続タイプです。このタイプは、時間の経過とともに変更する場合が あります。ConnType は、String データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM  Type as String
DIM  Obj as Object

Set Obj = CreateObject("ZIEWin.autECLOIA")

' Initialize the connection
Obj.SetConnectionByName("A")
' Save the type
Type = Obj.ConnType
```

ConnType プロパティーの接続タイプは、以下のとおりです。

戻されるストリング	意味
DISP3270	3270 表示装置
DISP5250	5250 ディスプレイ
PRNT3270	3270 印刷装置
PRNT5250	5250 プリンター
ASCII	VT エミュレーション

CodePage

これは、autECLOIA が設定された接続のコード・ページです。このコード・ページは、時間の経過とともに変更される場合があります。CodePage は、Long データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM  CodePage as Long
DIM  Obj as Object
Set Obj = CreateObject("ZIEWin.autECLOIA")

' Initialize the connection
Obj.SetConnectionByName("A")
' Save the code page
CodePage = Obj.CodePage
```

開始済み

これは、エミュレーター・ウィンドウが開始されたかどうかを示します。ウィンドウがオープンしている場合、値は True です。 その他の場合は False です。Started は、Boolean データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM  Hand as Long
DIM  Obj as Object

Set Obj = CreateObject("ZIEWin.autECLOIA")

' Initialize the connection
Obj.SetConnectionByName("A")

' This code segment checks to see if A is started.
' The results are sent to a text box called Result.
If Obj.Started = False Then
  Result.Text = "No"
Else
  Result.Text = "Yes"
End If
```

CommStarted

これは、ホストへの接続の状況を示しています。ホストが接続されている場合、値は True です。 その他の場合は False です。CommStarted は、Boolean データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM  Hand as Long
DIM  Obj as Object

Set Obj = CreateObject("ZIEWin.autECLOIA")

' Initialize the connection
Obj.SetConnectionByName("A")

' This code segment checks to see if communications are connected
' for A.  The results are sent to a text box called
' CommConn.
If Obj.CommStarted = False Then
    CommConn.Text = "No"
Else
    CommConn.Text = "Yes"
End If
```

APIEnabled

これは、エミュレーターが API 使用可能かどうかを示します。API 設定 (Z and I Emulator for Windows のウィンドウでは、**「ファイル」->「API の設定」**を選択) の状態に応じて、接続が可能な場合と、接続できない場合があります。エミュレーターが使用可能の場合には、True です。その他の場合には、False です。APIEnabled は、Boolean データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM  Hand as Long
DIM  Obj as Object

Set Obj = CreateObject("ZIEWin.autECLOIA")

' Initialize the connection
Obj.SetConnectionByName("A")

' This code segment checks to see if A is API enabled.
' The results are sent to a text box called Result.
If Obj.APIEnabled = False Then
  Result.Text = "No"
Else
  Result.Text = "Yes"
End If
```

作動可能

これは、エミュレーター・ウィンドウが 開始されていて、API 使用可能で、そして接続されているかどうかを示します。このプロパティーは、3 つのすべてのプロパティーを確認します。エミュレーターが準備できている場合には、値は True です。 その他の場合には、False です。Ready は、Boolean データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM  Hand as Long
DIM  Obj as Object

Set Obj = CreateObject("ZIEWin.autECLOIA")

' Initialize the connection
Obj.SetConnectionByName("A")

' This code segment checks to see if A is ready.
' The results are sent to a text box called Result.
If Obj.Ready = False Then
  Result.Text = "No"
Else
  Result.Text = "Yes"
End If
```

NumLock

このプロパティーは、オペレーター情報域を照会して、 キーボード NumLock キーがオンかどうかを判別します。NumLock は、Boolean データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM autECLOIA as Object
    DIM autECLConnList as Object

    Set autECLOIA = CreateObject ("ZIEWin.autECLOIA")
Set autECLConnList = CreateObject ("ZIEWin.autECLConnList")

' Initialize the connection
autECLConnList.Refresh
autECLOIA.SetConnectionByFHandle (autECLConnList (1) .Handle)

' Check if the num lock is on
If autECLOIA.NumLock Then . . .
```

autECLOIA メソッド

以下のセクションで、autECLOIA オブジェクトに有効なメソッドを説明します。

void RegisterOIAEvent() void UnregisterOIAEvent() void SetConnectionByName (String Name) void SetConnectionByHandle (Long Handle) void StartCommunication() void StopCommunication() Boolean WaitForInputReady([optional] Variant TimeOut) Boolean WaitForSystemAvailable([optional] Variant TimeOut) Boolean WaitForAppAvailable([optional] Variant TimeOut) Boolean WaitForTransition([optional] Variant Index, [optional] Variant timeout) void CancelWaits()

RegisterOIAEvent

このメソッドは、すべての OIA イベントの通知を受け取るためにオブジェクトを登録します。

プロトタイプ

void RegisterOIAEvent()

パラメーター

なし

戻り値

なし

例

例については、イベント処理の例 (ページ 298)を参照してください。

UnregisterOIAEvent

OIA イベント処理を終了します。

プロトタイプ

void UnregisterOIAEvent()

パラメーター

なし

戻り値

なし

例

例については、イベント処理の例 (ページ 298)を参照してください。

SetConnectionByName

この SetConnectionByName メソッドは、接続名を使用して、 新しく作成された autECLOIA オブジェクトの接続を設定します。Z and I Emulator for Windows では、この接続名は、短い接続 ID (文字 A から Z、または a から z) です。Z and I Emulator for Windows 接続でオープンできるのは、1 つの名前につき 1 つしかありません。例えば、一度に 1 つの接続 A のみをオープンできます。

注: autECLSession 内で autECLOIA オブジェクトを使用している場合には、 これを呼び出さないでください。

プロトタイプ

void SetConnectionByName (String Name)

パラメーター

String 型、Name 型

1 文字の接続のストリング短縮名 (A から Z、または a から z)。

戻り値

なし

例

以下の例は、新しく作成された autECLOIA オブジェクトの接続を設定するために、 接続ハンドルを使用する方法を示します。

```
DIM  autECLOIA as Object

Set autECLOIA = CreateObject("ZIEWin.autECLOIA")

' Initialize the connection
autECLOIA.SetConnectionByName("A")
' For example, see if its num lock is on
If ( autECLOIA.NumLock = True ) Then
  'your logic here...
Endif
```

SetConnectionByHandle

この SetConnectionByHandle メソッドは、 接続ハンドルを使用して、新しく作成された autECLOIA オブジェクトの接続を設定します。Z and I Emulator for Windows では、この接続ハンドルは Long integer です。特定の 1 つのハンドルに対して 1 つしか、Z and I Emulator for Windows 接続をオープンできません。例えば、一度に 1 つの接続 A のみをオープンできます。

注: autECLSession 内で autECLOIA オブジェクトを使用している場合には、 これを呼び出さないでください。

```
DIM  autECLOIA as Object
DIM  autECLConnList as Object

Set autECLOIA = CreateObject("ZIEWin.autECLOIA")
Set autECLConnList = CreateObject("ZIEWin.autECLConnList")

' Initialize the connection
autECLConnList.Refresh
autECLOIA.SetConnectionByHandle(autECLConnList(1).Handle)
' For example, see if its num lock is on
If ( autECLOIA.NumLock = True ) Then
  'your logic here...
Endif
```

プロトタイプ

void SetConnectionByHandle(Long Handle)

パラメーター

Long Handle

オブジェクトに設定される接続の Long integer 値。

戻り値

なし

例

以下の例は、接続ハンドルを使用して、新しく作成された autELCOIA オブジェクトの接続を 設定する方法を示します。

StartCommunication

StartCommunication 集合要素メソッドは、ホスト・データ・ストリームに ZIEWin エミュレーターを接続します。これは、ZIEWin エミュレーター**「通信」**メニューを表示して**「接続」**を選択した場合と同じ結果になります。

プロトタイプ

void StartCommunication()

パラメーター

なし

戻り値

なし

例

なし

```
Dim OIAObj as Object
Dim autECLConnList as Object

Set autECLConnList = CreateObject("ZIEWin.autECLConnList")
Set OIAObj = CreateObject("ZIEWin.autECLOIA")

' Initialize the session
autECLConnList.Refresh
OIAObj.SetConnectionByHandle(autECLConnList(1).Handle)

OIAObj.StartCommunication()
```

StopCommunication

StopCommunication 集合要素メソッドは、ホスト・データ・ストリームから ZIEWin エミュレーターを切断します。これは、ZIEWin エミュレーター**「通信」**メニューを表示して**「切断」**を選択した場合と同じ結果になります。

プロトタイプ

void StopCommunication()

パラメーター

なし

戻り値

なし

例

以下の例は、ZIEWin エミュレーター・セッションをホストへ接続する方法を示しています。

```
Dim OIAObj as Object
Dim autECLConnList as Object

Set autECLConnList = CreateObject("ZIEWin.autECLConnList")
Set OIAObj = CreateObject("ZIEWin.autECLOIA")

' Initialize the session
autECLConnList.Refresh
OIAObj.SetConnectionByHandle(autECLConnList(1).Handle)

OIAObj.StopCommunication()
```

WaitForInputReady

WaitForInputReady メソッドは、autECLOIA オブジェクトに関連した接続の OIA が、 この接続にキーボード入力の受け入れが可能であることを示すまで待機します。

プロトタイプ

Boolean WaitForInputReady([optional] Variant TimeOut)

パラメーター

Variant TimeOut

待機する時間の最大長 (ミリ秒)。このパラメーターはオプションです。デフォルトは Infinite (無期限) です。

戻り値

メソッドは、条件が合致していれば True を戻し、 タイムアウト値が超えていれば False を戻します。

例

```
Dim autECLOIAObj as Object

Set autECLOIAObj = CreateObject("ZIEWin.autECLOIA")
autECLOIAObj.SetConnectionByName("A")

if (autECLOIAObj.WaitForInputReady(10000)) then
msgbox "Ready for input"
else
msgbox "Timeout Occurred"
end if
```

WaitForSystemAvailable

WaitForSystemAvailable メソッドは、autECLOIA オブジェクトに関連した接続の OIA が、 その接続がホスト・システムに接続されていると指示するまで待機します。

プロトタイプ

Boolean WaitForSystemAvailable([optional] Variant TimeOut)

パラメーター

Variant TimeOut

待機する時間の最大長 (ミリ秒)。このパラメーターはオプションです。デフォルトは Infinite (無期限) です。

戻り値

メソッドは、条件が合致していれば True を戻し、 タイムアウト値が超えていれば False を戻します。

例

```
Dim autECLOIAObj as Object

Set autECLOIAObj = CreateObject("ZIEWin.autECLOIA")
autECLOIAObj.SetConnectionByName("A")

if (autECLOIAObj.WaitForSystemAvailable(10000)) then
msgbox "System Available"
else
msgbox "Timeout Occurred"
end if
```

WaitForAppAvailable

WaitForAppAvailable メソッドは、autECLOIA オブジェクトに関連した接続の OIA が そのアプリケーションが処理中であることを示しているまで待機します。

プロトタイプ

Boolean WaitForAppAvailable([optional] Variant TimeOut)

パラメーター

Variant TimeOut

待機する時間の最大長 (ミリ秒)。このパラメーターはオプションです。デフォルトは Infinite (無期限) です。

戻り値

メソッドは、条件が合致していれば True を戻し、 タイムアウト値が超えていれば False を戻します。

例

```
Dim autECLOIAObj as Object

Set autECLOIAObj = CreateObject("ZIEWin.autECLOIA")
autECLOIAObj.SetConnectionByName("A")

if (autECLOIAObj.WaitForAppAvailable (10000)) then
msgbox "Application is available"
else
msgbox "Timeout Occurred"
end if
```

WaitForTransition

WaitForTransition メソッドは、autECLOIA オブジェクトに関連した接続の 指定された OIA 位置が変更されるのを待ちます。

プロトタイプ

Boolean WaitForTransition([optional] Variant Index, [optional] Variant timeout)

パラメーター

Variant Index

モニターする OIA の 1 バイトの 16 進数位置。このパラメーターはオプションです。デフォルトは、3 です。

Variant TimeOut

待機する時間の最大長 (ミリ秒)。このパラメーターはオプションです。デフォルトは Infinite (無期限) です。

戻り値

メソッドは、条件が合致していれば True を戻し、 タイムアウト値が超えていれば False を戻します。

例

```
Dim autECLOIAObj as Object
Dim Index

Index = 03h

Set autECLOIAObj = CreateObject("ZIEWin.autECLOIA")
autECLOIAObj.SetConnectionByName("A")

if (autECLOIAObj.WaitForTransition(Index,10000)) then
    msgbox "Position " " Index " " of the OIA Changed"
else
    msgbox "Timeout Occurred"
end if
```

CancelWaits

現在活動待ちのメソッドを取り消します。

プロトタイプ

void CancelWaits()

パラメーター

なし

戻り値

なし

autECLOIA イベント

以下のイベントは autECLOIA に有効です。

void NotifyOIAEvent() void NotifyOIAError() void NotifyOIAStop(Long Reason)

NotifyOIAEvent

所定の OIA が発生しました。

プロトタイプ

void NotifyOIAEvent()

パラメーター

なし

例

例については、イベント処理の例 (ページ 298)を参照してください。

NotifyOIAError

このイベントは、OIA にエラーの発生したときに発生します。

プロトタイプ

void NotifyOIAError()

パラメーター

なし

例

例については、イベント処理の例 (ページ 298)を参照してください。

NotifyOIAStop

このイベントは、イベント処理が停止したときに起こります。

プロトタイプ

void NotifyOIAStop(Long Reason)

パラメーター

Long Reason

停止を表す Long Reason コード。現在は、これは常に 0 です。

イベント処理の例

以下は、OIA イベントの実装方法についての簡単な例です。

```
Option Explicit
Private WithEvents myOIA As autECLOIA 'AutOIA added as reference

sub main()
'Create Objects
Set myOIA = New AutOIA

Set myConnMgr = New AutConnMgr

myOIA.SetConnectionByName ("B") 'Monitor Session B for OIA Updates

myOIA.RegisterOIAEvent 'register for OIA Notifications

' Display your form or whatever here  (this should be a blocking call, otherwise sub just ends
call DisplayGUI()

'Clean up
myOIA.UnregisterOIAEvent

Private Sub myOIA_NotifyOIAEvent()
' do your processing here
End Sub
Private Sub myOIA_NotifyOIAError()
' do your processing here
End Sub
 'This event occurs when Communications Status Notification ends
Private Sub myOIA_NotifyOIAStop(Reason As Long)
'Do any stop processing here
End Sub
```

autECLPS クラス

autECLPS は、表示スペースでの操作を実行します。レジストリーでのその名前は ZIEWin.autECLPS です。

最初に、作成したオブジェクトの接続を設定しなければなりません。SetConnectionByName または SetConnectionByHandle を使用して、オブジェクトを初期化します。接続は一度しか設定できません。接続が設定された後は、SetConnection メソッドをさらに呼び出すと例外を引き起こします。また、接続を設定せずにプロパティーまたはメソッドにアクセスしようとしても、 例外が引き起こされます。

注:

1. 表示スペース内で、最初の行座標は行 1 で、 最初の桁座標は桁 1 です。したがって、最上行の左端の位置は、行 1 桁 1 になります。
2. autECLSession オブジェクト内の autECLPS オブジェクトは、 autECLSession オブジェクトにより設定されます。

以下に示すのは、Visual Basic で autECLPS オブジェクトを作成し設定する方法の例です。

```
DIM  autECLPSObj as Object
DIM  NumRows as Long
Set autECLPSObj = CreateObject("ZIEWin.autECLPS")
' Initialize the connection
autECLPSObj .SetConnectionByName("A")
' For example, get the number of rows in the PS
NumRows = autECLPSObj.NumRows
```

プロパティー

このセクションでは、autECLPS オブジェクトのプロパティーを説明します。

タイプ	名前	属性
オブジェクト	autECLFieldList	読み取り専用
長形式	NumRows	読み取り専用
長形式	NumCols	読み取り専用
長形式	CursorPosRow	読み取り専用
長形式	CursorPosCol	読み取り専用
ストリング	名前	読み取り専用
長形式	ハンドル	読み取り専用
ストリング	ConnType	読み取り専用
長形式	CodePage	読み取り専用
ブール値	開始済み	読み取り専用
ブール値	CommStarted	読み取り専用
ブール値	APIEnabled	読み取り専用
ブール値	作動可能	読み取り専用

autECLFieldList

これは、autECLPS オブジェクトに関連する 接続のフィールド集合オブジェクトです。詳しくは、autECLFieldList クラス (ページ 270)を参照してください。以下の例は、このオブジェクトを示しています。

```
Dim  autECLPSObj as Object
Dim  autECLConnList as Object
Set autECLPSObj = CreateObject("ZIEWin.autECLPS")
Set autECLConnList = CreateObject("ZIEWin.autECLConnList")
```

```
' Initialize the connection
autECLConnList.Refresh
autECLPSObj.SetConnectionByHandle(autECLConnList(1).Handle)

' Build the field list
CurPosCol = autECLPSObj.autECLFieldList.Refresh(1)
```

NumRows

これは、autECLPS オブジェクトに関連する接続の表示スペースでの 行の数です。NumRows は、Long データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
Dim  autECLPSObj as Object
Dim  autECLConnList as Object
Dim  Rows as Long
Set autECLPSObj = CreateObject("ZIEWin.autECLPS")
Set autECLConnList = CreateObject("ZIEWin.autECLConnList")

' Initialize the connection
autECLConnList.Refresh
autECLPSObj.SetConnectionByHandle(autECLConnList(1).Handle)
Rows = autECLPSObj.NumRows
```

NumCols

これは、autECLPS オブジェクトに関連する接続の表示スペースでの 桁の数です。NumCols は、Long データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
Dim  autECLPSObj as Object
Dim  autECLConnList as Object
Dim  Cols as Long
Set autECLPSObj = CreateObject("ZIEWin.autECLPS")
Set autECLConnList = CreateObject("ZIEWin.autECLConnList")

' Initialize the connection
autECLConnList.Refresh
autECLPSObj.SetConnectionByHandle(autECLConnList(1).Handle)
Cols = autECLPSObj.NumCols
```

CursorPosRow

これは、autECLPS オブジェクトに関連する接続の表示スペースでの カーソルの現在行位置です。CursorPosRow は Long データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
Dim  autECLPSObj as Object
Dim  autECLConnList as Object
Dim  CurPosRow as Long
Set autECLPSObj = CreateObject("ZIEWin.autECLPS")
Set autECLConnList = CreateObject("ZIEWin.autECLConnList")

' Initialize the connection
autECLConnList.Refresh
```

```
autECLPSObj.SetConnectionByHandle(autECLConnList(1).Handle)
CurPosRow = autECLPSObj.CursorPosRow
```

CursorPosCol

これは、autECLPS オブジェクトに関連する接続の表示スペースでの カーソルの現在桁位置です。CursorPosCol は、Long データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
Dim  autECLPSObj as Object
Dim  autECLConnList as Object
Dim  CurPosCol as Long
Set autECLPSObj = CreateObject("ZIEWin.autECLPS")
Set autECLConnList = CreateObject("ZIEWin.autECLConnList")

' Initialize the connection
autECLConnList.Refresh
autECLPSObj.SetConnectionByHandle(autECLConnList(1).Handle)
CurPosCol = autECLPSObj.CursorPosCol
```

名前

これは、autECLPS が設定された接続の接続名ストリングです。Z and I Emulator for Windows は、短い文字 ID (A から Z、または a から z) のみをストリングで戻します。Z and I Emulator for Windows 接続でオープンできるのは、1 つの名前につき 1 つしかありません。例えば、一度に 1 つの接続 A のみをオープンできます。Name は、String データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM  Name as String
DIM  Obj as Object
Set Obj = CreateObject("ZIEWin.autECLPS")

' Initialize the connection
Obj.SetConnectionByName("A")

' Save the name
Name = Obj.Name
```

ハンドル

これは、autECLPS オブジェクトが設定された接続のハンドルです。特定の 1 つのハンドルに対して 1 つしか、Z and I Emulator for Windows 接続をオープンできません。例えば、一度に 1 つの接続 A のみをオープンできます。Handle は、Long データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM  Obj as Object
Set Obj = CreateObject("ZIEWin.autECLPS")

' Initialize the connection
Obj.SetConnectionByName("A")

' Save the connection handle
Hand = Obj.Handle
```

ConnType

これは、autECLPS が設定された接続タイプです。この接続タイプは、時間の経過とともに変更する場合があります。ConnType は、String データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM  Type as String
DIM  Obj as Object

Set Obj = CreateObject("ZIEWin.autECLPS")

' Initialize the connection
Obj.SetConnectionByName("A")
' Save the type
Type = Obj.ConnType
```

ConnType プロパティーの接続タイプは、以下のとおりです。

戻されるストリング	意味
DISP3270	3270 表示装置
DISP5250	5250 ディスプレイ
PRNT3270	3270 印刷装置
PRNT5250	5250 プリンター
ASCII	VT エミュレーション

CodePage

これは、autECLPS が設定された接続のコード・ページです。このコード・ページは、時間の経過とともに変更される場合があります。CodePage は、Long データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM  CodePage as Long
DIM  Obj as Object
Set Obj = CreateObject("ZIEWin.autECLPS")

' Initialize the connection
Obj.SetConnectionByName("A")
' Save the code page
CodePage = Obj.CodePage
```

開始済み

これは、接続エミュレーター・ウィンドウが開始されたか どうかを示します。ウィンドウがオープンしている場合、値は True です。 その他の場合は False です。Started は、Boolean データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM  Hand as Long
DIM  Obj as Object
```

```
Set Obj = CreateObject("ZIEWin.autECLPS")

' Initialize the connection
Obj.SetConnectionByName("A")

' This code segment checks to see if A is started.
' The results are sent to a text box called Result.
If Obj.Started = False Then
  Result.Text = "No"
Else
  Result.Text = "Yes"
End If
```

CommStarted

これは、ホストへの接続の状況を示しています。ホストが接続されている場合、値は True です。 その他の場合は False です。CommStarted は、Boolean データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM  Hand as Long
DIM  Obj as Object

Set Obj = CreateObject("ZIEWin.autECLPS")

' Initialize the connection
Obj.SetConnectionByName("A")

' This code segment checks to see if communications are connected
' for A.  The results are sent to a text box called
' CommConn.
If Obj.CommStarted = False Then
    CommConn.Text = "No"
Else
    CommConn.Text = "Yes"
End If
```

APIEnabled

これは、エミュレーターが API 使用可能かどうかを示します。API 設定 (Z and I Emulator for Windows のウィンドウでは、**「ファイル」->「API の設定」**を選択) の状態に応じて、接続が可能な場合と、接続できない場合があります。API が使用可能の場合、値は True です。 その他の場合は False です。APIEnabled は、Boolean データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM  Hand as Long
DIM  Obj as Object

Set Obj = CreateObject("ZIEWin.autECLPS")

' Initialize the connection
Obj.SetConnectionByName("A")

' This code segment checks to see if A is API enabled.
```

```
' The results are sent to a text box called Result.
If Obj.APIEnabled = False Then
  Result.Text = "No"
Else
  Result.Text = "Yes"
End If
```

作動可能

これは、エミュレーター・ウィンドウが 開始されていて、API 使用可能で、そして接続されているかどうかを示します。これは、3 つのすべてのプロパティーを確認します。エミュレーターが準備できている場合には、値は True です。 その他の場合には、False です。Ready は、Boolean データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM  Hand as Long
DIM  Obj as Object

Set Obj = CreateObject("ZIEWin.autECLPS")

' Initialize the connection
Obj.SetConnectionByName("A")

' This code segment checks to see if A is ready.
' The results are sent to a text box called Result.
If Obj.Ready = False Then
  Result.Text = "No"
Else
  Result.Text = "Yes"
End If
```

autECLPS メソッド

以下のセクションで、autECLPS オブジェクトに有効なメソッドを 説明します。

void RegisterPSEvent() void RegisterKeyEvent() void RegisterCommEvent() void UnregisterPSEvent() void UnregisterKeyEvent() void UnregisterCommEvent() void SetConnectionByName (String Name) void SetConnectionByHandle (Long Handle) void SetCursorPos(Long Row, Long Col) void SendKeys(String text, [optional] Long row, [optional] Long col) Boolean SearchText(String text, [optional] Long Dir, [optional] Long row, [optional] Long col) String GetText([optional] Long row, [optional] Long col, [optional] Long lenToGet) void SetText(String Text, [optional] Long Row, [optional] Long Col) void CopyText([optional] Long Row, [optional] Long Col, [optional] Long LenToGet) void PasteText([optional] Long Row, [optional] Long Col, [optional] Long LenToGet) String GetTextRect(Long StartRow, Long StartCol, Long EndRow, Long EndCol) void StartCommunication() void StopCommunication() void StartMacro(String MacroName) void Wait(milliseconds as Long) Boolean WaitForCursor(Variant Row, Variant Col, [optional]Variant TimeOut, [optional] Boolean bWaitForIr) Boolean WaitWhileCursor(Variant Row, Variant Col, [optional]Variant TimeOut, [optional] Boolean bWaitForIr) Boolean WaitForString(Variant WaitString, [optional] Variant Row, [optional] Variant Col, [optional] Variant TimeOut, [optional] Boolean bWaitForIr, [optional] Boolean bCaseSens) Boolean WaitWhileString(Variant WaitString,

[optional] Variant Row, [optional] Variant Col, [optional] Variant TimeOut, [optional] Boolean bWaitForIr, [optional] Boolean bCaseSens) Boolean WaitForStringInRect(Variant WaitString, Variant sRow, Variant sCol, Variant eRow, Variant eCol, [optional] Variant nTimeOut, [optional] Boolean bWaitForIr, [optional] Boolean bCaseSens) Boolean WaitWhileStringInRect(Variant WaitString, Variant sRow, Variant sCol, Variant eRow, Variant eCol, [optional] Variant nTimeOut, [optional] Boolean bWaitForIr, [optional] Boolean bCaseSens) Boolean WaitForAttrib(Variant Row, Variant Col, Variant WaitData, [optional] Variant MaskData, [optional] Variant plane, [optional] Variant TimeOut, [optional] Boolean bWaitForIr) Boolean WaitWhileAttrib(Variant Row, Variant Col, Variant WaitData, [optional] Variant MaskData, [optional] Variant plane, [optional] Variant TimeOut, [optional] Boolean bWaitForIr) Boolean WaitForScreen(Object screenDesc, [optional] Variant TimeOut) Boolean WaitWhileScreen(Object screenDesc, [optional] Variant TimeOut) void CancelWaits()

RegisterPSEvent

このメソッドは、接続されたセッションの PS に対するすべての変更の通知を 受け取るための autECLPS オブジェクトを登録します。

プロトタイプ

void RegisterPSEvent()

パラメーター

なし

戻り値

なし

例

例については、イベント処理の例 (ページ 337)を参照してください。

RegisterKeyEvent

キー・ストローク・イベント処理を開始します。

プロトタイプ

void RegisterKeyEvent()

パラメーター

なし

戻り値

なし

例

例については、イベント処理の例 (ページ 337)を参照してください。

RegisterCommEvent

このメソッドは、すべての通信リンク接続/接続解除のイベントの通知を受け取るための オブジェクトを登録します。

プロトタイプ

void RegisterCommEvent()

パラメーター

なし

戻り値

なし

例

例については、イベント処理の例 (ページ 337)を参照してください。

UnregisterPSEvent

PS イベント処理を終了します。

プロトタイプ

void UnregisterPSEvent()

パラメーター

なし

戻り値

なし

例

例については、イベント処理の例 (ページ 337)を参照してください。

UnregisterKeyEvent

キー・ストローク・イベント処理を終了します。

プロトタイプ

void UnregisterKeyEvent()

パラメーター

なし

戻り値

なし

例

例については、イベント処理の例 (ページ 337)を参照してください。

UnregisterCommEvent

通信リンク・イベント処理を終了します。

プロトタイプ

void UnregisterCommEvent()

パラメーター

なし

戻り値

なし

SetConnectionByName

このメソッドは、接続名を使用して、 新しく作成された autECLPS オブジェクトの接続を設定します。Z and I Emulator for Windows では、この接続名は、短い ID (文字 A から Z、または a から z) です。Z and I Emulator for

Windows 接続でオープンできるのは、1 つの名前につき 1 つしかありません。例えば、一度に 1 つの接続 A のみをオープンできます。

注: autECLSession 内で autECLPS オブジェクトを使用している場合は、 これを呼び出さないでください。

プロトタイプ

void SetConnectionByName (String Name)

パラメーター

String 型、Name 型

1 文字の接続のストリング短縮名 (A から Z、または a から z)。

戻り値

なし

例

以下の例は、接続名を使用して、新しく作成された autECLPS オブジェクトの接続を 設定する方法を示します。

```
DIM  autECLPSObj as Object
DIM  NumRows as Long
Set autECLPSObj = CreateObject("ZIEWin.autECLPS")

' Initialize the connection
autECLPSObj.SetConnectionByName("A")
' For example, get the number of rows in the PS
NumRows = autECLPSObj.NumRows
```

SetConnectionByHandle

このメソッドは、接続ハンドルを使用して、 新しく作成された autECLPS オブジェクトの接続を設定します。Z and I Emulator for Windows では、この接続ハンドルは Long integer です。特定の 1 つのハンドルに対して 1 つしか、Z and I Emulator for Windows 接続をオープンできません。例えば、一度に 1 つの接続 A のみをオープンできます。

注: autECLSession 内で autECLPS オブジェクトを使用している場合は、 これを呼び出さないでください。

プロトタイプ

void SetConnectionByHandle(Long Handle)

パラメーター

Long Handle

オブジェクトに設定される接続の Long integer 値。

戻り値

なし

例

以下の例は、新しく作成された autECLPS オブジェクトの接続を、 接続ハンドルを使用して設定する方法を示します。

```
DIM  autECLPSObj as Object
DIM  autECLConnList as Object
DIM  NumRows as Long
Set autECLPSObj = CreateObject("ZIEWin.autECLPS")
Set autECLConnList = CreateObject("ZIEWin.autECLConnList")

' Initialize the connection with the first in the list
autECLConnList.Refresh
autECLPSObj.SetConnectionByHandle(autECLConnList(1).Handle)
' For example, get the number of rows in the PS
NumRows = autECLPSObj.NumRows
```

SetCursorPos

SetCursorPos メソッドは、autECLPS オブジェクト に関連する接続の表示スペースでのカーソルの位置を設定します。位置設定は、行および桁単位です。

プロトタイプ

void SetCursorPos(Long Row, Long Col)

パラメーター

Long Row

表示スペースでのカーソルの行位置。

Long Col

表示スペースでのカーソルの桁位置。

戻り値

なし

例

以下の例は、autECLPS オブジェクトに関連する接続の表示スペースでの カーソルの位置を設定する方法を示しています。

```
DIM  autECLPSObj as Object
DIM  autECLConnList as Object
Set autECLPSObj = CreateObject("ZIEWin.autECLPS")
Set autECLConnList = CreateObject("ZIEWin.autECLConnList")

' Initialize the connection with the first in the list
autECLConnList.Refresh
autECLPSObj.SetConnectionByHandle(autECLConnList(1).Handle)
autECLPSObj.SetCursorPos 2, 1
```

SendKeys

SendKeys メソッドは、キーのストリング を autECLPS オブジェクトに関連する接続の表示スペースに送信します。このメソッドによって、略号キー・ストロークを表示スペースに送信できます。これらのキー・ストロークのリストは、Sendkeys 略号キーワード (ページ 430)を参照してください。

プロトタイプ

無効SendKeys(ストリングtext, [オプション] 長い行, [オプション] 長い列)

パラメーター

String text

表示スペースに送るキー・ストリング。

Long Row

表示スペースにキーを送信する行位置。このパラメーターはオプションです。デフォルト値は、現在のカーソルの行位置です。row を指定した場合は、col も指定しなければなりません。

Long Col

表示スペースにキーを送信する桁位置。このパラメーターはオプションです。デフォルト値は、現在のカーソルの桁位置です。col を指定した場合は、row も指定しなければなりません。

戻り値

なし

例

次の例は、SendKeys メソッドを使用して、autECLPS オブジェクトに関連付けられた接続の表示スペースにキーのストリングを送信する方法を示します。

```
オブジェクトとしての Dim autECLPSObj Dim autECLConnList オブジェクトセット autECLPSObj =
 CreateObject("ZIEWin .autECLPS") Set autECLConnList = CreateObject("ZIEWin .autECLConnList") '
 接続を初期化します autECLConnList.Refresh autECLPSObj.SetConnectionByHandle(autECLConnList(1).ハンドル)
 autECLPSObj.SendKeys "HCLはとても良い会社です", 3, 1
```

SearchText

SearchText メソッドは、autECLPS オブジェクトに関連する 接続の表示スペースでのテキストの最初の出現を検索します。検索は大文字小文字を区別して行われます。テキストが見つかった場合、メソッドは True 値を戻します。テキストが見つからなかった場合は、False 値を戻します。オプションの行および桁パラメーターが使用された場合には、**row** および **col** も戻され、見つかった場合はテキストの位置を示します。

プロトタイプ

boolean SearchText(String text, [optional] Long Dir, [optional] Long Row, [optional] Long Col)

パラメーター

String text

検索するストリング。

Long Dir

検索の方向。前方検索の **1** か、 後方検索の **2** のどちらかを指定しなければなりません。このパラメーターはオプションです。デフォルト値は、前方検索の 1 です。

Long Row

表示スペース内で検索を開始する行位置。検索が成功した場合、見つかったテキストの行が戻されます。このパラメーターはオプションです。row を指定した場合は、col も指定しなければなりません。

Long Col

表示スペース内で検索を開始する桁位置。検索が成功した場合 、見つかったテキストの桁が戻されます。このパラメーターはオプションです。col を指定した場合は、row も指定しなければなりません。

戻り値

テキストが見つかった場合には True、 見つからなかった場合には False が戻されます。

例

以下の例は、autECLPS オブジェクトに関連する接続について、 表示スペース内のテキストを検索する方法を示しています。

```
Dim  autECLPSObj as Object
Dim  autECLConnList as Object
Dim  Row as Long
Dim  Col as Long
Set autECLPSObj = CreateObject("ZIEWin.autECLPS")
Set autECLConnList = CreateObject("ZIEWin.autECLConnList")
```

```
' Initialize the connection
autECLConnList.Refresh
autECLPSObj.SetConnectionByHandle(autECLConnList(1).Handle)

// Search forward in the PS from the start of the PS.  If found
// then call a hypothetical found routine, if not found, call a hypothetical

// not found routine.
row = 3
col = 1
If ( autECLPSObj.SearchText "HCL", 1, row, col) Then
  Call FoundFunc (row, col)
Else
  Call NotFoundFunc
Endif
```

GetText

GetText メソッドは、autECLPS オブジェクトに関連する 接続の表示スペースから文字を検索します。

プロトタイプ

String GetText([optional] Long Row, [optional] Long Col, [optional] Long LenToGet)

パラメーター

Long Row

表示スペースで検索を開始する行位置。このパラメーターはオプションです。

Long Col

表示スペースで検索を開始する桁位置。このパラメーターはオプションです。

Long LenToGet

表示スペースから検索する文字数。このパラメーターはオプションです。デフォルト値は、BuffLen として渡される配列の長さです。

戻り値

ストリング

PS からのテキスト。

例

以下の例は、autECLPS オブジェクトに関連する接続について、 表示スペースからストリングを検索する方法を示しています。

```
Dim  autECLPSObj as Object
Dim  PSText as String
```

```
' Initialize the connection
Set autECLPSObj = CreateObject("ZIEWin.autECLPS")
autECLPSObj.SetConnectionByName("A")

PSText = autECLPSObj.GetText(2,1,50)
```

SetText

SetText メソッドは、autECLPS オブジェクトに関連した 接続用の表示スペースにストリングを送信します。このメソッドは SendKeys メソッドに類似しているものの、 このメソッドでは略号キー・ストローク (例えば、[enter] または [pf1]) は送信しません。

プロトタイプ

void SetText(String Text, [optional] Long Row, [optional] Long Col)

パラメーター

String Text

送信する文字配列。

Long Row

表示スペースで検索を開始する行。このパラメーターはオプションです。デフォルト値は、現在のカーソルの行位置です。

Long Col

表示スペースで検索を開始する桁位置。このパラメーターはオプションです。デフォルト値は、現在のカーソルの桁位置です。

戻り値

なし

例

以下の例は、autECLPS オブジェクトに関連する接続について、 表示スペース内のテキストを検索する方法を示しています。

```
Dim autECLPSObj as Object

'Initialize the connection
Set autECLPSObj = CreateObject("ZIEWin.autECLPS")
autECLPSObj.SetConnectionByName("A")
autECLPSObj.SetText"HCL is great", 2, 1
```

CopyText

このメソッドは、指定された長さの表示スペース内の所定の場所からクリップボードにテキストをコピーします。コピーされるテキストの長さは指定された長さになります。 長さを指定しない場合、表示スペースの終わりまでのテキストがコピーされます。場所を指定しない場合、テキストは表示スペース内の現行カーソル位置からコピーされます。パラメーターを指定しないと、表示スペース全体がクリップボードにコピーされます。

プロトタイプ

void CopyText([optional] Long Row, [optional] Long Col, [optional] Long LenToGet)

パラメーター

Long LenToGet

表示スペースからコピーする文字数。このパラメーターはオプションです。デフォルト値は、BuffLen として渡される配列の長さです。

Long Row

表示スペースでコピーを開始する行。このパラメーターはオプションです。デフォルト値は、現在のカーソルの行位置です。

Long Col

表示スペースでコピーを開始する桁位置。このパラメーターはオプションです。デフォルト値は、現在のカーソルの桁です。

戻り値

なし

例

以下の例は、autECLPS オブジェクトに関連する接続について、 表示スペース内のテキストをクリップボードにコピーする方法を示しています。

```
Dim autECLPSObj as Object
'Initialize the connection

Set autECLPSObj = CreateObject("ZIEWin.autECLPS")

autECLPSObj.SetConnectionByName("A")
autECLPSObj.CopyText 6, 53, 10
```

PasteText

このメソッドは、指定された長さのテキストをクリップボードから表示スペースの指定の場所に貼り付けます。貼り付けられるテキストの長さは、指定された長さになります。長さを指定しない場合、クリップボードのテキスト全体が、表示スペースの最後に達するまで貼り付けられます。ロケーションが指定されていない場合、テキストは、表示

スペースの現在のカーソル位置に貼り付けられます。表示スペースがフィールド形式であり、クリップボードの内容の貼り付け中にタブ文字 '\t' が指定されている場合、残りの貼り付け内容は次の書き込み可能フィールドに移動します。

プロトタイプ

void PasteText([optional] Long Row, [optional] Long Col, [optional] Long LenToGet)

パラメーター

Long LenToGet

クリップボードから表示スペースに貼り付ける文字数。このパラメーターはオプションです。デフォルトはクリップボード内のテキストの長さです。

Long Row

クリップボードから表示スペースへの貼り付けを開始する行。このパラメーターはオプションです。デフォルト値は、現在のカーソルの行位置です。

Long Col

クリップボードから表示スペースへの貼り付けを開始する桁位置。このパラメーターはオプションです。デフォルト値は、現在のカーソルの桁位置です。

戻り値

なし

例

以下の例は、autECLPS オブジェクトに関連する接続について、クリップボードから表示スペースにテキストを貼り付ける方法を示しています。

```
Dim autECLPSObj as Object
'Initialize the connection

Set autECLPSObj = CreateObject("ZIEWin.autECLPS")

autECLPSObj.SetConnectionByName("A")
autECLPSObj.PasteText 8, 53, 10
```

GetTextRect

GetTextRect メソッドは、autECLPS オブジェクトに関連する 接続の表示スペースの長方形域から文字を検索します。テキスト検索では折り返しは実行されません。長方形域のみ検索されます。

プロトタイプ

String GetTextRect(Long StartRow, Long StartCol, Long EndRow, Long EndCol)

パラメーター

Long StartRow

表示スペースで検索を開始する行。

Long StartCol

表示スペースで検索を開始する桁。

Long EndRow

表示スペースで検索を終了する行。

Long EndCol

表示スペースで検索を終了する桁。

戻り値

ストリング

PS テキスト

例

以下の例は、autECLPS オブジェクトに関連する接続について、 表示スペース内の長方形域から文字を検索する方法を示しています。

```
Dim  autECLPSObj as Object
Dim  PSText String

' Initialize the connection
Set autECLPSObj = CreateObject ("ZIEWin.autELCPS")
autECLPSObj.SetConnectionByName("A")

PSText = GetTextRect(1,1,2,80)
```

SetTextRect

SetTextRect メソッドにより、表示スペース内の長方形の領域に文字を設定し、autECLPS オブジェクトに関連させて 接続します。テキストを設定する際、折り返しは行われません。つまり、長方形の領域だけが設定されます。

プロトタイプ

SetTextRect(String Text, Long StartRow, Long StartCol, Long EndRow, Long EndCol)

パラメーター

String Text

表示スペースに設定する文字配列。

Long StartRow

表示スペースで設定を開始する行。

Long StartCol

表示スペースで設定を開始する桁。

Long EndRow

表示スペースで設定を終了する行。

Long EndCol

表示スペースで設定を終了する桁。

戻り値

なし

例

次の例は、表示スペース内の長方形の領域に文字を設定し、autECLPS オブジェクトに関連させて 接続する方法を示しています。

```
Dim autECLPSObj as Object
Dim PSText String

’ Initialize the connection
Set autECLPSObj = CreateObject ("ZIEWin.autELCPS")
autECLPSObj.SetConnectionByName("A")

SetTextRect "HCL is great company to collaborate with", 1, 1, 4, 8
```

括弧を使用する場合は、以下のように SetTextRect を使用します。

```
call SetTextRect("HCL is great company to collaborate with", 1, 1, 4, 8)
```

StartCommunication

StartCommunication 集合要素メソッドは、ホスト・データ・ストリームに ZIEWin エミュレーターを接続します。これは、ZIEWin エミュレーター**「通信」**メニューを表示して**「接続」**を選択した場合と同じ結果になります。

プロトタイプ

void StartCommunication()

パラメーター

なし

戻り値

なし

例

以下の例は、ZIEWin エミュレーター・セッションをホストへ接続する方法を示しています。

```
Dim PSObj as Object
Dim autECLConnList as Object

Set autECLConnList = CreateObject("ZIEWin.autECLConnList")
Set PSObj = CreateObject("ZIEWin.autECLPS")

' Initialize the session
autECLConnList.Refresh
PSObj.SetConnectionByHandle(autECLConnList(1).Handle)

PSObj.StartCommunication()
```

StopCommunication

StopCommunication 集合要素メソッドは、ホスト・データ・ストリームから ZIEWin エミュレーターを切断します。これは、ZIEWin エミュレーター**「通信」**メニューを表示して**「切断」**を選択した場合と同じ結果になります。

プロトタイプ

void StopCommunication()

パラメーター

なし

戻り値

なし

例

以下の例は、ZIEWin エミュレーター・セッションをホストへ接続する方法を示しています。

```
Dim PSObj as Object
Dim autECLConnList as Object

Set autECLConnList = CreateObject("ZIEWin.autECLConnList")
Set PSObj = CreateObject("ZIEWin.autECLPS")

' Initialize the session
autECLConnList.Refresh
PSObj.SetConnectionByHandle(autECLConnList(1).Handle)
```

```
PSObj.StopCommunication()
```

StartMacro

StartMacro メソッドは、MacroName パラメーターにより指示された Z and I Emulator for Windows のマクロ・ファイルを実行します。

プロトタイプ

void StartMacro(String MacroName)

パラメーター

String MacroName

Z and I Emulator for Windows のユーザー・クラス・アプリケーション・データ・ディレクトリー (インストール時に指定) に入っているマクロ・ファイルの名前でファイル拡張子を持っていない。このメソッドは、 長いファイル名をサポートしません。

戻り値

なし

使用上の注意

マクロ名には、短いファイル名を使用する必要があります。このメソッドは、 長いファイル名をサポートしません。

例

以下の例は、マクロを開始する方法を示しています。

```
Dim PS as Object

Set PS = CreateObject("ZIEWin.autECLPS")
PS.StartMacro "mymacro"
```

待機

Wait メソッドは、milliseconds パラメーターで指定されたミリ秒間 待機します。

プロトタイプ

void Wait(milliseconds as Long)

パラメーター

Long milliseconds

待機するミリ秒数。

戻り値

なし

例

```
Dim autECLPSObj as Object

Set autECLPSObj = CreateObject ("ZIEWin.autECLPS")
autECLPSObj.SetConnectionByName ("A")

' Wait for 10 seconds
autECLPSObj.Wait(10000)
```

WaitForCursor

WaitForCursor メソッドは、autECLPS オブジェクトに関連した 接続の表示スペースでカーソルが指定された位置に配置されるのを待ちます。

プロトタイプ

Boolean WaitForCursor(Variant Row, Variant Col, [optional]Variant TimeOut, [optional] Boolean bWaitForIr)

パラメーター

Variant Row

カーソルの行の位置。

Variant Col

カーソルの列の位置。

Variant TimeOut

待機する時間の最大長 (ミリ秒)。このパラメーターはオプションです。デフォルトは Infinite (無期限) です。

Boolean bWaitForIr

この値が True である場合、待ち条件の基準を満たすと、関数は OIA の入力受け入れ準備 ができるまで待機します。このパラメーターはオプションです。デフォルトは False です。

戻り値

メソッドは、条件が合致していれば True を戻し、 タイムアウト値が超えていれば False を戻します。

例

```
Dim autECLPSObj as Object
Dim Row, Col

Set autECLPSObj = CreateObject("ZIEWin.autECLPS")
autECLPSObj.SetConnectionByName("A")

Row = 20
Col = 16

if (autECLPSObj.WaitForCursor(Row,Col,10000)) then
    msgbox "Cursor is at " " Row " "," " Col
else
    msgbox "Timeout Occurred"
end if
```

WaitWhileCursor

WaitWhileCursor メソッドは、autECLPS オブジェクトに関連した接続の 表示スペースでカーソルが指定された位置に配置されている間待機します。

プロトタイプ

Boolean WaitWhileCursor(Variant Row, Variant Col, [optional]Variant TimeOut, [optional] Boolean bWaitForIr)

パラメーター

Variant Row

カーソルの行の位置。

Variant Col

カーソルの列の位置。

Variant TimeOut

待機する時間の最大長 (ミリ秒)。このパラメーターはオプションです。デフォルトは Infinite (無期限) です。

Boolean bWaitForIr

この値が True である場合、待ち条件の基準を満たすと、関数は OIA の入力受け入れ準備 ができるまで待機します。このパラメーターはオプションです。デフォルトは False です。

戻り値

メソッドは、条件が合致していれば True を戻し、 タイムアウト値が超えていれば False を戻します。

例

```
Dim autECLPSObj as Object
Dim Row, Col

Set autECLPSObj = CreateObject("ZIEWin.autECLPS")
autECLPSObj.SetConnectionByName("A")

Row = 20
Col = 16

if (autECLPSObj.WaitWhileCursor(Row,Col,10000)) then
     msgbox "Cursor is no longer at " " Row " "," " Col
else
     msgbox "Timeout Occurred"
end if
```

WaitForString

WaitForString メソッドは、autECLPS オブジェクトに関連した接続の 表示スペースで、指定されたストリングが現れるのを待ちます。オプションの行パラメーターおよび桁パラメーターが 使用される場合は、ストリングは指定された位置から開始しなければなりません。行と桁に、それぞれ 0 が渡された場合は、メソッドは PS 全体を探索します。

プロトタイプ

```
Boolean WaitForString(Variant WaitString, [optional] Variant Row, [optional] Variant Col, [optional] Variant TimeOut, [optional] Boolean bWaitForIr, [optional] Boolean bCaseSens)
```

パラメーター

Variant WaitString

待機の対象となるストリング。

Variant Row

ストリングが開始する行位置。このパラメーターはオプションです。デフォルトは 0 です。

Variant Col

ストリングが開始する桁位置。このパラメーターはオプションです。デフォルトは 0 です。

Variant TimeOut

待機する時間の最大長 (ミリ秒)。このパラメーターはオプションです。デフォルトは Infinite (無期限) です。

Boolean bWaitForIr

この値が True である場合、待ち条件の基準を満たすと、関数は OIA の入力受け入れ準備 ができるまで待機します。このパラメーターはオプションです。デフォルトは False です。

Boolean bCaseSens

この値が True の場合は、待ち条件は大/小文字の区別があるものとして検査されます。このパラメーターはオプションです。デフォルトは False です。

戻り値

メソッドは、条件が合致していれば True を戻し、 タイムアウト値が超えていれば False を戻します。

例

```
Dim autECLPSObj as Object
Dim Row, Col, WaitString

Set autECLPSObj = CreateObject("ZIEWin.autECLPS")
autECLPSObj.SetConnectionByName("A")

WaitString = "Enter USERID"
Row = 20
Col = 16

if (autECLPSObj.WaitForString(WaitString,Row,Col,10000))  then
    msgbox WaitString " " found at " " Row " "," " Col
else
    msgbox "Timeout Occurred"
end if
```

WaitWhileString

WaitWhileString メソッドは、指定されたストリングが autECLPS オブジェクトに関連した接続の 表示スペースに現れている間待機します。オプションの行パラメーターおよび桁パラメーターが 使用される場合は、ストリングは指定された位置から開始しなければなりません。行と桁に、それぞれ 0 が渡された場合は、メソッドは PS 全体を探索します。

プロトタイプ

```
Boolean WaitWhileString(Variant WaitString, [optional] Variant Row, [optional] Variant Col, [optional] Variant TimeOut, [optional] Boolean bWaitForIr, [optional] Boolean bCaseSens)
```

パラメーター

Variant WaitString

このストリング値が存在している間、メソッドは待機します。

Variant Row

ストリングが開始する行位置。このパラメーターはオプションです。デフォルトは 0 です。

Variant Col

ストリングが開始する桁位置。このパラメーターはオプションです。デフォルトは 0 です。

Variant TimeOut

待機する時間の最大長 (ミリ秒)。このパラメーターはオプションです。デフォルトは Infinite (無期限) です。

Boolean bWaitForIr

この値が True である場合、待ち条件の基準を満たすと、関数は OIA の入力受け入れ準備 ができるまで待機します。このパラメーターはオプションです。デフォルトは False です。

Boolean bCaseSens

この値が True の場合は、待ち条件は大/小文字の区別があるものとして検査されます。このパラメーターはオプションです。デフォルトは False です。

戻り値

メソッドは、条件が合致していれば True を戻し、 タイムアウト値が超えていれば False を戻します。

例

```
Dim autECLPSObj as Object
Dim Row, Col, WaitString

Set autECLPSObj = CreateObject("ZIEWin.autECLPS")
autECLPSObj.SetConnectionByName("A")

WaitString = "Enter USERID"
Row = 20
Col = 16

if (autECLPSObj.WaitWhileString(WaitString,Row,Col,10000))  then
    msgbox WaitString " " was found at " " Row " "," " Col
else
    msgbox "Timeout Occurred"
end if
```

WaitForStringInRect

WaitForStringInRect メソッドは、指定された長方形内の autECLPS オブジェクトに関連した 接続の表示スペースで、指定されたストリングが現れるのを待ちます。

プロトタイプ

Boolean WaitForStringInRect(Variant WaitString, Variant sRow, Variant sCol, Variant eRow, Variant eCol, [optional] Variant nTimeOut, [optional] Boolean bWaitForIr, [optional] Boolean bCaseSens)

パラメーター

Variant WaitString

待機の対象となるストリング。

Variant sRow

長方形探索を開始する行位置。

Variant sCol

長方形探索を開始する桁位置。

Variant eRow

長方形探索を終了する行位置。

Variant eCol

長方形探索を終了する桁位置。

Variant nTimeOut

待機する時間の最大長 (ミリ秒)。このパラメーターはオプションです。デフォルトは Infinite (無期限) です。

Boolean bWaitForIr

この値が True である場合、待ち条件の基準を満たすと、関数は OIA の入力受け入れ準備 ができるまで待機します。このパラメーターはオプションです。デフォルトは False です。

Boolean bCaseSens

この値が True の場合は、待ち条件は大/小文字の区別があるものとして検査されます。このパラメーターはオプションです。デフォルトは False です。

戻り値

メソッドは、条件が合致していれば True を戻し、 タイムアウト値が超えていれば False を戻します。

例

```
Dim autECLPSObj as Object
Dim sRow, sCol, eRow, eCol, WaitString

Set autECLPSObj = CreateObject("ZIEWin.autECLPS")
autECLPSObj.SetConnectionByName("A")

WaitString = "Enter USERID"
sRow = 20
```

```
sCol = 16
eRow = 21
eCol = 31

if (autECLPSObj.WaitForStringInRect(WaitString,sRow,sCol,eRow,eCol,10000))  then
    msgbox WaitString " " found in rectangle"
else
    msgbox "Timeout Occurred"
end if
```

WaitWhileStringInRect

WaitWhileStringInRect メソッドは、指定されたストリングが指定長方形内の autECLPS オブジェクト に関連した接続の表示スペースに現れている間待機します。

プロトタイプ

Boolean WaitWhileStringInRect(Variant WaitString, Variant sRow, Variant sCol, Variant eRow, Variant eCol, [optional] Variant nTimeOut, [optional] Boolean bWaitForIr, [optional] Boolean bCaseSens)

パラメーター

Variant WaitString

このストリング値が存在している間、メソッドは待機します。

Variant sRow

長方形探索を開始する行位置。

Variant sCol

長方形探索を開始する桁位置。

Variant eRow

長方形探索を終了する行位置。

Variant eCol

長方形探索を終了する桁位置。

Variant nTimeOut

待機する時間の最大長 (ミリ秒)。このパラメーターはオプションです。デフォルトは Infinite (無期限) です。

Boolean bWaitForIr

この値が True である場合、待ち条件の基準を満たすと、関数は OIA の入力受け入れ準備 ができるまで待機します。このパラメーターはオプションです。デフォルトは False です。

Boolean bCaseSens

この値が True の場合は、待ち条件は大/小文字の区別があるものとして検査されます。このパラメーターはオプションです。デフォルトは False です。

戻り値

メソッドは、条件が合致していれば True を戻し、 タイムアウト値が超えていれば False を戻します。

例

```
Dim autECLPSObj as Object
Dim sRow, sCol, eRow, eCol, WaitString

Set autECLPSObj = CreateObject("ZIEWin.autECLPS")
autECLPSObj.SetConnectionByName("A")

WaitString = "Enter USERID"
sRow = 20
sCol = 16
eRow = 21
eCol = 31

if (autECLPSObj.WaitWhileStringInRect(WaitString,sRow,sCol,eRow,eCol,10000))  then
    msgbox WaitString " " no longer in rectangle"
else
    msgbox "Timeout Occurred"
end if
```

WaitForAttrib

WaitForAttrib メソッドは、指定行/列位置にある autECLPS オブジェクトに関連した接続 の表示スペースで、指定された属性値が現れるまで待ちます。オプションの MaskData パラメーターを使用して、どの属性値を探索するのかを制御することができます。オプションのプレーン・パラメーターに より、4 つの PS プレーンの内から任意のプレーンを選択することが可能となります。

プロトタイプ

```
Boolean WaitForAttrib(Variant Row, Variant Col, Variant WaitData, [optional] Variant MaskData, [optional] Variant plane, [optional] Variant TimeOut, [optional] Boolean bWaitForIr)
```

パラメーター

Variant Row

属性の行の位置。

Variant Col

属性の列の位置。

Variant WaitData

待機する属性値。この値は 1 バイトの 16 進数値です。

Variant MaskData

属性をマスクするのに使用する 1 バイトの 16 進数の値。このパラメーターはオプションです。デフォルト値は 0xFF です。

Variant plane

取得する属性のプレーン。プレーンは、以下のような値が可能です。

1

テキスト・プレーン

2

カラー・プレーン

3

フィールド・プレーン (デフォルト)

4

拡張フィールド・プレーン

Variant TimeOut

待機する時間の最大長 (ミリ秒)。このパラメーターはオプションです。デフォルトは Infinite (無期限) です。

Boolean bWaitForIr

この値が True である場合、待ち条件の基準を満たすと、関数は OIA の入力受け入れ準備 ができるまで待機します。このパラメーターはオプションです。デフォルトは False です。

戻り値

メソッドは、条件が合致していれば True を戻し、 タイムアウト値が超えていれば False を戻します。

例

```
Dim autECLPSObj as Object
Dim Row, Col, WaitData, MaskData, plane

Set autECLPSObj = CreateObject("ZIEWin.autECLPS")
autECLPSObj.SetConnectionByName("A")

Row = 20
Col = 16
WaitData = E8h
MaskData = FFh
plane = 3

if (autECLPSObj.WaitForAttrib(Row, Col, WaitData, MaskData, plane, 10000))  then
    msgbox "Attribute " " WaitData " " found"
```

```
else
    msgbox "Timeout Occurred"
end if
```

WaitWhileAttrib

WaitWhileAttrib メソッドは、指定行/列位置にある autECLPS オブジェクトに関連した接続 の表示スペースに、指定された属性値が表示されている間待ちます。オプションの MaskData パラメーターを使用して、どの属性値を探索するのか を制御することができます。オプションのプレーン・パラメーターに より、4 つの PS プレーンの内から任意のプレーンを選択することが可能となります。

プロトタイプ

Boolean WaitWhileAttrib(Variant Row, Variant Col, Variant WaitData, [optional] Variant MaskData, [optional] Variant plane, [optional] Variant TimeOut, [optional] Boolean bWaitForIr)

パラメーター

Variant Row

属性の行の位置。

Variant Col

属性の列の位置。

Variant WaitData

待機する属性値。この値は 1 バイトの 16 進数値です。

Variant MaskData

属性をマスクするのに使用する 1 バイトの 16 進数の値。このパラメーターはオプションです。デフォルト値は 0xFF です。

Variant plane

取得する属性のプレーン。プレーンは、以下のような値が可能です。

1

テキスト・プレーン

2

カラー・プレーン

3

フィールド・プレーン (デフォルト)

4

拡張フィールド・プレーン

Variant TimeOut

待機する時間の最大長 (ミリ秒)。このパラメーターはオプションです。デフォルトは Infinite (無期限) です。

Boolean bWaitForIr

この値が True である場合、待ち条件の基準を満たすと、関数は OIA の入力受け入れ準備 ができるまで待機します。このパラメーターはオプションです。デフォルトは False です。

戻り値

メソッドは、条件が合致していれば True を戻し、 タイムアウト値が超えていれば False を戻します。

例

```
Dim autECLPSObj as Object
Dim Row, Col, WaitData, MaskData, plane

Set autECLPSObj = CreateObject("ZIEWin.autECLPS")
autECLPSObj.SetConnectionByName("A")

Row = 20
Col = 16
WaitData = E8h
MaskData = FFh
plane = 3

if (autECLPSObj.WaitWhileAttrib(Row, Col, WaitData, MaskData, plane, 10000))  then
         msgbox "Attribute " " WaitData " " No longer exists"
else
         msgbox "Timeout Occurred"
end if
```

WaitForScreen

autECLScreenDesc パラメーターにより記述された画面が表示スペースに 現れるのを同期して待ちます。

注: OIA 入力フラグの待機は autECLScreenDesc オブジェクト 上に設定され、パラメーターとして待機メソッドに渡されません。

プロトタイプ

Boolean WaitForScreen(Object screenDesc, [optional] Variant TimeOut)

パラメーター

Object screenDesc

画面を記述する autECLScreenDesc オブジェクト (autECLScreenDesc クラス (ページ 339)を参照)。

Variant TimeOut

待機する時間の最大長 (ミリ秒)。このパラメーターはオプションです。デフォルトは Infinite (無期限) です。

戻り値

メソッドは、条件が合致していれば True を戻し、 タイムアウト値が超えていれば False を戻します。

例

```
Dim autECLPSObj as Object
Dim autECLScreenDescObj as Object

Set autECLScreenDescObj = CreateObject("ZIEWin.autECLScreenDesc")
Set autECLPSObj = CreateObject("ZIEWin.autECLPS")
autECLPSObj.SetConnectionByName("A")

autECLScreenDesObj.AddCursorPos 23, 1

if (autECLPSObj.WaitForScreen(autECLScreenDesObj, 10000))  then
    msgbox "Screen found"
else
    msgbox "Timeout Occurred"
end if
```

WaitWhileScreen

autECLScreenDesc パラメーターにより記述された画面が表示スペースから なくなるまで同期して待ちます。

注: OIA 入力フラグの待機は autECLScreenDesc オブジェクト 上に設定され、パラメーターとして待機メソッドに渡されません。

プロトタイプ

Boolean WaitWhileScreen(Object screenDesc, [optional] Variant TimeOut)

パラメーター

Object ScreenDesc

画面を記述する autECLScreenDesc オブジェクト (autECLScreenDesc クラス (ページ 339)を参照)。

Variant TimeOut

待機する時間の最大長 (ミリ秒)。このパラメーターはオプションです。デフォルトは Infinite (無期限) です。

戻り値

メソッドは、条件が合致していれば True を戻し、 タイムアウト値が超えていれば False を戻します。

例

```
Dim autECLPSObj as Object
Dim autECLScreenDescObj as Object

Set autECLScreenDescObj = CreateObject("ZIEWin.autECLScreenDesc")
Set autECLPSObj = CreateObject("ZIEWin.autECLPS")
autECLPSObj.SetConnectionByName("A")

autECLScreenDesObj.AddCursorPos 23, 1

if (autECLPSObj.WaitWhileScreen(autECLScreenDesObj, 10000))  then
    msgbox "Screen exited"
else
    msgbox "Timeout Occurred"
end if
```

CancelWaits

現在活動待ちのメソッドを取り消します。

プロトタイプ

void CancelWaits()

パラメーター

なし

戻り値

なし

autECLPS イベント

以下のイベントは autECLPS に有効です。

void NotifyPSEvent() void NotifyKeyEvent(string KeyType, string KeyString, PassItOn as Boolean) void NotifyCommEvent(boolean bConnected) void NotifyPSError() void NotifyKeyError() void NotifyCommError() void NotifyPSStop(Long Reason) void NotifyKeyStop(Long Reason) void NotifyCommStop(Long Reason)

NotifyPSEvent

指定された PS は更新されました。

プロトタイプ

void NotifyPSEvent()

パラメーター

なし

例

例については、イベント処理の例 (ページ 337)を参照してください。

NotifyKeyEvent

キー・ストローク・イベントが起こり、キー情報が提供されました。この関数は、指定された PS に対するキー・ストロークをインターセプトするために 使用することができます。キー情報はイベント・ハンドラーに渡され、 続いて次へ渡すことも、別のアクションを実行することも可能です。

注: 1 つのオブジェクトだけが、同時点で指定された PS に登録済みの キー・ストローク・イベント処理を持つことができます。

プロトタイプ

void NotifyKeyEvent(string KeyType, string KeyString, PassItOn as Boolean)

パラメーター

String KeyType

インターセプトされたキーのタイプ。

月

略号キー・ストローク

A

ASCII

String KeyString

インターセプトされたキー・ストローク。

Boolean PassItOn

キー・ストロークのエコーを PS に返すかどうかを示すためにフラグを付けます。

TRUE

キー・ストロークを PS に渡すようにします。

FALSE

キー・ストロークを PS に渡さないようにします。

例

例については、イベント処理の例 (ページ 337)を参照してください。

NotifyCommEvent

指定された通信リンクは、接続あるいは接続解除されています。

プロトタイプ

void NotifyCommEvent(boolean bConnected)

パラメーター

boolean bConnected

通信リンクが現在接続されている場合は True で、これ以外の場合は False。

例

例については、イベント処理の例 (ページ 337)を参照してください。

NotifyPSError

このイベントは、イベント処理でエラーが発生したときに起こります。

プロトタイプ

void NotifyPSError()

パラメーター

なし

例

例については、イベント処理の例 (ページ 337)を参照してください。

NotifyKeyError

このイベントは、イベント処理でエラーが発生したときに起こります。

プロトタイプ

void NotifyKeyError()

パラメーター

なし

例

例については、イベント処理の例 (ページ 337)を参照してください。

NotifyCommError

このイベントは、イベント処理でエラーが発生したときに起こります。

プロトタイプ

void NotifyCommError()

パラメーター

なし

例

例については、イベント処理の例 (ページ 337)を参照してください。

NotifyPSStop

このイベントは、イベント処理が停止したときに起こります。

プロトタイプ

void NotifyPSStop(Long Reason)

パラメーター

Long Reason

停止の理由コード。現在は、これは常に 0 です。

例

例については、イベント処理の例 (ページ 337)を参照してください。

NotifyKeyStop

このイベントは、イベント処理が停止したときに起こります。

プロトタイプ

void NotifyKeyStop(Long Reason)

パラメーター

Long Reason

停止の理由コード。現在は、これは常に 0 です。

例

例については、イベント処理の例 (ページ 337)を参照してください。

NotifyCommStop

このイベントは、イベント処理が停止したときに起こります。

プロトタイプ

void NotifyCommStop(Long Reason)

パラメーター

Long Reason

停止の理由コード。現在は、これは常に 0 です。

イベント処理の例

以下は、PS イベントの実装方法についての簡単な例です。

```
Option Explicit
Private WithEvents mPS As autECLPS 'AutPS added as reference
Private WithEvents Mkey as autECLPS

sub main()
'Create Objects
Set mPS = New autECLPS
```

```
Set mkey = New autECLPS
mPS.SetConnectionByName "A" 'Monitor Session A for PS Updates
mPS.SetConnectionByName "B" 'Intercept Keystrokes intended for Session B

mPS.RegisterPSEvent 'register for PS Updates
mPS.RegisterCommEvent ' register for Communications Link updates for session A
mkey.RegisterKeyEvent 'register for Key stroke intercept

' Display your form or whatever here  (this should be a blocking call, otherwise sub just ends
call DisplayGUI()

mPS.UnregisterPSEvent
mPS.UnregisterCommEvent
mkey.UnregisterKeyEvent
```

```
set mPS = Nothing
set mKey = Nothing
End Sub

'This sub will get called when the PS of the Session registered
'above changes
Private Sub mPS_NotifyPSEvent()
' do your processing here
End Sub

'This sub will get called when Keystrokes are entered into Session B
Private Sub mkey_NotifyKeyEvent(string KeyType, string KeyString, PassItOn as Boolean)
' do your keystroke filtering here
If (KeyType = "M") Then
'handle mnemonics here
if (KeyString = "[PF1]" then 'intercept PF1 and send PF2 instead
mkey.SendKeys "[PF2]"
set PassItOn = false
end if
end if
```

```
End Sub

'This event occurs if an error happens in PS event processing
Private Sub mPS_NotifyPSError()
'Do any error processing here
End Sub

'This event occurs when PS Event handling ends
Private Sub mPS_NotifyPSStop(Reason As Long)
'Do any stop processing here
End Sub

'This event occurs if an error happens in Keystroke processing
Private Sub mkey_NotifyKeyError()
'Do any error processing here
End Sub

'This event occurs when key stroke event handling ends
Private Sub mkey_NotifyKeyStop(Reason As Long)
'Do any stop processing here
End Sub
```

```
'This sub will get called when the Communication Link Status of the registered
'connection changes
Private Sub mPS_NotifyCommEvent()
' do your processing here
End Sub
```

```
'This event occurs if an error happens in Communications Link event processing
Private Sub mPS_NotifyCommError()
'Do any error processing here
End Sub

'This event occurs when Communications Status Notification ends
Private Sub mPS_NotifyCommStop()
'Do any stop processing here
End Sub
```

autECLScreenDesc クラス

autECLScreenDesc は、HCL ホスト・アクセス・クラス・ライブラリーの画面認識テクノロジーの画面を記述するために使用されるクラスです。これは、カーソル位置はもちろんのこと、 これを説明する表示スペースの 4 つの主なプレーン (テキスト、フィールド、 拡張フィールド、およびカラー・プレーン) すべてを使用します。

このオブジェクトで用意されているメソッドを使用して、プログラマーは指定された画面が ホスト・サイド・アプリケーションでどのように表示されるかを詳細に記述することが できます。autECLScreenDesc オブジェクトが作成されセットされると、これを autECLPS で提供される 同期 WaitFor... メソッドのいずれかに渡すか、または autECLScreenReco に渡すことが でき、autECLScreenDesc オブジェクトと一致する画面が PS 内に現れた場合は、非同期イベントを破棄します。

autECLScreenDesc メソッド

以下のセクションで、autECLScreenDesc に有効なメソッドを説明します。

void AddAttrib(Variant attrib, Variant row, Variant col, Variant plane) void AddCursorPos(Variant row, Variant col) void AddNumFields(Variant num) void AddNumInputFields(Variant num) void AddOIAInhibitStatus(Variant type) void AddString(String str, Variant row, Variant col, [optional] Boolean caseSense) void AddStringInRect(String str, Variant sRow, Variant sCol, Variant eRow, Variant eCol, [optional] Variant caseSense) void Clear()

AddAttrib

画面記述の指定位置に属性値を追加します。

プロトタイプ

void AddAttrib(Variant attrib, Variant row, Variant col, Variant plane)

パラメーター

Variant attrib

1 バイトの 16 進数属性の値。

Variant row

行位置。

Variant col

桁位置。

Variant plane

取得する属性のプレーン。プレーンは、以下のような値が可能です。

0. すべてのプレーン
1. テキスト・プレーン
2. カラー・プレーン
3. フィールド・プレーン
4. 拡張フィールド・プレーン

戻り値

なし

例

```
Dim autECLPSObj as Object
Dim autECLScreenDescObj as Object

Set autECLScreenDescObj = CreateObject("ZIEWin.autECLScreenDesc")
Set autECLPSObj = CreateObject("ZIEWin.autECLPS")
autECLPSObj.SetConnectionByName "A"

autECLScreenDesObj.AddCursorPos 23, 1
autECLScreenDesObj.AddAttrib E8h, 1, 1, 2
autECLScreenDesObj.AddCursorPos 23,1
autECLScreenDesObj.AddNumFields 45
autECLScreenDesObj.AddNumInputFields 17
autECLScreenDesObj.AddOIAInhibitStatus 1
autECLScreenDesObj.AddString "LOGON", 23, 11, True
autECLScreenDesObj.AddStringInRect "PASSWORD", 23, 1, 24, 80, False

if (autECLPSObj.WaitForScreen(autECLScreenDesObj, 10000)) then
     msgbox "Screen reached"
 else
     msgbox "Timeout Occurred"
end if
```

AddCursorPos

指定位置に画面記述のためのカーソル位置をセットします。

プロトタイプ

void AddCursorPos(Variant row, Variant col)

パラメーター

Variant row

行位置。

Variant col

桁位置。

戻り値

なし

例

```
Dim autECLPSObj as Object
Dim autECLScreenDescObj as Object

Set autECLScreenDescObj = CreateObject("ZIEWin.autECLScreenDesc")
Set autECLPSObj = CreateObject("ZIEWin.autECLPS")
autECLPSObj.SetConnectionByName "A"

autECLScreenDesObj.AddCursorPos 23, 1
autECLScreenDesObj.AddAttrib E8h, 1, 1, 2
autECLScreenDesObj.AddCursorPos 23,1
autECLScreenDesObj.AddNumFields 45
autECLScreenDesObj.AddNumInputFields 17
autECLScreenDesObj.AddOIAInhibitStatus 1
autECLScreenDesObj.AddString "LOGON", 23, 11, True
autECLScreenDesObj.AddStringInRect "PASSWORD", 23, 1, 24, 80, False

if (autECLPSObj.WaitForScreen(autECLScreenDesObj, 10000)) then
     msgbox "Screen reached"
else
     msgbox "Timeout Occurred"
end if
```

AddNumFields

画面記述にフィールド数を追加します。

プロトタイプ

void AddNumFields(Variant num)

パラメーター

Variant num

フィールドの数。

戻り値

なし

例

```
Dim autECLPSObj as Object
Dim autECLScreenDescObj as Object

Set autECLScreenDescObj = CreateObject("ZIEWin.autECLScreenDesc")
Set autECLPSObj = CreateObject("ZIEWin.autECLPS")
autECLPSObj.SetConnectionByName "A"

autECLScreenDesObj.AddCursorPos 23, 1
autECLScreenDesObj.AddAttrib E8h, 1, 1, 2
autECLScreenDesObj.AddCursorPos 23,1
autECLScreenDesObj.AddNumFields 45
autECLScreenDesObj.AddNumInputFields 17
autECLScreenDesObj.AddOIAInhibitStatus 1
autECLScreenDesObj.AddString "LOGON", 23, 11, True
autECLScreenDesObj.AddStringInRect "PASSWORD", 23, 1, 24, 80, False

if (autECLPSObj.WaitForScreen(autECLScreenDesObj, 10000)) then
    msgbox "Screen reached"
else
    msgbox "Timeout Occurred"
end if
```

AddNumInputFields

画面記述にフィールド数を追加します。

プロトタイプ

void AddNumInputFields(Variant num)

パラメーター

Variant num

入力フィールドの数。

戻り値

なし

例

```
Dim autECLPSObj as Object
Dim autECLScreenDescObj as Object

Set autECLScreenDescObj = CreateObject("ZIEWin.autECLScreenDesc")
Set autECLPSObj = CreateObject("ZIEWin.autECLPS")
autECLPSObj.SetConnectionByName "A"

autECLScreenDesObj.AddCursorPos 23, 1
autECLScreenDesObj.AddAttrib E8h, 1, 1, 2
autECLScreenDesObj.AddCursorPos 23,1
autECLScreenDesObj.AddNumFields 45
autECLScreenDesObj.AddNumInputFields 17
autECLScreenDesObj.AddOIAInhibitStatus 1
autECLScreenDesObj.AddString "LOGON", 23, 11, True
autECLScreenDesObj.AddStringInRect "PASSWORD", 23, 1, 24, 80, False

if (autECLPSObj.WaitForScreen(autECLScreenDesObj, 10000)) then
msgbox "Screen reached"
else
    msgbox "Timeout Occurred"
end if
```

AddOIAInhibitStatus

画面記述のための OIA モニターのタイプをセットします。

プロトタイプ

void AddOIAInhibitStatus(Variant type)

パラメーター

Variant type

OIA 状況のタイプ。有効な値は次のとおりです。

0. 任意
1. 使用可能

戻り値

なし

例

```
Dim autECLPSObj as Object
Dim autECLScreenDescObj as Object

Set autECLScreenDescObj = CreateObject("ZIEWin.autECLScreenDesc")
```

```
Set autECLPSObj = CreateObject("ZIEWin.autECLPS")
autECLPSObj.SetConnectionByName "A"

autECLScreenDesObj.AddCursorPos 23, 1
autECLScreenDesObj.AddAttrib E8h, 1, 1, 2
autECLScreenDesObj.AddCursorPos 23,1
autECLScreenDesObj.AddNumFields 45
autECLScreenDesObj.AddNumInputFields 17
autECLScreenDesObj.AddOIAInhibitStatus 1
autECLScreenDesObj.AddString "LOGON", 23, 11, True
autECLScreenDesObj.AddStringInRect "PASSWORD", 23, 1, 24, 80, False

if (autECLPSObj.WaitForScreen(autECLScreenDesObj, 10000)) then
    msgbox "Screen reached"
else
    msgbox "Timeout Occurred"
end if
```

AddString

画面記述の指定された位置にストリングを追加します。

プロトタイプ

void AddString(String str, Variant row, Variant col, [optional] Boolean caseSense)

パラメーター

String str

追加するストリング。

Variant row

行位置。

Variant col

桁位置。

Boolean caseSense

この値が True である場合は、ストリングは大/小文字の区別付きで 追加されます。このパラメーターはオプションです。デフォルト値は True です。

戻り値

なし

例

```
Dim autECLPSObj as Object
Dim autECLScreenDescObj as Object
```

```
Set autECLScreenDescObj = CreateObject("ZIEWin.autECLScreenDesc")
Set autECLPSObj = CreateObject("ZIEWin.autECLPS")
autECLPSObj.SetConnectionByName "A"

autECLScreenDesObj.AddCursorPos 23, 1
autECLScreenDesObj.AddAttrib E8h, 1, 1, 2
autECLScreenDesObj.AddCursorPos 23,1
autECLScreenDesObj.AddNumFields 45
autECLScreenDesObj.AddNumInputFields 17
autECLScreenDesObj.AddOIAInhibitStatus 1
autECLScreenDesObj.AddString "LOGON", 23, 11, True
autECLScreenDesObj.AddStringInRect "PASSWORD", 23, 1, 24, 80, False

if (autECLPSObj.WaitForScreen(autECLScreenDesObj, 10000)) then
    msgbox "Screen reached"
else
    msgbox "Timeout Occurred"
end if
```

AddStringInRect

画面記述の指定長方形内にストリングを追加します。

プロトタイプ

void AddStringInRect(String str, Variant sRow, Variant sCol, Variant eRow, Variant eCol, [optional] Variant caseSense)

パラメーター

String str

追加するストリング

Variant sRow

左上行位置。

Variant sCol

左上桁位置。

Variant eRow

右下行位置。

Variant eCol

右下桁位置。

Variant caseSense

この値が True である場合は、ストリングは大/小文字の区別付きで 追加されます。このパラメーターはオプションです。デフォルト値は True です。

戻り値

なし

例

```
Dim autECLPSObj as Object
Dim autECLScreenDescObj as Object

Set autECLScreenDescObj = CreateObject("ZIEWin.autECLScreenDesc")
Set autECLPSObj = CreateObject("ZIEWin.autECLPS")
autECLPSObj.SetConnectionByName "A"

autECLScreenDesObj.AddCursorPos 23, 1
autECLScreenDesObj.AddAttrib E8h, 1, 1, 2
autECLScreenDesObj.AddCursorPos 23,1
autECLScreenDesObj.AddNumFields 45
autECLScreenDesObj.AddNumInputFields 17
autECLScreenDesObj.AddOIAInhibitStatus 1
autECLScreenDesObj.AddString "LOGON", 23, 11, True
autECLScreenDesObj.AddStringInRect "PASSWORD", 23, 1, 24, 80, False

if (autECLPSObj.WaitForScreen(autECLScreenDesObj, 10000)) then
    msgbox "Screen reached"
else
    msgbox "Timeout Occurred"
end if
```

クリア

画面記述からすべての記述要素を取り除きます。

プロトタイプ

void Clear()

パラメーター

なし

戻り値

なし

例

```
Dim autECLPSObj as Object
Dim autECLScreenDescObj as Object

Set autECLScreenDescObj = CreateObject("ZIEWin.autECLScreenDesc")
Set autECLPSObj = CreateObject("ZIEWin.autECLPS")
```

```
autECLPSObj.SetConnectionByName "A"

autECLScreenDesObj.AddCursorPos 23, 1
autECLScreenDesObj.AddAttrib E8h, 1, 1, 2
autECLScreenDesObj.AddCursorPos 23,1
autECLScreenDesObj.AddNumFields 45
autECLScreenDesObj.AddNumInputFields 17
autECLScreenDesObj.AddOIAInhibitStatus 1
autECLScreenDesObj.AddString "LOGON", 23, 11, True
autECLScreenDesObj.AddStringInRect "PASSWORD", 23, 1, 24, 80, False

if (autECLPSObj.WaitForScreen(autECLScreenDesObj, 10000)) then
    msgbox "Screen reached"
else
    msgbox "Timeout Occurred"
end if

autECLScreenDesObj.Clear // start over for a new screen
```

autECLScreenReco クラス

autECLScreenReco クラスは、ホスト・アクセス・クラス・ライブラリーの画面認識システム にとってはエンジンに相当するものです。これには、画面の記述を追加したり除去したりするためのメソッドが含まれています。また、これにはこれらの画面を認識したり、これら画面用に ユーザーのイベント・ハンドラー・コードを非同期的に呼び戻したりするためのロジックも 含まれています。

autECLScreenReco クラスのオブジェクトを固有な認識セットと考えてください。このオブジェクトは、これが画面用に監視する複数の autECLPS オブジェクト、および 探索すべき複数の画面を持つことができ、これが追加された autECLPS オブジェクトの いずれかで登録済み画面を見つけた場合には、これはユーザーのアプリケーションで 定義されたイベント処理コードを破棄します。

ユーザーが実施すべきことは、アプリケーションを開始するときに ユーザーの autECLScreenReco オブジェクトを設定するだけです。 モニターしたい任意の autECLPS で画面が表示されると、イベント・コード が autECLScreenReco によって呼び出されます。ユーザーは、画面をモニターする 際に何も実行する必要はありません。

例については、イベント処理の例 (ページ 352)を参照してください。

autECLScreenReco メソッド

以下のセクションで、autECLScreenReco に有効なメソッドを説明します。

void AddPS(autECLPS ps) Boolean IsMatch(autECLPS ps, AutECLScreenDesc sd) void RegisterScreen(AutECLScreenDesc sd) void RemovePS(autECLPS ps) void UnregisterScreen(AutECLScreenDesc sd)

AddPS

autECLScreenReco オブジェクトに対して、モニターすべき autECLPS オブジェクトを追加します。

プロトタイプ

void AddPS(autECLPS ps)

パラメーター

autECLPS ps

モニター対象の PS オブジェクト。

戻り値

なし

例

例については、イベント処理の例 (ページ 352)を参照してください。

IsMatch

autECLPS オブジェクトおよび AutECLScreenDesc オブジェクトを渡すことを可能にし、 画面記述が PS の現在の状態に一致しているかどうかの判別を可能にします。画面認識のエンジンは このロジックを使用しますが、どのルーチンもそれを呼び出すことができるように作られます。

プロトタイプ

Boolean IsMatch(autECLPS ps, AutECLScreenDesc sd)

パラメーター

autECLPS ps

比較する autPS オブジェクト。

AutECLScreenDesc sd

比較する autECLScreenDesc オブジェクト。

戻り値

AutECLScreenDesc オブジェクトが PS 内の現行画面に一致する場合は True で、 それ以外の場合は False です。

例

```
Dim autPSObj as Object
Dim autECLScreenDescObj as Object
```

```
Set autECLScreenDescObj = CreateObject("ZIEWin.autECLScreenDesc")
Set autPSObj = CreateObject("ZIEWin.autECLPS")
autPSObj.SetConnectionByName "A"

autECLScreenDesObj.AddCursorPos 23, 1
autECLScreenDesObj.AddAttrib E8h, 1, 1, 2
autECLScreenDesObj.AddNumFields 45
autECLScreenDesObj.AddNumInputFields 17
autECLScreenDesObj.AddOIAInhibitStatus 1
autECLScreenDesObj.AddString "LOGON", 23, 11, True
autECLScreenDesObj.AddStringInRect "PASSWORD", 23, 1, 24, 80, False

if (autECLScreenReco.IsMatch(autPSObj, autECLScreenDesObj)) then
    msgbox "matched"
else
    msgbox "no match"
end if
```

RegisterScreen

指定された画面の記述のために画面認識オブジェクトに追加された すべての autECLPS オブジェクトのモニターを開始します。その画面が PS に現れると、NotifyRecoEvent が発生します。

プロトタイプ

void RegisterScreen(AutECLScreenDesc sd)

パラメーター

AutECLScreenDesc sd

登録対象の画面記述オブジェクト。

戻り値

なし

例

例については、イベント処理の例 (ページ 352)を参照してください。

RemovePS

画面認識モニターから autECLPS オブジェクトを除去します。

プロトタイプ

void RemovePS(autECLPS ps)

パラメーター

autECLPS ps

除去する autECLPS オブジェクト。

戻り値

なし

例

例については、イベント処理の例 (ページ 352)を参照してください。

UnregisterScreen

画面認識モニターから画面記述を除去します。

プロトタイプ

void UnregisterScreen(AutECLScreenDesc sd)

パラメーター

AutECLScreenDesc sd

除去する画面記述オブジェクト。

戻り値

なし

例

例については、イベント処理の例 (ページ 352)を参照してください。

autECLScreenReco イベント

以下のイベントは autECLScreenReco に有効です。

void NotifyRecoEvent(AutECLScreenDesc sd, autECLPS ps) void NotifyRecoError() void NotifyRecoStop(Long Reason)

NotifyRecoEvent

このイベントは、autECLScreenReco オブジェクトに追加された PS に登録済み画面記述が 現れたときに起こります。

プロトタイプ

void NotifyRecoEvent(AutECLScreenDesc sd, autECLPS ps)

パラメーター

AutECLScreenDesc sd

その基準を満たしている画面記述オブジェクト。

autECLPS ps

突き合わせが行われた PS オブジェクト。

例

例については、イベント処理の例 (ページ 352)を参照してください。

NotifyRecoError

このイベントは、イベント処理でエラーが発生したときに起こります。

プロトタイプ

void NotifyRecoError()

パラメーター

なし

例

例については、イベント処理の例 (ページ 352)を参照してください。

NotifyRecoStop

このイベントは、イベント処理が停止したときに起こります。

プロトタイプ

void NotifyRecoStop(Long Reason)

パラメーター

Long Reason

停止の理由コード。現在は、これは常に 0 です。

イベント処理の例

以下は、画面認識イベントの実装方法についての簡単な例です。

```
Dim myPS as Object
Dim myScreenDesc as Object
Dim WithEvents reco as autECLScreenReco  'autECLScreenReco added as reference

Sub Main()
  ' Create the objects
  Set reco= new autECLScreenReco
  myScreenDesc = CreateObject("ZIEWin.autECLScreenDesc")
  Set myPS = CreateObject("ZIEWin.autECLPS")
  myPS.SetConnectionByName "A"

  ' Set up the screen description
  myScreenDesc.AddCursorPos 23, 1
  myScreenDesc.AddString "LOGON"
  myScreenDesc.AddNumFields 59

  ' Add the PS to the reco object (can add multiple PS's)
  reco.addPS myPS

  ' Register the screen (can add multiple screen descriptions)
  reco.RegisterScreen myScreenDesc

  ' Display your form or whatever here  (this should be a blocking call, otherwise sub just ends
  call DisplayGUI()

  ' Clean up
  reco.UnregisterScreen myScreenDesc
  reco.RemovePS myPS
  set myPS = Nothing
  set myScreenDesc = Nothing
  set reco = Nothing
End Sub

'This sub will get called when the screen Description registered above appears in
'Session A.  If multiple PS objects or screen descriptions were added,  you can
'determine which screen and which PS via the parameters.

Sub reco_NotifyRecoEvent(autECLScreenDesc SD, autECLPS PS)
  If (reco.IsMatch(PS,myScreenDesc)) Then
    ' do your processing for your screen here
  End If
End Sub

Sub reco_NotifyRecoError
   'do your error handling here
```

```
End sub

Sub reco_NotifyRecoStop(Reason as Long)
    'Do any stop processing here
End sub
```

autECLSession クラス

autECLSession オブジェクトは、 一般エミュレーター関連サービスを提供し、ホスト・アクセス・クラス・ライブラリーの 他の主要なオブジェクトへのポインターを含んでいます。レジストリーでのその名前は ZIEWin.autECLSession です。

autECLSession に含まれているオブジェクトはそれぞれ 独立させることができますが、これらへのポインターは autECLSession クラスに 存在します。autECLSession オブジェクトが作成されるときは、autECLPS、autECLOIA、autECLXfer、autECLWindowMetrics、autECLPageSettings、 および autECLPrinterSettings オブジェクトも作成されます。それらは、他のプロパティーと同じように参照してください。

注:

1. このオブジェクトの現行バージョンは 1.2 です。このオブジェクトには 2 つのバージョンがあり、レジストリー内の ProgID はそれぞれ ZIEWin.autECLSession.1 および ZIEWin.autECLSession.2 です。バージョンを区別しない ProgID は、ZIEWin.autECLSession です。ZIEWin.autECLSession.1 オブジェクトは、プロパティー autECLPageSettings および autECLPrinterSettings をサポートしません。
2. 最初に、作成したオブジェクトの接続を設定しなければなりません。SetConnectionByName または SetConnectionByHandle を使用して、オブジェクトを初期化します。接続は一度しか設定できません。接続が設定された後は、SetConnection メソッドをさらに呼び出すと例外を引き起こします。また、接続を設定せずに autECLSession プロパティーまたはメソッドにアクセスしようと しても、例外が引き起こされます。

以下の例は、Visual Basic で autECLSession オブジェクトを作成し、 設定する方法を示しています。

```
DIM  SessObj as Object
Set SessObj = CreateObject("ZIEWin.autECLSession")

' Initialize the session
SessObj.SetConnectionByName("A")
' For example, set the host window to minimized
SessObj.autECLWinMetrics.Minimized = True
```

プロパティー

このセクションでは、autECLSession オブジェクトのプロパティーを説明します。

タイプ	名前	属性
ストリング	名前	読み取り専用
長形式	ハンドル	読み取り専用
ストリング	ConnType	読み取り専用
長形式	CodePage	読み取り専用
ブール値	開始済み	読み取り専用
ブール値	CommStarted	読み取り専用
ブール値	APIEnabled	読み取り専用
ブール値	作動可能	読み取り専用
オブジェクト	autECLPS	読み取り専用
オブジェクト	autECLOIA	読み取り専用
オブジェクト	autECLXfer	読み取り専用
オブジェクト	autECLWinMetrics	読み取り専用
オブジェクト	autECLPageSettings	読み取り専用
オブジェクト	autECLPrinterSettings	読み取り専用

名前

このプロパティーは、autECLSession が設定された接続の接続名ストリングです。Z and I Emulator for Windows は、短い文字 ID (A から Z、または a から z) のみをストリングで戻します。Z and I Emulator for Windows 接続でオープンできるのは、1 つの名前につき 1 つしかありません。例えば、一度に 1 つの接続 A のみをオープンできます。Name は、String データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM  Name as String
DIM  SessObj as Object
Set SessObj = CreateObject("ZIEWin.autECLSession")

' Initialize the session
SessObj.SetConnectionByName("A")

' Save the name
Name = SessObj.Name
```

ハンドル

これは、autECLSession オブジェクトが 設定された接続のハンドルです。特定の 1 つのハンドルに対して 1 つしか、Z and I Emulator for Windows 接続をオープンできません。例えば、一度に 1 つの接続 A のみをオープンできます。Handle は、Long データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM  SessObj as Object
Set SessObj = CreateObject("ZIEWin.autECLSession")

' Initialize the session
SessObj.SetConnectionByName("A")
```

```
' Save the session handle
Hand = SessObj.Handle
```

ConnType

これは、autECLXfer が設定された接続タイプです。このタイプは、時間の経過とともに変更する場合が あります。ConnType は、String データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM  Type as String
DIM  SessObj as Object

Set SessObj = CreateObject("ZIEWin.autECLSession")

' Initialize the session
SessObj.SetConnectionByName("A")
' Save the type
Type = SessObj.ConnType
```

ConnType プロパティーの接続タイプは、以下のとおりです。

戻されるストリング	意味
DISP3270	3270 表示装置
DISP5250	5250 ディスプレイ
PRNT3270	3270 印刷装置
PRNT5250	5250 プリンター
ASCII	VT エミュレーション

CodePage

これは、autECLXfer が設定された接続のコード・ページです。このコード・ページは、時間の経過とともに変更される場合があります。CodePage は、Long データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM  CodePage as Long
DIM  SessObj as Object
Set SessObj = CreateObject("ZIEWin.autECLSession")

' Initialize the session
SessObj.SetConnectionByName("A")
' Save the code page
CodePage = SessObj.CodePage
```

開始済み

これは、エミュレーター・ウィンドウが開始されたかどうかを示します。ウィンドウがオープンしている場合、値は True です。 その他の場合は False です。Started は、Boolean データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM  Hand as Long
DIM  SessObj as Object

Set SessObj = CreateObject("ZIEWin.autECLSession")

' Initialize the session
SessObj.SetConnectionByName("A")
' This code segment checks to see if A is started.
' The results are sent to a text box called Result.
If SessObj.Started = False Then
  Result.Text = "No"
Else
  Result.Text = "Yes"
End If
```

CommStarted

これは、ホストへの接続の状況を示しています。ホストが接続されている場合、値は True です。 その他の場合は False です。CommStarted は、Boolean データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM  Hand as Long
DIM  SessObj as Object

Set SessObj = CreateObject("ZIEWin.autECLSession")

' Initialize the session
SessObj.SetConnectionByName("A")

' This code segment checks to see if communications are connected
' for session A.  The results are sent to a text box called
' CommConn.
If SessObj.CommStarted = False Then
    CommConn.Text = "No"
Else
    CommConn.Text = "Yes"
End If
```

APIEnabled

これは、エミュレーターが API 使用可能かどうかを示します。API 設定 (Z and I Emulator for Windows のウィンドウでは、**「ファイル」->「API の設定」**を選択) の状態に応じて、接続が可能な場合と、接続できない場合があります。エミュレーターが使用可能の場合には、True です。その他の場合には、False です。APIEnabled は、Boolean データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM  Hand as Long
DIM  SessObj as Object

Set SessObj = CreateObject("ZIEWin.autECLSession")

' Initialize the session
SessObj.SetConnectionByName("A")
```

```
' This code segment checks to see if A is API enabled.
' The results are sent to a text box called Result.
If SessObj.APIEnabled = False Then
  Result.Text = "No"
Else
  Result.Text = "Yes"
End If
```

作動可能

これは、エミュレーター・ウィンドウが 開始されていて、API 使用可能で、そして接続されているかどうかを示します。このプロパティーは、3 つのすべてのプロパティーを確認します。エミュレーターが準備できている場合には、値は True です。 その他の場合には、False です。Ready は、Boolean データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM  Hand as Long
DIM  SessObj as Object

Set SessObj = CreateObject("ZIEWin.autECLSession")

' Initialize the session
SessObj.SetConnectionByName("A")

' This code segment checks to see if A is ready.
' The results are sent to a text box called Result.
If SessObj.Ready = False Then
  Result.Text = "No"
Else
  Result.Text = "Yes"
End If
```

autECLPS オブジェクト

autECLPS オブジェクトを使用すると、ZIEWin.autECLPS クラスに含まれるメソッドにアクセスすることができます。詳しくは、autECLPS クラス (ページ 299)を参照してください。以下の例は、このオブジェクトを示しています。

```
DIM  SessObj as Object
DIM  PSSize as Long
Set SessObj = CreateObject("ZIEWin.autECLSession")

' Initialize the session
SessObj.SetConnectionByName("A")
' For example, get the PS size
PSSize = SessObj.autECLPS.GetSize()
```

autECLOIA オブジェクト

autECLOIA オブジェクトを使用すると、ZIEWin.autECLOIA クラスに含まれるメソッドにアクセスすることができます。詳しくは、autECLOIA クラス (ページ 281)を参照してください。以下の例は、このオブジェクトを示しています。

```
DIM  SessObj as Object
Set SessObj = CreateObject("ZIEWin.autECLSession")

' Initialize the session
SessObj.SetConnectionByName("A")
' For example, set the host window to minimized
If (SessObj.autECLOIA.Katakana) Then
 'whatever
Endif
```

autECLXfer オブジェクト

autECLXfer オブジェクトを使用すると、ZIEWin.autECLXfer クラスに含まれるメソッドにアクセスすることができます。詳しくは、autECLXfer クラス (ページ 383)を参照してください。以下の例は、このオブジェクトを示しています。

```
DIM  SessObj as Object
Set SessObj = CreateObject("ZIEWin.autECLSession")

' Initialize the session
SessObj.SetConnectionByName("A")
' For example
SessObj.Xfer.Sendfile "c:\temp\filename.txt",
       "filename text a0",
       "CRLF ASCII"
```

autECLWinMetrics オブジェクト

autECLWinMetrics オブジェクトを使用すると、ZIEWin.autECLWinMetrics クラスに含まれるメソッドにアクセスすることができます。詳しくは、autECLWinMetrics クラス (ページ 366)を参照してください。以下の例は、このオブジェクトを示しています。

```
DIM  SessObj as Object
Set SessObj = CreateObject("ZIEWin.autECLSession")

' Initialize the session
SessObj.SetConnectionByName("A")
' For example, set the host window to minimized
SessObj.autECLWinMetrics.Minimized = True
```

autECLPageSettings オブジェクト

autECLPageSettings オブジェクトを使用すると、ZIEWin.autECLPageSettings クラスに含まれるメソッドにアクセスすることができます。詳しくは、autECLPageSettings クラス (ページ 397)を参照してください。

以下の例で、autECLPageSettings オブジェクトを示します。

```
DIM SessObj as Object
Set SessObj = CreateObject("ZIEWin.autECLSession")
'Initialize the session
SessObj.SetConnectionByName("A")

'For example, set the FaceName
SessObj.autECLPageSettings.FaceName = "Courier New"
```

autECLPageSettings オブジェクトは VBSCRIPT でもサポートされています。以下の例は、VBSCRIPT を使用する方法を示しています。

```
sub test_()
  autECLSession.SetConnectionByName(ThisSessionName)
  autECLSession.autECLPageSettings.FaceName="Courier"
 end sub
```

autECLPrinterSettings オブジェクト

autECLPrinterSettings オブジェクトを使用すると、ZIEWin.autECLPrinterSettings クラスに含まれるメソッドにアクセスすることができます。詳しくは、autECLPageSettings クラス (ページ 397)を参照してください。

以下の例で、autECLPageSettings オブジェクトを示します。

```
DIM SessObj as Object
Set SessObj = CreateObject("ZIEWin.autECLSession")
' Initialize the session
SessObj.SetConnectionByName("A")

'For example, set the Windows default printer
SessObj.autECLPrinterSettings.SetWinDefaultPrinter
```

autECLPrinterSettings オブジェクトは VBSCRIPT でもサポートされています。以下の例は、VBSCRIPT を使用する方法を示しています。

```
sub test_()
  autECLSession.SetConnectionByName(ThisSessionName)
  autECLSession.autECLPrinterSettings.SetWinDefaultPrinter
end sub
```

autECLSession メソッド

以下のセクションで、autECLSession オブジェクトに有効な メソッドを説明します。

void RegisterSessionEvent(Long updateType) void RegisterCommEvent() void UnregisterSessionEvent() void UnregisterCommEvent() void SetConnectionByName (String Name) void SetConnectionByHandle (Long Handle) void StartCommunication() void StopCommunication()

RegisterSessionEvent

このメソッドは、指定されたセッション・イベントの通知を受け取る ための autECLSession オブジェクトを登録します。

注: このメソッドはサポートされません。また、使用を推奨しません。

プロトタイプ

void RegisterSessionEvent(Long updateType)

パラメーター

Long updateType

モニターすべき更新のタイプ。

1. PS の更新
2. OIA の更新
3. PS または OIA の更新

戻り値

なし

例

例については、イベント処理の例 (ページ 366)を参照してください。

RegisterCommEvent

このメソッドは、すべての通信リンク接続/接続解除のイベントの通知を受け取るための オブジェクトを登録します。

プロトタイプ

void RegisterCommEvent()

パラメーター

なし

戻り値

なし

例

例については、イベント処理の例 (ページ 366)を参照してください。

UnregisterSessionEvent

セッション・イベント処理を終了します。

注: このメソッドはサポートされません。また、使用を推奨しません。

プロトタイプ

void UnregisterSessionEvent()

パラメーター

なし

戻り値

なし

例

例については、イベント処理の例 (ページ 366)を参照してください。

UnregisterCommEvent

通信リンク・イベント処理を終了します。

プロトタイプ

void UnregisterCommEvent()

パラメーター

なし

戻り値

なし

例

例については、イベント処理の例 (ページ 366)を参照してください。

SetConnectionByName

このメソッドは、接続名を使用して、新しく作成された autECLSession オブジェクトの 接続を設定します。Z and I Emulator for Windows では、この接続名は、短い ID (文字 A から Z、または a から z) です。Z and I Emulator for Windows 接続でオープンできるのは、1 つの名前につき 1 つしかありません。例えば、一度に 1 つの接続 A のみをオープンできます。

プロトタイプ

void SetConnectionByName (String Name)

パラメーター

String 型、Name 型

1 文字の接続のストリング短縮名 (A から Z、または a から z)。

戻り値

なし

例

以下の例は、接続名を使用して、新しく作成された autECLSession オブジェクトの接続を 設定する方法を示します。

```
DIM  SessObj as Object
Set SessObj = CreateObject("ZIEWin.autECLSession")

' Initialize the session
SessObj.SetConnectionByName("A")
' For example, set the host window to minimized
SessObj.autECLWinMetrics.Minimized = True
```

SetConnectionByHandle

このメソッドは、接続ハンドルを使用して、新しく作成された autECLSession オブジェクトの 接続を設定します。Z and I Emulator for Windows では、この接続ハンドルは Long integer です。特定の 1 つのハンドルに対して 1 つしか、Z and I Emulator for Windows 接続をオープンできません。例えば、一度に 1 つの接続 A のみをオープンできます。

プロトタイプ

void SetConnectionByHandle(Long Handle)

パラメーター

Long Handle

オブジェクトに設定される接続の Long integer 値。

戻り値

なし

例

以下の例は、接続ハンドルを使用して、新しく作成された autECLSession オブジェクトの 接続を設定する方法を示します。

```
Dim  SessObj as Object
Dim autECLConnList as Object

Set autECLConnList = CreateObject("ZIEWin.autECLConnList")
Set SessObj = CreateObject("ZIEWin.autECLSession")

' Initialize the session
autECLConnList.Refresh
autECLPSObj.SetConnectionByHandle(autECLConnList(1).Handle)
```

StartCommunication

StartCommunication 集合要素メソッドは、ホスト・データ・ストリームに ZIEWin エミュレーターを接続します。これは、ZIEWin エミュレーター**「通信」**メニューを表示して**「接続」**を選択した場合と同じ結果になります。

プロトタイプ

void StartCommunication()

パラメーター

なし

戻り値

なし

例

以下の例は、ZIEWin エミュレーター・セッションをホストへ接続する方法を示しています。

```
Dim SessObj as Object
Dim autECLConnList as Object

Set autECLConnList = CreateObject("ZIEWin.autECLConnList")
Set SessObj = CreateObject("ZIEWin.autECLSession")

' Initialize the session
autECLConnList.Refresh
SessObj.SetConnectionByHandle(autECLConnList(1).Handle)

SessObj.StartCommunication()
```

StopCommunication

StopCommunication 集合要素メソッドは、ホスト・データ・ストリームから ZIEWin エミュレーターを切断します。これは、ZIEWin エミュレーター**「通信」**メニューを表示して**「切断」**を選択した場合と同じ結果になります。

プロトタイプ

void StopCommunication()

パラメーター

なし

戻り値

なし

例

以下の例は、ZIEWin エミュレーター・セッションをホストへ接続する方法を示しています。

```
Dim SessObj as Object
Dim autECLConnList as Object

Set autECLConnList = CreateObject("ZIEWin.autECLConnList")
Set SessObj = CreateObject("ZIEWin.autECLSession")

' Initialize the session
autECLConnList.Refresh
SessObj.SetConnectionByHandle(autECLConnList(1).Handle)

SessObj.StopCommunication()
```

autECLSession イベント

以下のイベントは、autECLSession に有効です。

void NotifyCommEvent(boolean bConnected) void NotifyCommError() void NotifyCommStop(Long Reason)

NotifyCommEvent

指定された通信リンクは、接続あるいは接続解除されています。

プロトタイプ

void NotifyCommEvent(boolean bConnected)

パラメーター

boolean bConnected

通信リンクが現在接続されている場合は True。それ以外の場合は FALSE。

例

例については、イベント処理の例 (ページ 366)を参照してください。

NotifyCommError

このイベントは、イベント処理でエラーが発生したときに起こります。

プロトタイプ

void NotifyCommError()

パラメーター

なし

例

例については、イベント処理の例 (ページ 366)を参照してください。

NotifyCommStop

このイベントは、イベント処理が停止したときに起こります。

プロトタイプ

void NotifyCommStop(Long Reason)

パラメーター

Long Reason

停止の理由コード。現在は、これは常に 0 です。

イベント処理の例

以下は、セッション・イベントの実装方法についての簡単な例です。

```
Option Explicit
Private WithEvents mSess As autECLSession   'AutSess added as reference

sub main()
   'Create Objects
   Set mSess = New autECLSession
   mSess.SetConnectionByName "A"
    mSess.RegisterCommEvent          'register for communication link notifications
   ' Display your form or whatever here  (this should be a blocking call, otherwise sub just ends
   call DisplayGUI()
   mSess.UnregisterCommEvent
   set mSess = Nothing
End Sub

'This sub will get called when the Communication Link Status of the registered
'connection changes
Private Sub mSess_NotifyCommEvent()
    ' do your processing here
End Sub

'This event occurs if an error happens in Communications Link event processing
Private Sub mSess_NotifyCommError()
   'Do any error processing here
End Sub

'This event occurs when Communications Status Notification ends
Private Sub mSess_NotifyCommStop()
   'Do any stop processing here
End Sub
```

autECLWinMetrics クラス

autECLWinMetrics オブジェクト は、エミュレーター・ウィンドウでの操作を実行します。これによって、ウィンドウ長方形および位置の操作 (例えば、SetWindowRect、Ypos、および Width) を、 ウィンドウ状態の操作 (例えば、Visible または Restored) と同じように実行できます。レジストリーでのその名前は ZIEWin.autECLWinMetrics です。

最初に、作成したオブジェクトの接続を設定しなければなりません。SetConnectionByName または SetConnectionByHandle を使用して、オブジェクトを初期化します。接続は一度しか設定できません。接続が設定さ

れた後は、接続設定メソッドをさらに呼び出すと例外を引き起こします。また、接続を設定せずにプロパティーまたはメソッドにアクセスしようとしても、 例外が引き起こされます。

注: autECL オブジェクト内の autECLSession オブジェクト は、autECL オブジェクトにより設定されます。

以下の例は、Visual Basic で autECLWinMetrics オブジェクトを作成し 設定する方法を示しています。

```
DIM  autECLWinObj as Object
Set autECLWinObj = CreateObject("ZIEWin.autECLWinMetrics")

' Initialize the connection
autECLWinObj.SetConnectionByName("A")
' For example, set the host window to minimized
autECLWinObj.Minimized = True
```

プロパティー

このセクションでは、autECLWinMetrics オブジェクトのプロパティーを説明します。

タイプ	名前	属性
ストリング	WindowTitle	Read/Write
長形式	Xpos	Read/Write
長形式	Ypos	Read/Write
長形式	Width	Read/Write
長形式	Height	Read/Write
ブール値	表示	Read/Write
ブール値	アクティブ	Read/Write
ブール値	最小化	Read/Write
ブール値	最大化	Read/Write
ブール値	復元	Read/Write
ストリング	名前	読み取り専用
長形式	ハンドル	読み取り専用
ストリング	ConnType	読み取り専用
長形式	CodePage	読み取り専用
ブール値	開始済み	読み取り専用
ブール値	CommStarted	読み取り専用
ブール値	APIEnabled	読み取り専用
ブール値	作動可能	読み取り専用

WindowTitle

これは、autECLWinMetrics オブジェクトに 関連する接続のタイトル・バーに現在あるタイトルです。このプロパティーは、変更と検索の両方が可能です。WindowTitle は、String データ型で読み取り/書き込み可能です。以下の例は、このプロセスを示しています。以下の例は、このプロパティーを示しています。

```
Dim  autECLWinObj as Object
Dim  ConnList as Object
Dim  WinTitle as String
Set autECLWinObj = CreateObject("ZIEWin.autECLWinMetrics")
Set ConnList = CreateObject("ZIEWin.autECLConnList")

' Initialize the connection
ConnList.Refresh
autECLWinObj.SetConnectionByHandle(ConnList(1).Handle)

WinTitle = autECLWinObj.WindowTitle 'get the window title

' or...

autECLWinObj.WindowTitle = "Flibberdeejibbet" 'set the window title
```

使用上の注意

ウィンドウ・タイトルがブランクに設定された場合、接続のウィンドウ・タイトルは その元の設定に復元されます。

Xpos

これは、エミュレーター・ウィンドウ長方形の上方左の角の *x* 位置です。このプロパティーは、変更と検索の両方が可能です。Xpos は、Long データ型で 読み取り/書き込み可能です。ただし、付加した接続がインプレースの組み込みオブジェクトである場合は、 このプロパティーは読み取り専用です。以下の例は、このプロパティーを示しています。

```
Dim  autECLWinObj as Object
Dim  ConnList as Object
Dim  x as Long
Set autECLWinObj = CreateObject("ZIEWin.autECLWinMetrics")
Set ConnList = CreateObject("ZIEWin.autECLConnList")

' Initialize the connection
ConnList.Refresh
autECLWinObj.SetConnectionByHandle(ConnList(1).Handle)

x = autECLWinObj.Xpos 'get the x position

' or...

autECLWinObj.Xpos = 6081 'set the x position
```

Ypos

これは、エミュレーター・ウィンドウ長方形の上方左の角の y 位置です。このプロパティーは、変更と検索の両方が可能です。Ypos は、Long データ型で読み取り/書き込み可能です。ただし、付加した接続がインプレースの組み込みオブジェクトである場合は、 このプロパティーは読み取り専用です。以下の例は、このプロパティーを示しています。

```
Dim  autECLWinObj as Object
Dim  ConnList as Object
Dim  y as Long
Set autECLWinObj = CreateObject("ZIEWin.autECLWinMetrics")
Set ConnList = CreateObject("ZIEWin.autECLConnList")

' Initialize the connection
ConnList.Refresh
autECLWinObj.SetConnectionByHandle(ConnList(1).Handle)

y = autECLWinObj.Ypos 'get the y position

' or...

autECLWinObj.Ypos = 6081 'set the y position
```

Width

これは、エミュレーター・ウィンドウ長方形の幅です。このプロパティーは、変更と検索の両方が可能です。Width は、Long データ型で 読み取り/書き込み可能です。ただし、付加した接続がインプレースの組み込みオブジェクトである場合は、 このプロパティーは読み取り専用です。以下の例は、このプロパティーを示しています。

```
Dim  autECLWinObj as Object
Dim  ConnList as Object
Dim  cx as Long
Set autECLWinObj = CreateObject("ZIEWin.autECLWinMetrics")
Set ConnList = CreateObject("ZIEWin.autECLConnList")

' Initialize the connection
ConnList.Refresh
autECLWinObj.SetConnectionByHandle(ConnList(1).Handle)

cx = autECLWinObj.Width 'get the width

' or...

autECLWinObj.Width = 6081 'set the width
```

Height

これは、エミュレーター・ウィンドウ長方形の高さです。このプロパティーは、変更と検索の両方が可能です。Height は、Long データ型で 読み取り/書き込み可能です。ただし、付加した接続がインプレースの組み込みオブジェクトである場合は、 このプロパティーは読み取り専用です。以下の例は、このプロパティーを示しています。

```
Dim  autECLWinObj as Object
Dim  ConnList as Object
Dim  cy as Long
Set autECLWinObj = CreateObject("ZIEWin.autECLWinMetrics")
Set ConnList = CreateObject("ZIEWin.autECLConnList")

' Initialize the connection
ConnList.Refresh
autECLWinObj.SetConnectionByHandle(ConnList(1).Handle)

cy = autECLWinObj.Height 'get the height

' or...

autECLWinObj.Height = 6081 'set the height
```

表示

これは、エミュレーター・ウィンドウの可視状態です。このプロパティーは、変更と検索の両方が可能です。Visible は、Boolean データ型で読み取り/書き込み可能です。ただし、付加した接続がインプレースの組み込みオブジェクトである場合は、 このプロパティーは読み取り専用です。以下の例は、このプロパティーを示しています。

```
Dim  autECLWinObj as Object
Dim  ConnList as Object
Set autECLWinObj = CreateObject("ZIEWin.autECLWinMetrics")
Set ConnList = CreateObject("ZIEWin.autECLConnList")

' Initialize the connection
ConnList.Refresh
autECLWinObj.SetConnectionByHandle(ConnList(1).Handle)

' Set to Visible if not, and vice versa
If ( autECLWinObj.Visible) Then
 autECLWinObj.Visible = False
Else
 autECLWinObj.Visible = True
End If
```

アクティブ

これは、エミュレーター・ウィンドウのフォーカス状態です。このプロパティーは、変更と検索の両方が可能です。Active は、Boolean データ型で読み取り/書き込み可能です。ただし、付加した接続がインプレースの組み込みオブジェクトである場合は、 このプロパティーは読み取り専用です。以下の例は、このプロパティーを示しています。

```
Dim  autECLWinObj as Object
Dim  ConnList as Object
Set autECLWinObj = CreateObject("ZIEWin.autECLWinMetrics")
Set ConnList = CreateObject("ZIEWin.autECLConnList")

' Initialize the connection
```

```
ConnList.Refresh
autECLWinObj.SetConnectionByHandle(ConnList(1).Handle)

' Set to Active if not, and vice versa
If ( autECLWinObj.Active) Then
 autECLWinObj.Active = False
Else
 autECLWinObj.Active = True
End If
```

最小化

これは、エミュレーター・ウィンドウの最小化状態です。このプロパティーは、変更と検索の両方が可能です。Minimized は Boolean データ型で読み取り/書き込み可能です。ただし、付加した接続がインプレースの組み込みオブジェクトである場合は、 このプロパティーは読み取り専用です。以下の例は、このプロパティーを示しています。

```
Dim  autECLWinObj as Object
Dim  ConnList as Object
Set autECLWinObj = CreateObject("ZIEWin.autECLWinMetrics")
Set ConnList = CreateObject("ZIEWin.autECLConnList")

' Initialize the connection
ConnList.Refresh
autECLWinObj.SetConnectionByHandle(ConnList(1).Handle)

' Set to minimized if not, if minimized set to maximized
If ( autECLWinObj.Minimized) Then
 autECLWinObj.Maximized = True
Else
 autECLWinObj.Minimized = True
End If
```

最大化

これは、エミュレーター・ウィンドウの最大化状態です。このプロパティーは、変更と検索の両方が可能です。Maximized は、Boolean データ型で読み取り/書き込み可能です。ただし、付加した接続がインプレースの組み込みオブジェクトである場合は、 このプロパティーは読み取り専用です。以下の例は、このプロパティーを示しています。

```
Dim  autECLWinObj as Object
Dim  ConnList as Object
Set autECLWinObj = CreateObject("ZIEWin.autECLWinMetrics")
Set ConnList = CreateObject("ZIEWin.autECLConnList")

' Initialize the connection
ConnList.Refresh
autECLWinObj.SetConnectionByHandle(ConnList(1).Handle)

' Set to maximized if not, if maximized set to minimized
If ( autECLWinObj.Maximized) Then
 autECLWinObj.Minimized = False
```

```
Else
 autECLWinObj.Maximized = True
End If
```

復元

これは、エミュレーター・ウィンドウの復元状態です。Restored は、Boolean データ型で読み取り/書き込み可能です。ただし、付加した接続がインプレースの組み込みオブジェクトである場合は、 このプロパティーは読み取り専用です。以下の例は、このプロパティーを示しています。

```
Dim  autECLWinObj as Object
Dim  SessList as Object
Set autECLWinObj = CreateObject("ZIEWin.autECLWinMetrics")
Set SessList = CreateObject("ZIEWin.autECLConnList")

' Initialize the session
SessList.Refresh
autECLWinObj.SetSessionByHandle(SessList(1).Handle)

' Set to restored if not, if restored set to minimized
If ( autECLWinObj.Restored) Then
 autECLWinObj.Minimized = False
Else
 autECLWinObj.Restored = True
End If
```

名前

このプロパティーは、autECLWinMetrics が設定された接続の 接続名ストリングです。現在、Z and I Emulator for Windows は、短い文字 ID (A から Z、または a から z) のみをストリングで戻します。Z and I Emulator for Windows 接続でオープンできるのは、1 つの名前につき 1 つしかありません。例えば、一度に 1 つの接続 A のみをオープンできます。Name は、String データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM  Name as String
DIM  Obj as Object
Set Obj = CreateObject("ZIEWin.autECLWinMetrics")

' Initialize the connection
Obj.SetConnectionByName("A")

' Save the name
Name = Obj.Name
```

ハンドル

これは、autECLWinMetrics オブジェクトが 設定された接続のハンドルです。特定の 1 つのハンドルに対して 1 つしか、Z and I Emulator for Windows 接続をオープンできません。例えば、一度に 1 つの接続 A のみをオープンできます。Handle は、Long データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM  Obj as Object
Set Obj = CreateObject("ZIEWin.autECLWinMetrics")

' Initialize the connection
Obj.SetConnectionByName("A")

' Save the handle
Hand = Obj.Handle
```

ConnType

これは、autECLWinMetrics が設定された接続タイプです。このタイプは、時間の経過とともに変更する場合が あります。ConnType は、String データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM  Type as String
DIM  Obj as Object

Set Obj = CreateObject("ZIEWin.autECLWinMetrics")

' Initialize the connection
Obj.SetConnectionByName("A")
' Save the type
Type = Obj.ConnType
```

ConnType プロパティーの接続タイプは、以下のとおりです。

戻されるストリング	意味
DISP3270	3270 表示装置
DISP5250	5250 ディスプレイ
PRNT3270	3270 印刷装置
PRNT5250	5250 プリンター
ASCII	VT エミュレーション

CodePage

これは、autECLWinMetrics が設定された接続のコード・ページです。このコード・ページは、時間の経過とともに変更される場合があります。CodePage は、Long データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM  CodePage as Long
DIM  Obj as Object
Set Obj = CreateObject("ZIEWin.autECLWinMetrics")

' Initialize the connection
Obj.SetConnectionByName("A")
' Save the code page
CodePage = Obj.CodePage
```

開始済み

これは、エミュレーター・ウィンドウが開始されたかどうかを示します。ウィンドウがオープンしている場合、値は True です。 その他の場合は False です。Started は、Boolean データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM  Hand as Long
DIM  Obj as Object

Set Obj = CreateObject("ZIEWinZIEWin.autECLWinMetrics")

' Initialize the connection
Obj.SetConnectionByName("A")

' This code segment checks to see if A is started.
' The results are sent to a text box called Result.
If Obj.Started = False Then
  Result.Text = "No"
Else
  Result.Text = "Yes"
End If
```

CommStarted

これは、ホストへの接続の状況を示しています。ホストが接続されている場合、値は True です。 その他の場合は False です。CommStarted は、Boolean データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM  Hand as Long
DIM  Obj as Object

Set Obj = CreateObject("ZIEWin.autECLWinMetrics")

' Initialize the connection
Obj.SetConnectionByName("A")

' This code segment checks to see if communications are connected
' for A.  The results are sent to a text box called
' CommConn.
If Obj.CommStarted = False Then
    CommConn.Text = "No"
Else
    CommConn.Text = "Yes"
End If
```

APIEnabled

これは、エミュレーターが API 使用可能かどうかを示します。API 設定 (Z and I Emulator for Windows のウィンドウでは、**「ファイル」->「API の設定」**を選択) の状態に応じて、接続が可能な場合と、接続できない場合があります。エミュレーターが使用可能の場合には、True です。その他の場合には、False です。APIEnabled は、Boolean データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM  Hand as Long
DIM  Obj as Object

Set Obj = CreateObject("ZIEWin.autECLWinMetrics")

' Initialize the connection
Obj.SetConnectionByName("A")

' This code segment checks to see if A is API enabled.
' The results are sent to a text box called Result.
If Obj.APIEnabled = False Then
  Result.Text = "No"
Else
  Result.Text = "Yes"
End If
```

作動可能

これは、エミュレーター・ウィンドウが 開始されていて、API 使用可能で、そして接続されているかどうかを示します。このプロパティーは、3 つのすべてのプロパティーを確認します。エミュレーターが準備できている場合には、値は True です。 その他の場合には、False です。Ready は、Boolean データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM  Hand as Long
DIM  Obj as Object

Set Obj = CreateObject("ZIEWin.autECLWinMetrics")

' Initialize the connection
Obj.SetConnectionByName("A")

' This code segment checks to see if A is ready.
' The results are sent to a text box called Result.
If Obj.Ready = False Then
  Result.Text = "No"
Else
  Result.Text = "Yes"
End If
```

autECLWinMetrics メソッド

以下のセクションで、autECLWinMetrics オブジェクトに有効な メソッドを説明します。

void RegisterCommEvent() void UnregisterCommEvent() void SetConnectionByName(String Name) void SetConnectionByHandle(Long Handle) void GetWindowRect(Variant Left, Variant Top, Variant Right, Variant Bottom) void SetWindowRect(Long Left, Long Top, Long Right, Long Bottom) void StartCommunication() void StopCommunication()

RegisterCommEvent

このメソッドは、すべての通信リンク接続/接続解除のイベントの通知を受け取るための オブジェクトを登録します。

プロトタイプ

void RegisterCommEvent()

パラメーター

なし

戻り値

なし

例

例については、イベント処理の例 (ページ 382)を参照してください。

UnregisterCommEvent

通信リンク・イベント処理を終了します。

プロトタイプ

void UnregisterCommEvent()

パラメーター

なし

戻り値

なし

SetConnectionByName

このメソッドは、接続名を使用して、新しく作成された autECLWinMetrics オブジェクトの接続を設定します。Z and I Emulator for Windows では、この接続名は、短い ID (文字 A から Z、または a から z) です。Z and I Emulator for Windows 接続でオープンできるのは、1 つの名前につき 1 つしかありません。例えば、一度に 1 つの接続 A のみをオープンできます。

注: autECLSession 内で autECLWinMetrics オブジェクトを使用している場合は、 これを呼び出さないでください。

プロトタイプ

void SetConnectionByName (String Name)

パラメーター

String 型、Name 型

1 文字の接続のストリング短縮名 (A から Z、または a から z)。

戻り値

なし

例

以下の例は、接続名を使用して、新しく作成された autECLWinMetrics オブジェクトの接続を 設定する方法を示します。

```
DIM  autECLWinObj as Object
Set autECLWinObj = CreateObject("ZIEWin.autECLWinMetrics")

' Initialize the connection
autECLWinObj.SetConnectionByName("A")
' For example, set the host window to minimized
autECLWinObj.Minimized = True
```

SetConnectionByHandle

このメソッドは、接続ハンドルを使用して、新しく作成された autECLWinMetrics オブジェクトの接続を設定します。Z and I Emulator for Windows では、この接続ハンドルは Long integer です。特定の 1 つのハンドルに対して 1 つしか、Z and I Emulator for Windows 接続をオープンできません。例えば、一度に 1 つの接続 A のみをオープンできます。

注: autECLSession 内で autECLWinMetrics オブジェクトを使用している場合は、 これを呼び出さないでください。

プロトタイプ

void SetConnectionByHandle(Long Handle)

パラメーター

Long Handle

オブジェクトに設定される接続の Long integer 値。

戻り値

なし

例

以下の例は、接続ハンドルを使用して、新しく作成された autECLWinMetrics オブジェクトの 接続を設定する方法を示します。

```
DIM  autECLWinObj as Object
DIM  ConnList as Object
Set autECLWinObj = CreateObject("ZIEWin.autECLWinMetrics")
Set ConnList = CreateObject("ZIEWin.autECLConnList")

' Initialize the connection
ConnList.Refresh
autECLWinObj.SetConnectionByHandle(ConnList(1).Handle)
' For example, set the host window to minimized
autECLWinObj.Minimized = True
```

GetWindowRect

GetWindowRect メソッドは、エミュレーター・ウィンドウ長方形の境界点を戻します。

プロトタイプ

void GetWindowRect(Variant Left, Variant Top, Variant Right, Variant Bottom)

パラメーター

Variant Left, Top, Right, Bottom

エミュレーター・ウィンドウの境界点。

戻り値

なし

例

以下の例は、エミュレーター・ウィンドウ長方形の境界点を戻す方法を示します。

```
Dim  autECLWinObj as Object
Dim  ConnList as Object
Dim  left
```

```
Dim  top
Dim  right
Dim  bottom
Set autECLWinObj = CreateObject("ZIEWin.autECLWinMetrics")
Set ConnList = CreateObject("ZIEWin.autECLConnList")

' Initialize the connection
ConnList.Refresh
autECLWinObj.SetConnectionByHandle(ConnList(1).Handle)
autECLWinObj.GetWindowRect left, top, right, bottom
```

SetWindowRect

SetWindowRect メソッドは、エミュレーター・ウィンドウ長方形の境界点を設定します。

プロトタイプ

void SetWindowRect(Long Left, Long Top, Long Right, Long Bottom)

パラメーター

Long Left, Top, Right, Bottom

エミュレーター・ウィンドウの境界点。

戻り値

なし

例

以下の例は、エミュレーター・ウィンドウ長方形の境界点を設定する方法を示します。

```
Dim  autECLWinObj as Object
Dim  ConnList as Object
Set autECLWinObj = CreateObject("ZIEWin.autECLWinMetrics")
Set ConnList = CreateObject("ZIEWin.autECLConnList")

' Initialize the connection
ConnList.Refresh
autECLWinObj.SetConnectionByHandle(ConnList(1).Handle)
autECLWinObj.SetWindowRect 0, 0, 6081, 6081
```

StartCommunication

StartCommunication 集合要素メソッドは、ホスト・データ・ストリームに ZIEWin エミュレーターを接続します。これは、ZIEWin エミュレーター**「通信」**メニューを表示して**「接続」**を選択した場合と同じ結果になります。

プロトタイプ

void StartCommunication()

パラメーター

なし

戻り値

なし

例

以下の例は、ZIEWin エミュレーター・セッションをホストへ接続する方法を示しています。

```
Dim WinObj as Object
Dim autECLConnList as Object

Set autECLConnList = CreateObject("ZIEWin.autECLConnList")
Set WinObj = CreateObject("ZIEWin.autECLWinMetrics")

' Initialize the session
autECLConnList.Refresh
WinObj.SetConnectionByHandle(autECLConnList(1).Handle)

WinObj.StartCommunication()
```

StopCommunication

StopCommunication 集合要素メソッドは、ホスト・データ・ストリームから ZIEWin エミュレーターを切断します。これは、ZIEWin エミュレーター**「通信」**メニューを表示して**「切断」**を選択した場合と同じ結果になります。

プロトタイプ

void StopCommunication()

パラメーター

なし

戻り値

なし

例

以下の例は、ZIEWin エミュレーター・セッションをホストへ接続する方法を示しています。

```
Dim WinObj as Object
Dim autECLConnList as Object

Set autECLConnList = CreateObject("ZIEWin.autECLConnList")
Set WinObj = CreateObject("ZIEWin.autECLWinMetrics")

' Initialize the session
autECLConnList.Refresh
WinObj.SetConnectionByHandle(autECLConnList(1).Handle)

WinObj.StopCommunication()
```

autECL WinMetrics イベント

以下のイベントは、autECL WinMetrics に有効です。

void NotifyCommEvent(boolean bConnected) NotifyCommError() void NotifyCommStop(Long Reason)

NotifyCommEvent

指定された通信リンクは、接続あるいは接続解除されています。

プロトタイプ

void NotifyCommEvent(boolean bConnected)

パラメーター

boolean bConnected

通信リンクが現在接続されている場合は True で、これ以外の場合は False。

例

例については、イベント処理の例 (ページ 382)を参照してください。

NotifyCommError

このイベントは、イベント処理でエラーが発生したときに起こります。

プロトタイプ

NotifyCommError()

パラメーター

なし

例

例については、イベント処理の例 (ページ 382)を参照してください。

NotifyCommStop

このイベントは、イベント処理が停止したときに起こります。

プロトタイプ

void NotifyCommStop(Long Reason)

パラメーター

Long Reason

停止の理由コード。現在は、これは常に 0 です。

イベント処理の例

以下は、WinMetrics イベントの実装方法についての簡単な例です。

```
Option Explicit
Private WithEvents mWmet As autECLWinMetrics   'AutWinMetrics added as reference

sub main()
   'Create Objects
   Set mWmet = New autECLWinMetrics
   mWmet.SetConnectionByName "A"  'Monitor Session A

   mWmet.RegisterCommEvent ' register for Communications Link updates for session A

   ' Display your form or whatever here  (this should be a blocking call, otherwise sub just ends
   call DisplayGUI()

   mWmet.UnregisterCommEvent

   set mWmet = Nothing
End Sub

'This sub will get called when the Communication Link Status of the registered
'connection changes
Private Sub mWmet _NotifyCommEvent()
    ' do your processing here
End Sub

'This event occurs if an error happens in Communications Link event processing
Private Sub mWmet _NotifyCommError()
   'Do any error processing here
End Sub

'This event occurs when Communications Status Notification ends
```

```
Private Sub mWmet _NotifyCommStop()
   'Do any stop processing here
End Sub
```

autECLXfer クラス

autECLXfer オブジェクト は、ファイル転送サービスを提供します。レジストリーでのその名前は ZIEWin.autECLXfer です。

最初に、作成したオブジェクトの接続を設定しなければなりません。SetConnectionByName または SetConnectionByHandle を使用して、オブジェクトを初期化します。接続は一度しか設定できません。接続が設定された後は、SetConnection メソッドをさらに呼び出すと例外を引き起こします。また、接続を設定せずに autECLXfer プロパティーまたはメソッドにアクセス しようとしても、例外が引き起こされます。以下の例は、Visual Basic で autECLXfer オブジェクトを作成し設定する方法を示しています。

```
DIM  XferObj as Object

Set XferObj = CreateObject("ZIEWin.autECLXfer")

' Initialize the connection
XferObj.SetConnectionByName("A")
```

プロパティー

このセクションでは、autECLXfer オブジェクトのプロパティーを説明します。

タイプ	名前	属性
ストリング	名前	読み取り専用
長形式	ハンドル	読み取り専用
ストリング	ConnType	読み取り専用
長形式	CodePage	読み取り専用
ブール値	開始済み	読み取り専用
ブール値	CommStarted	読み取り専用
ブール値	APIEnabled	読み取り専用
ブール値	作動可能	読み取り専用

名前

このプロパティーは、autECLXfer が設定された接続の接続名ストリングです。Z and I Emulator for Windows は、短い文字 ID (A から Z、または a から z) のみをストリングで戻します。Z and I Emulator for Windows 接続でオープンできるのは、1 つの名前につき 1 つしかありません。例えば、一度に 1 つの接続 A のみをオープンできます。Name は、String データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM  Name as String
DIM  Obj as Object
Set Obj = CreateObject("ZIEWin.autECLXfer")
```

```
' Initialize the connection
Obj.SetConnectionByName("A")

' Save the name
Name = Obj.Name
```

ハンドル

これは、autECLXfer オブジェクトが 設定された接続のハンドルです。特定の 1 つのハンドルに対して 1 つしか、Z and I Emulator for Windows 接続をオープンできません。例えば、一度に 1 つの接続 A のみをオープンできます。Handle は、Long データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM  Obj as Object
Set Obj = CreateObject("ZIEWin.autECLXfer")

' Initialize the connection
Obj.SetConnectionByName("A")

' Save the handle
Hand = Obj.Handle
```

ConnType

これは、autECLXfer が設定された接続タイプです。このタイプは、時間の経過とともに変更する場合が あります。ConnType は、String データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM  Type as String
DIM  Obj as Object

Set Obj = CreateObject("ZIEWin.autECLXfer")

' Initialize the connection
Obj.SetConnectionByName("A")
' Save the type
Type = Obj.ConnType
```

ConnType プロパティーの接続タイプは、以下のとおりです。

戻されるストリング	**意味**
DISP3270	3270 表示装置
DISP5250	5250 ディスプレイ
PRNT3270	3270 印刷装置
PRNT5250	5250 プリンター
ASCII	VT エミュレーション

CodePage

これは、autECLXfer が設定された接続のコード・ページです。このコード・ページは、時間の経過とともに変更される場合があります。CodePage は、Long データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM  CodePage as Long
DIM  Obj as Object
Set Obj = CreateObject("ZIEWin.autECLXfer")

' Initialize the connection
Obj.SetConnectionByName("A")
' Save the code page
CodePage = Obj.CodePage
```

開始済み

これは、エミュレーター・ウィンドウが開始されたかどうかを示します。ウィンドウがオープンしている場合、値は True です。 その他の場合は False です。Started は、Boolean データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM  Hand as Long
DIM  Obj as Object

Set Obj = CreateObject("ZIEWin.autECLXfer")

' Initialize the connection
Obj.SetConnectionByName("A")

' This code segment checks to see if A is started.
' The results are sent to a text box called Result.
If Obj.Started = False Then
  Result.Text = "No"
Else
  Result.Text = "Yes"
End If
```

CommStarted

これは、ホストへの接続の状況を示しています。ホストが接続されている場合、値は True です。 その他の場合は False です。CommStarted は、Boolean データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM  Hand as Long
DIM  Obj as Object

Set Obj = CreateObject("ZIEWin.autECLXfer")

' Initialize the connection
Obj.SetConnectionByName("A")

' This code segment checks to see if communications are connected
```

```
' for A.  The results are sent to a text box called
' CommConn.
If Obj.CommStarted = False Then
    CommConn.Text = "No"
Else
    CommConn.Text = "Yes"
End If
```

APIEnabled

これは、エミュレーターが API 使用可能かどうかを示します。API 設定 (Z and I Emulator for Windows のウィンドウでは、**「ファイル」->「API の設定」**を選択) の状態に応じて、接続が可能な場合と、接続できない場合があります。エミュレーターが使用可能の場合には、True です。その他の場合には、False です。APIEnabled は、Boolean データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM  Hand as Long
DIM  Obj as Object

Set Obj = CreateObject("ZIEWin.autECLXfer")

' Initialize the connection
Obj.SetConnectionByName("A")

' This code segment checks to see if A is API enabled.
' The results are sent to a text box called Result.
If Obj.APIEnabled = False Then
  Result.Text = "No"
Else
  Result.Text = "Yes"
End If
```

作動可能

これは、エミュレーター・ウィンドウが 開始されていて、API 使用可能で、そして接続されているかどうかを示します。このプロパティーは、3 つのすべてのプロパティーを確認します。エミュレーターが準備できている場合には、値は True です。 その他の場合には、False です。Ready は、Boolean データ型で読み取り専用です。以下の例は、このプロパティーを示しています。

```
DIM  Hand as Long
DIM  Obj as Object

Set Obj = CreateObject("ZIEWin.autECLXfer")

' Initialize the connection
Obj.SetConnectionByName("A")

' This code segment checks to see if A is ready.
' The results are sent to a text box called Result.
If Obj.Ready = False Then
  Result.Text = "No"
Else
```

```
  Result.Text = "Yes"
End If
```

autECLXfer メソッド

以下のセクションで、autECLXfer オブジェクトに有効な メソッドを説明します。

void RegisterCommEvent() void UnregisterCommEvent() void SetConnectionByName(String Name) void SetConnectionByHandle(Long Handle) void SendFile(String PCFile, String HostFile, String Options) void ReceiveFile(String PCFile, String HostFile, String Options) void StartCommunication() void StopCommunication()

RegisterCommEvent

このメソッドは、すべての通信リンク接続/接続解除のイベントの通知を受け取るための オブジェクトを登録します。

プロトタイプ

void RegisterCommEvent()

パラメーター

なし

戻り値

なし

例

例については、『イベント処理の例 (ページ 394)』を参照してください。

UnregisterCommEvent

通信リンク・イベント処理を終了します。

プロトタイプ

void UnregisterCommEvent()

パラメーター

なし

戻り値

なし

SetConnectionByName

この SetConnectionByName メソッドは、 接続名を使用して、新しく作成された autECLXfer オブジェクトの接続を設定します。Z and I Emulator for Windows では、この接続名は、短い ID (文字 A から Z、または a から z) です。Z and I Emulator for Windows 接続でオープンできるのは、1 つの名前につき 1 つしかありません。例えば、一度に 1 つの接続 A のみをオープンできます。

注: autECLSession 内で autECLXfer オブジェクトを使用している場合は、 これを呼び出さないでください。

プロトタイプ

void SetConnectionByName (String Name)

パラメーター

String 型、Name 型

1 文字の接続のストリング短縮名 (A から Z、または a から z)。

戻り値

なし

例

以下の例は、接続名を使用して、新しく作成された autECLXfer オブジェクトの接続を 設定する方法を示します。

```
DIM  XferObj as Object

Set XferObj = CreateObject("ZIEWin.autECLXfer")

' Initialize the connection
XferObj.SetConnectionByName("A")
```

SetConnectionByHandle

この SetConnectionByHandle メソッドは、 接続ハンドルを使用して、新しく作成された autECLXfer オブジェクトの接続を設定します。Z and I Emulator for Windows では、この接続ハンドルは Long integer です。特定の 1 つのハンドルに対して 1 つしか、Z and I Emulator for Windows 接続をオープンできません。例えば、一度に 1 つの接続 A のみをオープンできます。

注: autECLSession 内で autECLXfer オブジェクトを使用している場合は、 これを呼び出さないでください。

プロトタイプ

void SetConnectionByHandle(Long Handle)

パラメーター

Long Handle

オブジェクトに設定される接続の Long integer 値。

戻り値

なし

例

以下の例は、接続ハンドルを使用して、新しく作成された autECLXfer オブジェクトの接続を 設定する方法を示します。

```
DIM  XferObj as Object
DIM  autECLConnList as Object

Set XferObj = CreateObject("ZIEWin.autECLXfer")
Set autECLConnList = CreateObject("ZIEWin.autECLConnList")

' Initialize the connection with the first connection in the list
autECLConnList.Refresh
XferObj.SetConnectionByHandle(autECLConnList(1).Handle)
```

SendFile

SendFile メソッドは、autECLXfer オブジェクトに関連する 接続について、ワークステーションからホストへファイルを送信します。

プロトタイプ

void SendFile(String PCFile, String HostFile, String Options)

パラメーター

String PCFile

ワークステーション上のファイルの名前。

String HostFile

ホスト上のファイルの名前。

String Options (ストリング・オプション)

ホストに依存する転送オプション。詳しくは、使用上の注意 (ページ 390)を参照してください。

戻り値

なし

使用上の注意

ファイル転送オプションは、ホストに応じて異なります。以下に示すのは、VM/CMS ホストの有効なホスト・オプションの一部です。

ASCII
CRLF
APPEND
LRECL
RECFM
CLEAR/NOCLEAR
PROGRESS
QUIET

サポートされるホストとそれに関連したファイル転送オプションのリストは、「エミュレーター・プログラミング」を参照してください。

例

以下の例は、autECLXfer オブジェクトに関連する接続について、 ワークステーションからホストへファイルを送信する方法を示しています。

```
DIM  XferObj as Object
DIM  autECLConnList as Object
DIM  NumRows as Long

Set XferObj = CreateObject("ZIEWin.autECLXfer")
Set autECLConnList = CreateObject("ZIEWin.autECLConnList")

' Initialize the connection with the first connection in the autECLConnList
autECLConnList.Refresh
XferObj.SetConnectionByHandle(autECLConnList(1).Handle)

' For example, send the file to VM
XferObj.SendFile "c:\windows\temp\thefile.txt",
                     "THEFILE TEXT A0",
                     "CRLF ASCII"
```

ReceiveFile

ReceiveFile メソッドは、autECLXfer オブジェクト に関連する接続について、ホストからワークステーションへファイルを受信します。

プロトタイプ

void ReceiveFile(String PCFile, String HostFile, String Options)

パラメーター

String PCFile

ワークステーション上のファイルの名前。

String HostFile

ホスト上のファイルの名前。

String Options (ストリング・オプション)

ホストに依存する転送オプション。詳しくは、使用上の注意 (ページ 391)を参照してください。

戻り値

なし

使用上の注意

ファイル転送オプションは、ホストに応じて異なります。以下に示すのは、VM/CMS ホストの有効なホスト・オプションの一部です。

ASCII
CRLF
APPEND
LRECL
RECFM
CLEAR/NOCLEAR
PROGRESS
QUIET

サポートされるホストとそれに関連したファイル転送オプションのリストは、「エミュレーター・プログラミング」資料を参照してください。

例

以下の例は、autECLXfer オブジェクトに関連した接続について、 ホストからファイルを受け取り、それをワークステーションに送信する方法を示しています。

```
DIM  XferObj as Object
DIM  autECLConnList as Object
DIM  NumRows as Long

Set XferObj = CreateObject("ZIEWin.autECLXfer")
Set autECLConnList = CreateObject("ZIEWin.autECLConnList")

' Initialize the connection with the first connection in the list
autECLConnList.Refresh
XferObj.SetConnectionByHandle(autECLConnList(1).Handle)
' For example, send the file to VM
XferObj.ReceiveFile "c:\windows\temp\thefile.txt",
                      "THEFILE TEXT A0",
                      "CRLF ASCII"
```

StartCommunication

StartCommunication 集合要素メソッドは、ホスト・データ・ストリームに ZIEWin エミュレーターを接続します。これは、ZIEWin エミュレーター**「通信」**メニューを表示して**「接続」**を選択した場合と同じ結果になります。

プロトタイプ

void StartCommunication()

パラメーター

なし

戻り値

なし

例

以下の例は、ZIEWin エミュレーター・セッションをホストへ接続する方法を示しています。

```
Dim XObj as Object
Dim autECLConnList as Object

Set autECLConnList = CreateObject("ZIEWin.autECLConnList")
Set XObj = CreateObject("ZIEWin.autECLXfer")

' Initialize the session
autECLConnList.Refresh
XObj.SetConnectionByHandle(autECLConnList(1).Handle)

XObj.StartCommunication()
```

StopCommunication

StopCommunication 集合要素メソッドは、ホスト・データ・ストリームから ZIEWin エミュレーターを切断します。これは、ZIEWin エミュレーター**「通信」**メニューを表示して**「切断」**を選択した場合と同じ結果になります。

プロトタイプ

void StopCommunication()

パラメーター

なし

戻り値

なし

例

以下の例は、ZIEWin エミュレーター・セッションをホストへ接続する方法を示しています。

```
Dim XObj as Object
Dim autECLConnList as Object

Set autECLConnList = CreateObject("ZIEWin.autECLConnList")
Set XObj = CreateObject("ZIEWin.autECLXfer")

' Initialize the session
autECLConnList.Refresh
XObj.SetConnectionByHandle(autECLConnList(1).Handle)

SessObj.StopCommunication()
```

autECLXfer イベント

以下のイベントは、autECLXfer に有効です。

void NotifyCommEvent(boolean bConnected) NotifyCommError() void NotifyCommStop(Long Reason)

NotifyCommEvent

指定された通信リンクは、接続あるいは接続解除されています。

プロトタイプ

void NotifyCommEvent(boolean bConnected)

パラメーター

boolean bConnected

通信リンクが現在接続されている場合は True で、これ以外の場合は False。

例

例については、イベント処理の例 (ページ 394)を参照してください。

NotifyCommError

このイベントは、イベント処理でエラーが発生したときに起こります。

プロトタイプ

NotifyCommError()

パラメーター

なし

例

例については、イベント処理の例 (ページ 394)を参照してください。

NotifyCommStop

このイベントは、イベント処理が停止したときに起こります。

プロトタイプ

void NotifyCommStop(Long Reason)

パラメーター

Long Reason

停止の理由コード。現在は、これは常に 0 です。

イベント処理の例

以下は、Xfer イベントの実装方法についての簡単な例です。

```
Option Explicit
Private WithEvents mXfer As autECLXfer 'AutXfer added as reference
```

```
sub main()
'Create Objects
Set mXfer = New autECLXfer
mXfer.SetConnectionByName "A" 'Monitor Session A

mXfer.RegisterCommEvent ' register for Communications Link updates for session A

' Display your form or whatever here  (this should be a blocking call, otherwise sub just ends
call DisplayGUI()

mXfer.UnregisterCommEvent

set mXfer= Nothing
End Sub

'This sub will get called when the Communication Link Status of the registered
'connection changes
Private Sub mXfer _NotifyCommEvent()
' do your processing here
End Sub

'This event occurs if an error happens in Communications Link event processing
Private Sub mXfer _NotifyCommError()
'Do any error processing here
End Sub

'This event occurs when Communications Status Notification ends
Private Sub mXfer _NotifyCommStop()
'Do any stop processing here
End Sub
```

autSystem クラス

autSystem クラスは、 プログラム言語によっては存在しないユーティリティー操作を実行するために使用されます。

autSystem メソッド

以下のセクションで、autSystem オブジェクトに 有効なメソッドを説明します。

Long Shell(VARIANT ExeName, VARIANT Parameters, VARIANT WindowStyle) String Inputnd()

Shell

shell 関数は、実行可能ファイルを実行します。

プロトタイプ

Long Shell(VARIANT ExeName, VARIANT Parameters, VARIANT WindowStyle)

パラメーター

VARIANT ExeName

実行可能ファイルの絶対パスおよびファイル名。

VARIANT Parameters

実行可能ファイルに渡す任意のパラメーター。このパラメーターはオプションです。

VARIANT WindowStyle

実行可能であることを示すための初期ウィンドウ・スタイル。このパラメーターはオプションで、以下のような値が可能です。

1. フォーカスあり、通常 (デフォルト)
2. フォーカスあり、最小化
3. 最大化
4. フォーカスなしの通常ウィンドウ
5. フォーカスなしのアイコン

戻り値

メソッドは、正常終了の場合はプロセス ID を、失敗した場合はゼロを戻します。

例

```
Example autSystem - Shell()

'This example starts notepad with the file c:\test.txt loaded
dim ProcessID
dim SysObj as object

set SysObj = CreateObject("ZIEWin.autSystem")
ProcessID = SysObj.shell "Notepad.exe","C:\test.txt"
If ProcessID > 0 then
 Msgbox "Notepad Started, ProcessID = " + ProcessID
Else
 Msgbox "Notepad not started"
End if
```

Inputnd

Inputnd メソッドは、ユーザーに対して表示しないテキスト枠を使用してポップアップ入力ボックスを表示します。したがって、ユーザーがデータを入力したとき、アスタリスク (*) のみが表示されます。

プロトタイプ

String Inputnd()

パラメーター

なし

戻り値

入力ボックスに入力された文字、また、何も入力されていない場合は、"" が戻されます。

例

```
DIM strPassWord
dim SysObj as Object
dim PSObj as Object

set SysObj = CreateObject("ZIEWin.autSystem")
set PSObj = CreateObject("ZIEWin.autPS")

PSObj.SetConnectionByName("A")
'Prompt user for password
strPassWord = SysObj.Inputnd()
PSObj.SetText(strPasssWord)
DIM  XferObj as Object

Set XferObj = CreateObject("ZIEWin.autECLXfer")

' Initialize the connection
XferObj.SetConnectionByName("A")
```

autECLPageSettings クラス

autECLPageSettings オブジェクトは、Z and I Emulator for Windows接続のページ設定を制御します。レジストリーでのその名前は ZIEWin.autECLPageSettings です。この自動化オブジェクトは VB スクリプトでも使用できます。

読み取り専用プロパティー autECLPageSettings は autECLSession オブジェクトに追加されています。このプロパティーの使用方法については、autECLSession クラス (ページ 353)を参照してください。

注: autECLSession オブジェクト内の autECLPageSettings オブジェクトは、autECLSession オブジェクトにより設定されます。

以下の例は、Visual Basic で autECLPageSettings オブジェクトを作成し、 設定する方法を示しています。

```
DIM PgSet as Object
Set PgSet = CreateObject("ZIEWin.autECLPageSettings")
PgSet.SetConnectionByName("A")
```

使用上の注意

最初に、作成したオブジェクトの接続を設定しなければなりません。SetConnectionByName または SetConnectionByHandle を使用して、オブジェクトを初期化します。接続は一度しか設定できません。接続が設定された後は、接続設定メソッドをさらに呼び出すと例外を引き起こします。接続を設定せずにプロパティーまたはメソッドにアクセスしようとすると、 例外が引き起こされます。

プロパティー CPI、LPI、および FontSize はプロパティー FaceName によって決定されます。そのため、FaceName を設定する前に CPI、LPI、または FontSize を設定し、 それらが新しい FaceName に対して有効ではない場合、異なる CPI、LPI、または FontSize 値が接続で再構成される場合があります。CPI、LPI、または FontSize を設定する前に FaceName を設定する必要があります。そうしなかった場合には、FaceName を設定するたびに CPI、LPI、および FontSize を照会して、それらが希望する値であることを確認してください。

制約事項

各メソッドに関連する接続は、メソッドを成功させるために 特定の状態になければなりません。制約事項を満たしていないと、 該当する例外が引き起こされます。

autECLPageSettings オブジェクトのいずれかのプロパティーまたはメソッドが呼び出されるときには、 以下の制約事項を満たしている必要があります。

- この API が呼び出されるときにホスト・セッションが印刷中ではない。
- **「ファイル」→「ページ設定」**および**「ファイル」→「プリンター設定」**ダイアログが使用中ではない。
- 関連する接続が PDT モードではない。

特定のプロパティーまたはメソッドに対して追加の制約事項が適用される場合があります。

接続タイプ

autECLPageSettings クラス内のメソッドには、以下の接続タイプが有効です。

- 3270 表示装置
- 3270 印刷装置
- 5250 ディスプレイ
- VT (ASCII)

プロパティーまたはメソッドが、サポートされない接続上でアクセスされたり、呼び出されたりすると、 例外が引き起こされます。ConnType プロパティーを使用して接続タイプを判別してください。

プロパティー

このセクションでは、autECLPageSettings オブジェクトのプロパティーを説明します。

タイプ	**名前**	**属性**
長形式	CPI	Read/Write

タイプ	名前	属性
ブール値	FontCPI	読み取り専用
長形式	LPI	Read/Write
ブール値	FontLPI	読み取り専用
ストリング	FaceName	Read/Write
長形式	FontSize	Read/Write
長形式	MaxLinesPerPage	Read/Write
長形式	MaxCharsPerLine	Read/Write
ストリング	名前	読み取り専用
長形式	ハンドル	読み取り専用
ストリング	ConnType	読み取り専用
長形式	CodePage	読み取り専用
ブール値	開始済み	読み取り専用
ブール値	CommStarted	読み取り専用
ブール値	APIEnabled	読み取り専用
ブール値	作動可能	読み取り専用

CPI

このプロパティーは、1 インチ当たりに印刷される文字数を決定します。これは Long データ型で、読み取り/書き込み可能です。

このプロパティーを事前定義の定数 pcFontCPI に設定して、 「ページ設定 (Page Settings)」の「フォント CPI」を選択するか、特定の CPI 値に設定します。接続に FontCPI が構成されているときにこのプロパティーが照会されると、 実際の CPI 値が戻され、定数 pcFontCPI は戻されません。

FontCPI が接続に設定されているかどうかを判別するには、プロパティー FontCPI を使用します。

例

```
Dim PgSet as Object
Dim ConnList as Object
Dim CPI as Long

Set PgSet = CreateObject("ZIEWin.autECLPageSettings")
Set ConnList = CreateObject("ZIEWin.autECLConnList")
' Initialize the connection
ConnList.Refresh
PgSet.SetConnectionByHandle(ConnList(1).Handle)

CPI = PgSet.CPI ' get the CPI value
' or...
PgSet.CPI = pcFontCPI 'set the connection to use Font CPI.
```

FontCPI

これは、接続に「フォント CPI」が設定されているかどうかを判別します。FontCPI は、Boolean データ型で読み取り専用です。

例

```
Dim PgSet as Object
Dim ConnList as Object

Set PgSet = CreateObject("ZIEWin.autECLPageSettings")
Set ConnList = CreateObject("ZIEWin.autECLConnList")
' Initialize the connection
ConnList.Refresh
PgSet.SetConnectionByHandle(ConnList(1).Handle)

'check if Font CPI is set
If PgSet.FontCPI Then
 ...
```

LPI

このプロパティーは、1 インチ当たりに印刷される行数を決定します。これは Long データ型で、読み取り/書き込み可能です。このプロパティーを事前定義の定数 pcFontLPI に設定して、 「ページ設定」の「フォント LPI」を選択するか、特定の LPI 値に設定します。接続に FontLPI が構成されているときにこのプロパティーが照会されると、 実際の LPI 値が戻され、定数 pcFontLPI は戻されません。FontLPI が接続に設定されているかどうかを判別するには、プロパティー FontLPI を使用します。

例

```
Dim PgSet as Object
Dim ConnList as Object
Dim LPI as Long

Set PgSet = CreateObject("ZIEWin.autECLPageSettings")
Set ConnList = CreateObject("ZIEWin.autECLConnList")
' Initialize the connection
ConnList.Refresh
PgSet.SetConnectionByHandle(ConnList(1).Handle)

LPI = PgSet.LPI ' get the LPI value
' or...
PgSet.LPI = pcFontLPI 'set the connection to use Font LPI.
```

FontLPI

このプロパティーは、接続に「フォント LPI」が設定されているかどうかを判別します。FontLPI は、Boolean データ型で読み取り専用です。

例

```
Dim PgSet as Object
Dim ConnList as Object

Set PgSet = CreateObject("ZIEWin.autECLPageSettings")
Set ConnList = CreateObject("ZIEWin.autECLConnList")
' Initialize the connection
ConnList.Refresh
PgSet.SetConnectionByHandle(ConnList(1).Handle)

'check if Font LPI is set
If PgSet.FontLPI Then
 ...
```

FaceName

これは、接続の「ページ設定」の「フォント書体名」です。FaceName は、String データ型で読み取り/書き込み可能です。

例

```
Dim PgSet as Object
Dim ConnList as Object
Dim FaceName as String

Set PgSet = CreateObject("ZIEWin.autECLPageSettings")
Set ConnList = CreateObject("ZIEWin.autECLConnList")
' Initialize the connection
ConnList.Refresh
PgSet.SetConnectionByHandle(ConnList(1).Handle)
FaceName = PgSet.FaceName ' get the FaceName
' or...
PgSet.FaceName = "Courier New" 'set the FaceName
```

MaxLinesPerPage

このプロパティーは、1 ページに印刷可能な最大行数です。これは、最大印刷行数 または MPL とも呼ばれます。有効な値は 1 から 255 の範囲内です。これは Long データ型で、読み取り/書き込み可能です。

例

```
Dim PgSet as Object
Dim ConnList as Object
Dim MPL as Long

Set PgSet = CreateObject("ZIEWin.autECLPageSettings")
Set ConnList = CreateObject("ZIEWin.autECLConnList")
' Initialize the connection
ConnList.Refresh
```

```
PgSet.SetConnectionByHandle(ConnList(1).Handle)

MPL = PgSet.MaxLinesPerPage ' get the MaxLinesPerPage
' or...
PgSet.MaxLinesPerPage = 20 'set the MaxLinesPerPage
```

MaxCharsPerLine

このプロパティーは、1 行に印刷可能な最大文字数です。これは、最大印刷位置 または MPP とも呼ばれます。有効な値は 1 から 255 の範囲内です。これは Long データ型で、読み取り/書き込み可能です。

例

```
Dim PgSet as Object
Dim ConnList as Object
Dim MPP as Long

Set PgSet = CreateObject("ZIEWin.autECLPageSettings")
Set ConnList = CreateObject("ZIEWin.autECLConnList")
' Initialize the connection
ConnList.Refresh
PgSet.SetConnectionByHandle(ConnList(1).Handle)

MPP = PgSet.MaxCharsPerLine ' get the MaxCharsPerLine
' or...
PgSet.MaxCharsPerLine = 80 'set the MaxCharsPerLine
```

名前

このプロパティーは、autECLPageSettings が設定された接続の接続名ストリングです。Z and I Emulator for Windows は、短い文字 ID (**A** から **Z** の単一の英字) のみをストリングで戻します。Z and I Emulator for Windows 接続でオープンできるのは、1 つの名前につき 1 つしかありません。例えば、一度に 1 つの接続 **A** のみをオープンできます。Name は、String データ型で読み取り専用です。

例

```
Dim PgSet as Object
Dim ConnList as Object
DIM Name as String

Set PgSet = CreateObject("ZIEWin.autECLPageSettings")
Set ConnList = CreateObject("ZIEWin.autECLConnList")
' Initialize the connection
ConnList.Refresh
PgSet.SetConnectionByHandle(ConnList(1).Handle)

Name = PgSet.Name 'Save the name
```

ハンドル

このプロパティーは、autECLPageSettings オブジェクトが設定された接続のハンドルです。特定の 1 つのハンドルに対して 1 つしか、Z and I Emulator for Windows 接続をオープンできません。例えば、一度に 1 つの接続 **A** のみをオープンできます。Handle は、Long データ型で読み取り専用です。

例

```
Dim PgSet as Object
Dim ConnList as Object
Dim Hand as Long

Set PgSet = CreateObject("ZIEWin.autECLPageSettings")
Set ConnList = CreateObject("ZIEWin.autECLConnList")
' Initialize the connection
ConnList.Refresh
PgSet.SetConnectionByHandle(ConnList(1).Handle)

Hand = PgSet.Handle ' save the handle
```

ConnType

このプロパティーは、autECLPageSettings が設定された接続タイプです。このタイプは、時間の経過とともに変更される場合があります。ConnType は、String データ型で読み取り専用です。

ストリング値	**接続タイプ**
DISP3270	3270 表示装置
DISP5250	5250 ディスプレイ
PRNT3270	3270 印刷装置
PRNT5250	5250 プリンター
ASCII	VT エミュレーション

例

```
Dim PgSet as Object
Dim ConnList as Object
Dim Type as String

Set PgSet = CreateObject("ZIEWin.autECLPageSettings")
Set ConnList = CreateObject("ZIEWin.autECLConnList")
' Initialize the connection
ConnList.Refresh
PgSet.SetConnectionByHandle(ConnList(1).Handle)

Type = PgSet.ConnType ' save the type
```

CodePage

このプロパティーは、autECLPageSettings が設定された接続タイプです。このタイプは、時間の経過とともに変更される場合があります。ConnType は、String データ型で読み取り専用です。

例

```
Dim PgSet as Object
Dim ConnList as Object
Dim CodePage as Long

Set PgSet = CreateObject("ZIEWin.autECLPageSettings")
Set ConnList = CreateObject("ZIEWin.autECLConnList")
' Initialize the connection
ConnList.Refresh
PgSet.SetConnectionByHandle(ConnList(1).Handle)

CodePage = PgSet.CodePage ' save the codepage
```

開始済み

このプロパティーは、エミュレーター・ウィンドウが開始されたかどうかを示します。ウィンドウがオープンしている場合、値は True です。 その他の場合は False です。Started は、Boolean データ型で読み取り専用です。

例

```
Dim PgSet as Object
Dim ConnList as Object

Set PgSet = CreateObject("ZIEWin.autECLPageSettings")
Set ConnList = CreateObject("ZIEWin.autECLConnList")
' Initialize the connection
ConnList.Refresh
PgSet.SetConnectionByHandle(ConnList(1).Handle)

' This code segment checks to see if A is started.
' The results are sent to a text box called Result.
If PgSet.Started = False Then
 Result.Text = "No"
Else
 Result.Text = "Yes"
End If
```

CommStarted

このプロパティーは、ホストへの接続の状況を示しています。ホストが接続されている場合、値は True です。 その他の場合は False です。CommStarted は、Boolean データ型で読み取り専用です。

例

```
Dim PgSet as Object
Dim ConnList as Object

Set PgSet = CreateObject("ZIEWin.autECLPageSettings")
Set ConnList = CreateObject("ZIEWin.autECLConnList")
' Initialize the connection
ConnList.Refresh
PgSet.SetConnectionByHandle(ConnList(1).Handle)

' This code segment checks to see if communications are connected
' for A. The results are sent to a text box called
' CommConn.
If PgSet.CommStarted = False Then
 CommConn.Text = "No"
Else
 CommConn.Text = "Yes"
End If
```

APIEnabled

このプロパティーは、エミュレーターが API 使用可能かどうかを示します。API 設定の状態に応じて、接続を API 使用可能または API 使用不可にすることができます (Z and I Emulator for Windows のウィンドウで**「設定」→「API」**をクリック)。エミュレーターが API 使用可能の場合には、値は True です。 その他の場合には、False です。APIEnabled は、Boolean データ型で読み取り専用です。

例

```
Dim PgSet as Object
Dim ConnList as Object

Set PgSet = CreateObject("ZIEWin.autECLPageSettings")
Set ConnList = CreateObject("ZIEWin.autECLConnList")
' Initialize the connection
ConnList.Refresh
PgSet.SetConnectionByHandle(ConnList(1).Handle)

' This code segment checks to see if A is API-enabled.
' The results are sent to a text box called Result.
If PgSet.APIEnabled = False Then
 Result.Text = "No"
Else
 Result.Text = "Yes"
End If
```

作動可能

このプロパティーは、エミュレーター・ウィンドウが開始されているか、API 使用可能か、接続されているかどうかを示します。このプロパティーは、3 つのすべてのプロパティーを確認します。エミュレーターが準備できている場合には、値は True です。 その他の場合には、False です。Ready は、Boolean データ型で読み取り専用です。

例

```
Dim PgSet as Object
Dim ConnList as Object
Set PgSet = CreateObject("ZIEWin.autECLPageSettings")
Set ConnList = CreateObject("ZIEWin.autECLConnList")
' Initialize the connection
ConnList.Refresh
PgSet.SetConnectionByHandle(ConnList(1).Handle)
' This code segment checks to see if A is ready.
' The results are sent to a text box called Result.
If PgSet.Ready = False Then
 Result.Text = "No"
Else
 Result.Text = "Yes"
End If
```

autECLPageSettings メソッド

以下のセクションで、autECLPageSettings オブジェクトに有効なメソッドを説明します。

void RestoreTextDefaults() void SetConnectionByName (String Name) void SetConnectionByHandle (Long Handle)

RestoreTextDefaults

RestoreTextDefaults メソッドは、接続の「ページ設定」ダイアログの**「テキスト」**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、接続の「ページ設定」ダイアログの**「テキスト」**プロパティー・ページで**「デフォルト」**ボタンを押すことと同じです。

プロトタイプ

void RestoreTextDefaults()

パラメーター

なし

戻り値

なし

例

以下の例で、RestoreTextDefaults メソッドを示します。

```
Dim PgSet as Object
Dim ConnList as Object

Set PgSet = CreateObject("ZIEWin.autECLPageSettings")
Set ConnList = CreateObject("ZIEWin.autECLConnList")
' Initialize the connection
ConnList.Refresh
PgSet.SetConnectionByHandle(ConnList(1).Handle)

PgSet.RestoreTextDefaults 'Restores Text Default Settings
```

SetConnectionByName

この SetConnectionByName メソッドは、接続名を使用して、新しく作成された autECLPageSettings オブジェクトの接続を設定します。この接続名は、短い接続 ID (**A** から **Z** の単一の英字) です。Z and I Emulator for Windows 接続でオープンできるのは、1 つの名前につき 1 つしかありません。例えば、一度に 1 つの接続 **A** のみをオープンできます。

注: autECLSession オブジェクトに含まれる autECLPageSettings オブジェクトを使用している場合には、このメソッドを呼び出さないでください。

プロトタイプ

void SetConnectionByName (String Name)

パラメーター

String 型、Name 型

1 文字の接続のストリング短縮名。有効値は A から Z です。

戻り値

なし

例

以下の例は、接続名を使用して、新しく作成された autECLPageSettings オブジェクトの接続を設定する方法を示します。

```
Dim PgSet as Object
Set PgSet = CreateObject("ZIEWin.autECLPageSettings")
' Initialize the connection
PgSet.SetConnectionByName("A")
' For example, see if Font CPI is set
```

```
If PgSet.FontCPI Then
'your logic here...
End If
```

SetConnectionByHandle

この SetConnectionByHandle メソッドは、接続ハンドルを使用して、新しく作成された autECLPageSettings オブジェクトの接続を設定します。Z and I Emulator for Windows では、この接続ハンドルは Long integer です。特定の 1 つのハンドルに対して 1 つしか、Z and I Emulator for Windows 接続をオープンできません。例えば、一度に 1 つの接続 **A** のみをオープンできます。

注: autECLSession オブジェクトに含まれる autECLPageSettings オブジェクトを使用している場合には、このメソッドを呼び出さないでください。

プロトタイプ

void SetConnectionByHandle(Long Handle)

パラメーター

Long Handle

オブジェクトに設定される接続の Long integer 値。

戻り値

なし

例

以下の例は、接続ハンドルを使用して、新しく作成された autECLPageSettings オブジェクトの接続を設定する方法を示します。

```
Dim PgSet as Object
Dim ConnList as Object

Set PgSet = CreateObject("ZIEWin.autECLPageSettings")
Set ConnList = CreateObject("ZIEWin.autECLConnList")
' Initialize the connection
ConnList.Refresh
PgSet.SetConnectionByHandle(ConnList(1).Handle)

' For example, see if Font CPI is set
If PgSet.FontCPI Then
'your logic here...
End If
```

autECLPrinterSettings クラス

autECLPrinterSettings オブジェクトは、Z and I Emulator for Windows接続のプリンター設定を制御します。レジストリーでのその名前は `ZIEWin.autECLPrinterSettings` です。この自動化オブジェクトは VB スクリプトでも使用できます。

読み取り専用プロパティー **autECLPrinterSettings** が autECLSession オブジェクトに追加されています。このプロパティーの使用方法については、autECLSession クラス (ページ 353)を参照してください。

注: autECLSession オブジェクト内の autECLPrinterSettings オブジェクトは、autECLSession オブジェクトにより設定されます。

以下の例は、Visual Basic で autECLPrinterSettings オブジェクトを作成し、設定する方法を示しています。

```
DIM PrSet as Object
Set PrSet = CreateObject("ZIEWin.autECLPrinterSettings")
PrSet.SetConnectionByName("A")
```

使用上の注意

最初に、作成したオブジェクトの接続を設定しなければなりません。SetConnectionByName または SetConnectionByHandle を使用して、オブジェクトを初期化します。接続は一度しか設定できません。接続が設定された後は、接続設定メソッドをさらに呼び出すと例外を引き起こします。接続を設定せずにプロパティーまたはメソッドにアクセスしようとすると、 例外が引き起こされます。

プロパティー CPI、LPI、および FontSize はプロパティー FaceName によって決定されます。そのため、FaceName を設定する前に CPI、LPI、または FontSize を設定し、 それらが新しい FaceName に対して有効ではない場合、異なる CPI、LPI、または FontSize 値が接続で再構成される場合があります。CPI、LPI、または FontSize を設定する前に FaceName を設定する必要があります。そうしなかった場合には、FaceName を設定するたびに CPI、LPI、および FontSize を照会して、それらが希望する値であることを確認してください。

制約事項

各メソッドに関連する接続は、メソッドを成功させるために 特定の状態になければなりません。制約事項を満たしていないと、 該当する例外が引き起こされます。

autECLPageSettings オブジェクトのいずれかのプロパティーまたはメソッドが呼び出されるときには、 以下の制約事項を満たしている必要があります。

- この API が呼び出されるときにホスト・セッションが印刷中ではない。
- **「ファイル」→「ページ設定」**および**「ファイル」→「プリンター設定」**ダイアログが使用中ではない。

特定のプロパティーまたはメソッドに対して追加の制約事項が適用される場合があります。

プロパティー

このセクションでは、autECLPrinterSettings オブジェクトのプロパティーを説明します。

タイプ	名前	属性
ブール値	PDTMode	読み取り専用
ストリング	PDTFile	読み取り専用
長形式	PrintMode	読み取り専用
ストリング	プリンター	読み取り専用
ストリング	PrtToDskAppendFile	読み取り専用
ストリング	PrtToDskSeparateFile	読み取り専用
ブール値	PrompDialogOption	Read/Write
ストリング	名前	読み取り専用
長形式	ハンドル	読み取り専用
ストリング	ConnType	読み取り専用
ブール値	CodePage	読み取り専用
ブール値	開始済み	読み取り専用
ブール値	CommStarted	読み取り専用
ブール値	APIEnabled	読み取り専用
ブール値	作動可能	読み取り専用

PDTMode

このプロパティーは、接続が PDT モードであるかどうかを判別します。PDTMode は、Boolean データ型で、読み取り/書き込み可能です。

例

```
Dim PrSet as Object
Dim ConnList as Object

Set PrSet = CreateObject("ZIEWin.autECLPrinterSettings")
Set ConnList = CreateObject("ZIEWin.autECLConnList")
' Initialize the connection
ConnList.Refresh
PrSet.SetConnectionByHandle(ConnList(1).Handle)

'check if in PDT mode.
If PrSet.PDTMode Then
 ...
```

PDTFile

このプロパティーは、接続に PDT ファイルが構成されているかどうかを示します。このプロパティーは、接続に PDT ファイルが構成されていない場合にはヌル・ストリングを示します。それ以外の場合、このプロパティーは PDT ファイルの完全修飾パス名を示します。PDTFile は、String データ型で読み取り専用です。

例

```
Dim PrSet as Object
Dim ConnList as Object

Set PrSet = CreateObject("ZIEWin.autECLPrinterSettings")
Set ConnList = CreateObject("ZIEWin.autECLConnList")
' Initialize the connection
ConnList.Refresh
PrSet.SetConnectionByHandle(ConnList(1).Handle)

If PrSet. PDTFile = vbNullString Then ' get the
  ...
Else
  ...
```

PrintMode

このプロパティーは、接続の印刷モードを示します。PrintMode は、Long データ型で読み取り専用です。このプロパティーは、次の 4 つの列挙型値のいずれかを戻します。

値	列挙型定数の名前	説明
1	pcPrtToDskAppend	**ディスクへの印刷 - コピー追加**モード。これは、接続の「プリンター設定」ダイアログの**「プリンター」**リスト・ボックスで**「ディスクへの印刷」**→**「コピー追加」**オプションが選択されていることを意味します。
2	pcPrtToDskSeparate	**ディスクへの印刷 - 別個**モード。これは、接続の「プリンター設定」ダイアログの**「プリンター」**リスト・ボックスで **「ディスクへの印刷」**→**「別個」**オプションが選択されていることを意味します。
3	pcSpecificPrinter	**特定のプリンター**・モード。これは、接続の「プリンター設定」ダイアログの**「プリンター」**リスト・ボックスでいずれかのプリンターが選択されて、**「Windows のデフォルト・プリンターを使用」**チェック・ボックスがクリアされていることを意味します。
4	pcWinDefaultPrinter	**Windows® のデフォルト・プリンター**・モード。これは、**「Windows のデフォルト・プリンターを使用」**チェック・ボックスが選択されていることを意味します。

例

```
Dim PrSet as Object
Dim ConnList as Object
```

```
Set PrSet = CreateObject("ZIEWin.autECLPrinterSettings")
Set ConnList = CreateObject("ZIEWin.autECLConnList")
' Initialize the connection
ConnList.Refresh
PrSet.SetConnectionByHandle(ConnList(1).Handle)

If PrSet.PrintMode = pcPrtToDskAppend Then
  ...
ElseIf PrSet.PrintMode = pcPrtToDskSeparate Then
  ...
ElseIf PrSet.PrintMode = pcSpecificPrinter Then
  ...
ElseIf PrSet.PrintMode = pcWinDefaultPrinter Then
  ...
```

プリンター

このプロパティーは、プリンターの名前です。これには、以下のいずれかが入ります。

- 接続の PrintMode が pcSpecificPrinter の場合、特定のプリンターの名前。
- 接続の PrintMode が pcWinDefaultPrinter の場合、Windows のデフォルト・プリンターの名前。
- 接続にプリンターが構成されていない場合、あるいは接続の PrintMode が pcPrtToDskAppend または pcPrtToDskSeparate の場合、ヌル・ストリング。

Printer は、String データ型で読み取り専用です。

値の形式は次のようにする必要があります。

```
<Printer name> on <Port Name>
```

例:

- `HP LaserJet 4050 Series PCL 6 on LPT1`

例

```
Dim PrSet as Object
Dim ConnList as Object
Dim Printer as String

Set PrSet = CreateObject("ZIEWin.autECLPrinterSettings")
Set ConnList = CreateObject("ZIEWin.autECLConnList")
' Initialize the connection
ConnList.Refresh
PrSet.SetConnectionByHandle(ConnList(1).Handle)

Printer = PrSet.Printer ' get the Printer Name
```

PrtToDskAppendFile

このプロパティーは、**ディスクへの印刷 - コピー追加**モードのファイル・セットの名前です。このファイルは、ディスクへの印刷 - コピー追加ファイルと呼ばれます。このプロパティーには、以下のいずれかが入ります。

- **接続のディスクへの印刷 - コピー追加**ファイルの完全修飾パス名。
- 接続に**ディスクへの印刷 - コピー追加**ファイルが構成されていない場合、ヌル・ストリング。

PrtToDskAppendFile は、String データ型で読み取り専用です。

例

```
Dim PrSet as Object
Dim ConnList as Object
Dim DskAppFile as String

Set PrSet = CreateObject("ZIEWin.autECLPrinterSettings")
Set ConnList = CreateObject("ZIEWin.autECLConnList")
' Initialize the connection
ConnList.Refresh
PrSet.SetConnectionByHandle(ConnList(1).Handle)

DskAppFile = PrSet. PrtToDskAppendFile ' get the Disk append file.
```

PrtToDskSeparateFile

このプロパティーは、**ディスクへの印刷 - 別個**モードのファイル・セットの名前です。このファイルは、**ディスクへの印刷 - 別個**ファイルと呼ばれます。このプロパティーには、以下のいずれかが入ります。

- 接続の**ディスクへの印刷 - 別個**ファイルの完全修飾パス名。
- 接続に**ディスクへの印刷 - 別個**ファイルが構成されていない場合、ヌル・ストリング。

PrtToDskSeparateFile は、String データ型で読み取り専用です。

例

```
Dim PrSet as Object
Dim ConnList as Object
Dim DskSepFile as String

Set PrSet = CreateObject("ZIEWin.autECLPrinterSettings")
Set ConnList = CreateObject("ZIEWin.autECLConnList")
' Initialize the connection
ConnList.Refresh
PrSet.SetConnectionByHandle(ConnList(1).Handle)

DskSepFile = PrSet. PrtToDskSeparateFile ' get the Disk separate file.
```

PromptDialogOption

このプロパティーは、 印刷前に「プリンター設定」ダイアログを表示するオプションが設定されているかどうかを示します。PromptDialogOption は、Boolean データ型で読み取り専用です。

例

```
Dim PrSet as Object
Dim ConnList as Object
Dim PromptDialog as Boolean

Set PrSet = CreateObject("ZIEWin.autECLPrinterSettings")
Set ConnList = CreateObject("ZIEWin.autECLConnList")
' Initialize the connection
ConnList.Refresh
PrSet.SetConnectionByHandle(ConnList(1).Handle)

PromptDialog = PrSet.PromptDialogOption ' get the Prompt Dialog option
' or...
PrSet.PromptDialogOption = True 'set the Prompt Dialog option
```

名前

このプロパティーは、autECLPrinterSettings が設定された接続の接続名ストリングです。Z and I Emulator for Windows は、短い文字 ID (**A** から **Z** の単一の英字) のみをストリングで戻します。Z and I Emulator for Windows 接続でオープンできるのは、1 つの名前につき 1 つしかありません。例えば、一度に 1 つの接続 **A** のみをオープンできます。Name は、String データ型で読み取り専用です。

例

```
Dim PrSet as Object
Dim ConnList as Object
DIM Name as String

Set PrSet = CreateObject("ZIEWin.autECLPrinterSettings")
Set ConnList = CreateObject("ZIEWin.autECLConnList")
' Initialize the connection
ConnList.Refresh
PrSet.SetConnectionByHandle(ConnList(1).Handle)

Name = PrSet.Name 'Save the name
```

ハンドル

このプロパティーは、autECLPrinterSettings オブジェクトが設定された接続のハンドルです。特定の 1 つのハンドルに対して 1 つしか、Z and I Emulator for Windows 接続をオープンできません。例えば、一度に 1 つの接続 **A** のみをオープンできます。Handle は、Long データ型で読み取り専用です。

例

```
Dim PrSet as Object
Dim ConnList as Object
Dim Hand as Long

Set PrSet = CreateObject("ZIEWin.autECLPrinterSettings")
Set ConnList = CreateObject("ZIEWin.autECLConnList")
' Initialize the connection
ConnList.Refresh
PrSet.SetConnectionByHandle(ConnList(1).Handle)

Hand = PrSet.Handle ' save the handle
```

ConnType

このプロパティーは、autECLPrinterSettings が設定された接続タイプです。このタイプは、時間の経過とともに変更される場合があります。ConnType は、String データ型で読み取り専用です。

ストリング値	接続タイプ
DISP3270	3270 表示装置
DISP5250	5250 ディスプレイ
PRNT3270	3270 印刷装置
PRNT5250	5250 プリンター
ASCII	VT エミュレーション

例

```
Dim PrSet as Object
Dim ConnList as Object
Dim Type as String

Set PrSet = CreateObject("ZIEWin.autECLPrinterSettings")
Set ConnList = CreateObject("ZIEWin.autECLConnList")
' Initialize the connection
ConnList.Refresh
PrSet.SetConnectionByHandle(ConnList(1).Handle)

Type = PrSet.ConnType ' save the type
```

CodePage

このプロパティーは、autECLPrinterSettings が設定された接続のコード・ページです。このコード・ページは、時間の経過とともに変更される場合があります。CodePage は、Long データ型で読み取り専用です。

例

```
Dim PrSet as Object
Dim ConnList as Object
Dim CodePage as Long

Set PrSet = CreateObject("ZIEWin.autECLPrinterSettings")
Set ConnList = CreateObject("ZIEWin.autECLConnList")
' Initialize the connection
ConnList.Refresh
PrSet.SetConnectionByHandle(ConnList(1).Handle)

CodePage = PrSet.CodePage ' save the codepage
```

開始済み

このプロパティーは、エミュレーター・ウィンドウが開始されたかどうかを示します。ウィンドウがオープンしている場合、値は True です。 その他の場合は False です。Started は、Boolean データ型で読み取り専用です。

例

```
Dim PrSet as Object
Dim ConnList as Object

Set PrSet = CreateObject(".autECLPrinterSettings")
Set ConnList = CreateObject("ZIEWin.autECLConnList")
' Initialize the connection
ConnList.Refresh
PrSet.SetConnectionByHandle(ConnList(1).Handle)

' This code segment checks to see if A is started.
' The results are sent to a text box called Result.
If PrSet.Started = False Then
 Result.Text = "No"
Else
 Result.Text = "Yes"
End If
```

CommStarted

このプロパティーは、ホストへの接続の状況を示しています。ホストが接続されている場合、値は True です。 その他の場合は False です。CommStarted は、Boolean データ型で読み取り専用です。

例

```
Dim PrSet as Object
Dim ConnList as Object

Set PrSet = CreateObject("ZIEWin.autECLPrinterSettings")
Set ConnList = CreateObject("ZIEWin.autECLConnList")
```

```
' Initialize the connection
ConnList.Refresh
PrSet.SetConnectionByHandle(ConnList(1).Handle)

' This code segment checks to see if communications are connected
' for A. The results are sent to a text box called
' CommConn.
If PrSet.CommStarted = False Then
 CommConn.Text = "No"
Else
 CommConn.Text = "Yes"
End If
```

APIEnabled

このプロパティーは、エミュレーターが API 使用可能かどうかを示します。API 設定の状態に応じて、接続は API 使用可能または API 使用不可になります (Z and I Emulator for Windows のウィンドウで**「設定」→「API」**をクリック)。

エミュレーターが API 使用可能の場合には、値は True です。 その他の場合には、False です。APIEnabled は、Boolean データ型で読み取り専用です。

例

```
Dim PrSet as Object
Dim ConnList as Object

Set PrSet = CreateObject("ZIEWin.autECLPrinterSettings")
Set ConnList = CreateObject("ZIEWin.autECLConnList")
' Initialize the connection
ConnList.Refresh
PrSet.SetConnectionByHandle(ConnList(1).Handle)

' This code segment checks to see if A is API-enabled.
' The results are sent to a text box called Result.
If PrSet.APIEnabled = False Then
 Result.Text = "No"
Else
 Result.Text = "Yes"
End If
```

作動可能

このプロパティーは、エミュレーター・ウィンドウが開始されているか、API 使用可能か、接続されているかどうかを示します。このプロパティーは、3 つのすべてのプロパティーを確認します。エミュレーターが準備できている場合には、値は True です。 その他の場合には、False です。Ready は、Boolean データ型で読み取り専用です。

例

```
Dim PrSet as Object
Dim ConnList as Object
```

```
Set PrSet = CreateObject("ZIEWin.autECLPrinterSettings")
Set ConnList = CreateObject("ZIEWin.autECLConnList")
' Initialize the connection
ConnList.Refresh
PrSet.SetConnectionByHandle(ConnList(1).Handle)

' This code segment checks to see if A is ready.
' The results are sent to a text box called Result.
If PrSet.Ready = False Then
 Result.Text = "No"
Else
 Result.Text = "Yes"
End If
```

autECLPrinterSettings メソッド

以下のセクションで、autECLPrinterSettings オブジェクトに有効なメソッドを説明します。

void SetPDTMode(Boolean bPDTMode, [optional] String PDTFile) void SetPrtToDskAppend([optional] String FileName) void SetPrtToDskSeparate([optional] String FileName) void SetSpecificPrinter(String Printer) void SetWinDefaultPrinter() void SetConnectionByName (String Name) void SetConnectionByHandle (Long Handle)

SetPDTMode

SetPDTMode メソッドは、所定の PDT ファイルによって PDT モードに接続を設定するか、非 PDT モード (*GDI* モードとも呼ばれる) に接続を設定します。

制約事項

このメソッドが bPDTMode を False に設定して呼び出される場合には、 関連する接続の PrintMode が既に SpecificPrinter または WinDefaultPrinter に設定されている必要があります。

プロトタイプ

void SetPDTMode(Boolean bPDTMode, [optional] String PDTFile)

パラメーター

Boolean bPDTMode

以下の値を指定できます。

- PDT モードに接続を設定するには **TRUE**。
- 非 PDT モード (GDI モード) に接続を設定するには **FALSE**。

String PDTFile

このオプショナル・パラメーターには PDT ファイル名が入ります。

このパラメーターは、bPDTMode が TRUE の場合にのみ使用されます。このパラメーターが指定されていない場合に、bPDTMode が TRUE に設定されると、接続に構成されている PDT ファイルが使用されます。接続にまだ PDT ファイルが構成されていない場合、このメソッドは例外をスローして失敗します。

bPDTMode が FALSE の場合、このパラメーターは無視されます。

以下の値を指定できます。

- パスなしのファイル名

  Z and I Emulator for Windows のインストール・パスの PDFPDT サブフォルダー内の PDTFile が使用されます。
- ファイルの完全修飾パス名

  PDTFile が存在しない場合、このメソッドは例外をスローして失敗します。

戻り値

なし

例

```
Dim PrSet as Object
Dim ConnList as Object

Set PrSet = CreateObject("ZIEWin.autECLPrinterSettings")
Set ConnList = CreateObject("ZIEWin.autECLConnList")
' Initialize the connection
ConnList.Refresh
PrSet.SetConnectionByHandle(ConnList(1).Handle)

PrSet.SetPDTMode(True, "epson.pdt") 'Set PDT mode
PrSet.SetPDTMode(False) 'Set non-PDT mode (also called GDI mode)
```

SetPrtToDskAppend

このメソッドは、接続の PrintMode を**ディスクへの印刷 - コピー追加**モードに設定します。これは、このモードに該当するファイルも設定します。

注:

1. このファイルを設定するフォルダーには書き込みアクセス権限が必要です。権限がない場合、 このメソッドは例外をスローして失敗します。
2. 関連する接続は PDT モードにする必要があります。

プロトタイプ

void SetPrtToDskAppend([optional] String FileName)

パラメーター

String FileName

このオプショナル・パラメーターには、**ディスクへの印刷 - コピー追加**ファイルの名前が入ります。

ファイルが存在する場合には、それが使用されます。それ以外の場合、 印刷の完了時に作成されます。

以下の値を指定できます。

- 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を見つけます。
- ファイルの完全修飾パス名

  パス内にディレクトリーが存在している必要があります。 ない場合、メソッドは例外をスローして失敗します。ファイルがパス内に存在する必要はありません。

このパラメーターが指定されていない場合、 接続でこの PrintMode に構成されているファイルが使用されます。接続にまだファイルが構成されていないと、このメソッドは例外をスローして失敗します。

戻り値

なし

例

```
Dim PrSet as Object
Dim ConnList as Object

Set PrSet = CreateObject("ZIEWin.autECLPrinterSettings")
Set ConnList = CreateObject("ZIEWin.autECLConnList")
' Initialize the connection
ConnList.Refresh
PrSet.SetConnectionByHandle(ConnList(1).Handle)

'If PDTMode, set PrintMode to pcPrtToDskAppend
If PrSet.PDTMode Then
 PrSet.SetPrtToDskAppend("dskapp.txt")
```

SetPrtToDskSeparate

このメソッドは、接続の PrintMode を**ディスクへの印刷 - 別個**モードに設定します。これは、このモードに該当するファイルも設定します。

注:

1. このファイルを設定するフォルダーには書き込みアクセス権限が必要です。権限がない場合、 このメソッドは例外をスローして失敗します。
2. 関連する接続は PDT モードにする必要があります。

プロトタイプ

void SetPrtToDskSeparate([optional] String FileName)

パラメーター

String FileName

このオプショナル・パラメーターには、**ディスクへの印刷 - 別個**ファイルの名前が入ります。

このパラメーターが指定されていない場合、 接続でこの PrintMode に構成されているファイルが使用されます。

指定可能な値は以下のとおりです。

- **NULL** (デフォルト)

  接続でこの PrintMode に現在構成されているファイルが使用されます。接続にまだファイルが構成されていないと、このメソッドは例外をスローして失敗します。
- 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を見つけます。
- ファイルの完全修飾パス名

  パス内にディレクトリーが存在している必要があります。 ない場合、メソッドは例外をスローして失敗します。ファイルがパス内に存在する必要はありません。

注: ファイル名には拡張子を入れないでください。拡張子が入っていると、このメソッドは例外をスローして失敗します。

戻り値

なし

例

```
Dim PrSet as Object
Dim ConnList as Object

Set PrSet = CreateObject("ZIEWin.autECLPrinterSettings")
Set ConnList = CreateObject("ZIEWin.autECLConnList")
' Initialize the connection
ConnList.Refresh
PrSet.SetConnectionByHandle(ConnList(1).Handle)

'If PDTMode, set PrintMode to pcPrtToDskSeparate
If PrSet.PDTMode Then
 PrSet.SetPrtToDskSeparate("dsksep")
```

SetSpecificPrinter

このメソッドは、接続の PrintMode を Printer パラメーターで指定されたプリンターによって特定のプリンター・モードに設定します。

プロトタイプ

void SetSpecificPrinter(String Printer)

パラメーター

String Printer

プリンターの名前が入ります。プリンターが存在しない場合、このメソッドは例外をスローして失敗します。
値の形式は次のようにする必要があります。

```
<Printer name> on <Port Name>
```

例:

- `HP LaserJet 4050 Series PCL 6 on LPT1`

戻り値

なし

例

```
Dim PrSet as Object
Dim ConnList as Object

Set PrSet = CreateObject("ZIEWin.autECLPrinterSettings")
Set ConnList = CreateObject("ZIEWin.autECLConnList")
' Initialize the connection
ConnList.Refresh
PrSet.SetConnectionByHandle(ConnList(1).Handle)

'Set PrintMode to pcSpecificPrinter
PrSet. SetSpecificPrinter("HCL InfoPrint 40 PS on Network Port")
```

SetWinDefaultPrinter

このメソッドは、接続の PrintMode を Windows のデフォルト・プリンター・モードに設定します (接続は Windows のデフォルト・プリンターを使用します)。Windows のデフォルト・プリンターが構成されていない場合、 このメソッドは例外をスローして失敗します。

プロトタイプ

void SetWinDefaultPrinter()

パラメーター

なし

戻り値

なし

例

```
Dim PrSet as Object
Dim ConnList as Object

Set PrSet = CreateObject("ZIEWin.autECLPrinterSettings")
Set ConnList = CreateObject("ZIEWin.autECLConnList")
' Initialize the connection
ConnList.Refresh
PrSet.SetConnectionByHandle(ConnList(1).Handle)

'Set PrintMode to pcWinDefaultPrinter
PrSet. SetWinDefaultPrinter
```

SetConnectionByName

この SetConnectionByName メソッドは、接続名を使用して、新しく作成された autECLPrinterSettings オブジェクトの接続を設定します。Z and I Emulator for Windows では、この接続名は、短い接続 ID (**A** から **Z** の単一の英字) です。Z and I Emulator for Windows 接続でオープンできるのは、1 つの名前につき 1 つしかありません。例えば、一度に 1 つの接続 **A** のみをオープンできます。

注: autECLSession オブジェクトに含まれる autECLPrinterSettings オブジェクトを使用している場合には、このメソッドを呼び出さないでください。

プロトタイプ

void SetConnectionByName (String Name)

パラメーター

String 型、Name 型

1 文字の接続のストリング短縮名。有効値は A から Z です。

戻り値

なし

例

以下の例は、接続名を使用して、新しく作成された autECLPrinterSettings オブジェクトの接続を設定する方法を示します。

```
Dim PrSet as Object
Set PrSet = CreateObject("ZIEWin.autECLPrinterSettings")
' Initialize the connection
PrSet.SetConnectionByName("A")
' For example, see if PDTMode
If PrSet.PDTMode Then
'your logic here...
End If
```

SetConnectionByHandle

この SetConnectionByHandle メソッドは、接続ハンドルを使用して、新しく作成された autECLPrinterSettings オブジェクトの接続を設定します。Z and I Emulator for Windows では、この接続ハンドルは Long integer です。

特定の 1 つのハンドルに対して 1 つしか、Z and I Emulator for Windows 接続をオープンできません。例えば、一度に 1 つの接続 **A** のみをオープンできます。

注: autECLSession オブジェクトに含まれる autECLPrinterSettings オブジェクトを使用している場合には、このメソッドを呼び出さないでください。

プロトタイプ

void SetConnectionByHandle(Long Handle)

パラメーター

Long Handle

オブジェクトに設定される接続の Long integer 値。

戻り値

なし

例

以下の例は、接続ハンドルを使用して、新しく作成された autECLPrinterSettings オブジェクトの接続を設定する方法を示します。

```
Dim PrSet as Object
Dim ConnList as Object

Set PrSet = CreateObject("ZIEWin.autECLPrinterSettings")
Set ConnList = CreateObject("ZIEWin.autECLConnList")
' Initialize the connection
ConnList.Refresh
PrSet.SetConnectionByHandle(ConnList(1).Handle)

' For example, see if PDTMode
If PrSet.PDTMode Then
'your logic here...
End If
```

自動化オブジェクトへのプライマリー相互運用アセンブリーのサポート

HCL Z and I Emulator for Windows により公開された自動化オブジェクトは、.NET フレームワークを対象とする任意の言語で作成されたアプリケーションで使用できます。管理対象の .NET アプリケーションは、自動化オブジェクトをラップするプライマリー相互運用アセンブリー (PIA) を使用して、Z and I Emulator for Windows をプログラミングできます。相互運用アセンブリーは、管理対象の (.NET) アプリケーションが COM 準拠オブジェクトを使用する際のメカニズムです。相互運用アセンブリーには、バインディングおよびメタデータ情報が含まれ、 これにより、.NET フレームワーク (CLR) は COM オブジェクトをロードまたはマーシャルして、.NET アプリケーション用にラップできます。PIA には、COM 型の提供者によって定義された COM 型の正式な記述が含まれます。PIA は、必ずオリジナルの COM 型の提供者によってデジタル署名されています。

.NET アプリケーションがアセンブリーを参照できる方法は 2 つあります。

- 単純なアプリケーションまたは単一のアプリケーションがアセンブリーを使用する場合、 Microsoft はアセンブリーをアプリケーションと同じディレクトリーにコピーすることを推奨しています。
- 複数のアプリケーションがアセンブリーを参照する場合、 アプリケーションをグローバル・アセンブリー・キャッシュ (GAC) にインストールして、すべてのソリューションに GAC の アセンブリーを参照させることができます。

相互運用アセンブリーによって公開されている型のプログラミング・モデルは COM とよく似ています。COM オブジェクトによって公開されているメソッド、プロパティー、 およびイベントは、いずれの .NET 言語でもその言語の構文を使用してアクセスできます。C# で作成されたサンプル・アプリケーション (ECLSamps.net) が Z and I Emulator for Windows のインストール・イメージの \samples ディレクトリーに提供されています。このサンプルは、 相互運用アセンブリーのさまざまな型の単純な使用法を示します。

Visual Basic 6.0 で、Z and I Emulator for Windows の自動化オブジェクトを使用し、変換支援ウィザードを使用して Visual Basic .NET に移行したプロジェクトでは、ユーザーは、対応する Z and I Emulator for Windows 相互運用参照 (\Interops ディレクトリー) を使用して変換支援ウィザードによって暗黙で生成される参照を置換して、再コンパイルするだけです。参照を置換する方法は、 変換支援によって生成されたすべての参照を削除し、Visual Studio .NET を使用して .NET 相互運用参照を追加します。GAC に登録してあるアセンブリーを使用する場合、参照を追加して、Z and I Emulator for Windows 相互運用参照の「ローカルのコピー」プロパティーを**「False」**に設定します。

Z and I Emulator for Windows のエミュレーター自動化オブジェクトの PIA は、Z and I Emulator for Windows のインストール・イメージの \Interops ディレクトリーにインストールされます。Z and I Emulator for Windows 製品インストーラーによって .NET フレームワークが存在することが検出された場合、GAC 内に型を登録するための追加オプションが提供されます。GAC にアセンブリーをインストールすると、PIA は対応する型ライブラリーのレジストリー・キーの下のレジストリーにも置かれます。

表 2 : Z and I Emulator for Windows の自動化オブジェクトへのプライマリー相互運用アセンブリー (ページ 426)
に、Z and I Emulator for Windows の自動化オブジェクトに提供される PIA をリストします。

表 2. Z and I Emulator for Windows の自動化オブジェクトへのプライマリー相互運用アセンブリー

自動化オブジェクト	相互運用アセンブリー依存関係
autECLConnList	Interop.AutConnListTypeLibrary.dll
autECLConnMgr	Interop.AutConnMgrTypeLibrary.dll
autECLConnList	Interop.AutPSTypeLibrary.dll
autECLOIA	Interop.AutOIATypeLibrary.dll
autECLPS	Interop.AutPSTypeLibrary.dll
autECLScreenDesc	Interop.AutScreenDescTypeLibrary.dll
autECLScreenReco	Interop.AutScreenRecoTypeLibrary.dll
autECLSession	Interop.AutSessTypeLibrary.dll
autECLPageSettings	Interop.AutSettingsTypeLibrary.dll
autECLPrinterSettings	Interop.AutSettingsTypeLibrary.dll
autECLWinMetrics	Interop.AutWinMetricsTypeLibrary.dll

表 2. Z and I Emulator for Windows の自動化オブジェクトへのプライマリー相互運用アセンブリー (続く)

自動化オブジェクト	相互運用アセンブリー依存関係
autECLXfer	Interop.AutXferTypeLibrary.dll
autSystem	Interop.AutSystemTypeLibrary.dll

第 4 章. Java 用ホスト・アクセス・クラス・ライブラリー

ホスト・アクセス・クラス・ライブラリー (HACL) Java クラスでは、Java プログラミング環境で Z and I Emulator for Windows の HACL 関数が使用できるようにします。したがって、HACL の 各クラスで提供されている関数を使用する Java アプレットおよびアプリケーションの作成が可能です。

HACL Java HTML ファイルは、以下のパスにある Z and I Emulator for Windows の製品資料に付属している Docs_Admin_Aids zip フォルダーにあります: *ZIEWin_3.0_Docs_Admin_Aids.zip\publications\ja\doc\hacl* ディレクトリー。

第 5 章. トラブルシューティング

問題のトラブルシューティングを支援する以下のセルフヘルプ情報リソースとツールを使用できます。

- ご使用の製品のリリース情報で、既知の問題、次善策、およびトラブルシューティング情報を参照します。
- お客さまの問題を解決するダウンロードまたはフィックスが提供されているどうかを確認する。
- 提供されているナレッジ・ベースを検索して、問題の解決方法が既に 文書化されているかどうかを調べる。
- さらに支援が必要な場合は、HCL ソフトウェア・サポートに連絡して、問題を報告してください。

HCL Z and I Emulator for Windows .NET Interop アセンブリーでセッション OIA 通知を起動できない

問題

OIA イベント通知の登録を行う .NET アプリケーションが、これらのイベントの通知を受け取りません。また、対応する COM タイプ・ライブラリーのいくつかのメソッドが Visual Studio のインテリジェンス機能で表示されません。

原因

.NET Interop アセンブリーは、Microsoft SDK に付属するツール TlbImp.exe を使用して対応する COM タイプ・ライブラリーから派生します。タイプ・ライブラリー・インポーターは、COM タイプ・ライブラリー内で見つかったタイプ定義を、共通言語ランタイム・アセンブリー内の同等の定義に変換します。しかし、ランタイム・マーシャラーは一部のデータ・タイプをマーシャルできません。そのため、いくつかの COM タイプ・ライブラリー定義が、結果として生成された共通言語ランタイム・アセンブリーに見つかりません。

解決方法

これは TlbImp.exe の制限事項です。

付録 A. Sendkeys 略号キーワード

表 3 : Sendkeys メソッドの略号キーワード (ページ 430) では、Sendkeys メソッドの略号キーワードを記載しています。

表 3. Sendkeys メソッドの略号キーワード

キーワード	説明
[backtab]	後退タブ
[clear]	画面消去
[delete]	削除
[enter]	Enter
[eraseeof]	フィールドの終わりを消去
[help]	ヘルプ\n
[insert]	挿入
[jump]	ジャンプ
[left]	左
[newline]	新規行
[space]	スペース
[print]	印刷
[reset]	リセット
[tab]	Tab
[up]	上へ
[Down]	下
[capslock]	CapsLock
[right]	右
[home]	ホーム
[pf1]	PF2
[pf2]	PF2
[pf3]	PF3
[pf4]	PF4
[pf5]	PF5
[pf6]	PF6
[pf7]	PF7
[pf8]	PF8
[pf9]	PF9
[pf10]	PF10
[pf11]	PF11
[pf12]	PF12
[pf13]	PF13

表 3. Sendkeys メソッドの略号キーワード (続く)

キーワード	説明
[pf14]	PF14
[pf15]	PF15
[pf16]	PF16
[pf17]	PF17
[pf18]	PF18
[pf19]	PF19
[pf20]	PF20
[pf21]	PF21
[pf22]	PF22
[pf23]	PF23
[pf24]	PF24
[eof]	ファイルの終わり
[scrlock]	[Scroll Lock]
[numlock]	[Num Lock]
[pageup]	PageUp
[pagedn]	PageDown
[pa1]	PA 1
[pa2]	PA 2
[pa3]	PA 3
[test]	テスト
[worddel]	ワード削除
[fldext]	フィールドの終了
[erinp]	入力消去
[sysreq]	システム要求
[instog]	挿入切り替え
[crsel]	カーソル選択
[fastleft]	高速カーソル左移動
[attn]	警告
[devcance]	装置取り消し (DvCnl)
[printps]	表示スペース印刷
[fastup]	高速カーソル上移動
[fastdown]	高速カーソル下移動
[hex]	16 進
[fastright]	高速カーソル右移動
[revvideo]	反転表示
[underscr]	下線
[rstvideo]	反転表示のリセット

表 3. Sendkeys メソッドの略号キーワード (続く)

キーワード	説明
[red]	赤
[pink]	Pink
[green]	緑
[yellow]	黄
[blue]	青
[turq]	ターコイズ
[white]	White
[rstcolor]	ホスト・カラー のリセット
[printpc]	印刷 (PC)
[wordright]	正方向ワード・タブ
[wordleft]	逆方向ワード・タブ
[field-]	フィールド -
[field+]	フィールド +
[rcdbacksp]	レコード・バックスペース
[printhost]	ホストでの表示スペース印刷
[dup]	重複
[fieldmark]	フィールド・マーク
[dispsosi]	SO/SI の表示
[gensosi]	SO/SI 生成
[dispattr]	表示属性
[fwdchar]	文字前進
[splitbar]	縦破線
[altcsr]	カーソル切り替え
[backspace]	Backspace
[NULL]	NULL

付録 B. ECL プレーン ー 形式および内容

この付録では、ECL 表示スペース・モデルでの 種々のデータ・プレーンの形式および内容を説明します。各プレーンでは、ホスト表示スペースの 異なる面を示しています。例えば、文字の内容、色指定、 フィールド属性などです。ECL::GetScreen メソッドその他によって、 異なる表示スペース・プレーンからデータを戻します。

各プレーンには、各ホスト表示スペース文字の位置 につき 1 バイトが含まれています。それぞれのプレーンについては、論理的内容およびデータ・フォーマットの点から、以下のセクションに分けて説明しています。プレーンのタイプは、ECLPS.HPP ヘッダー・ファイルに列挙されています。

TextPlane

テキスト・プレーンには、表示スペースにある可視の文字が表示されます。テキスト・プレーンには非表示フィールドも表示されます。テキスト・プレーンの それぞれの要素のバイト値は、表示される文字の ASCII 値に対応します。テキスト・プレーンには、バイナリー・ゼロ (null) 文字値は含まれません。表示スペース内の null 文字 (null 埋め込み 入力フィールドなど) は、ASCII ブランク (0x20) 文字として示されます。

FieldPlane

フィールド・プレーンは、表示スペースでのフィールド位置および属性を示します。このプレーンは、フィールド形式の表示スペースにのみ意味があります(例えば、VT 接続はフォーマット設定されていません)。

このプレーンは、フィールド属性値の散在的な配列です。このプレーンの値は、表示スペースでフィールド属性文字がある場所を除いてすべてバイナリー・ゼロです。バイナリー・ゼロ以外の位置では、値はそこで開始するフィールドの属性です。フィールドの長さは、フィールド属性位置と表示スペース内の次のフィールド属性との間の線形距離であり、 その属性位置自体は含まれません。

フィールド属性位置の値は、以下の表に示されているとおりです。

注: 属性値は、接続のタイプによって異なります。

表 4. 3270 フィールド属性

ビット位置 (0 が最下位のビット)	意味
7	常に "1"
6	常に "1"
5	**0** 無保護 **1** 保護されています

表 4. 3270 フィールド属性 (続く)

ビット位置 (0 が最下位のビット)	意味
4	**0** 英数字データ **1** 数値データのみ
3, 2	**0, 0** 通常輝度、ペン検出不可能 **0, 1** 通常輝度、ペン検出可能 **1, 0** 高輝度、ペン検出可能 **1, 1** 非表示、ペン検出不可能
1	予約済み
0	**0** フィールドは変更されていません **1** 無保護フィールドが変更されました

表 5. 5250 フィールド属性

ビット位置 (0 が最下位のビット)	意味
7	常に "1"
6	**0** 非表示 **1** 表示
5	**0** 無保護 **1** 保護されています
4	**0** 通常輝度

表 5. 5250 フィールド属性 (続く)

ビット位置 (0 が最下位のビット)	意味
	1 高輝度
3, 2, 1	**0, 0, 0** 英数字データ **0, 0, 1** 英字のみ **0, 1, 0** 数字シフト **0, 1, 1** 数値データおよび数値特殊記号 **1, 0, 1** 数値のみ **1, 1, 0** 磁気ストライプ読み取り装置データのみ **1, 1, 1** 符号付き数字のみ
0	**0** フィールドは変更されていません **1** 無保護フィールドが変更されました

表 6 : マスク値 (ページ 435) は、さまざまなマスク値を定義しています。

表 6. マスク値

ニーモニック	マスク	説明
FATTR_MDT	0x01	変更フィールド
FATTR_PEN_MASK	0x0C	ペン検出可能フィールド
FATTR_BRIGHT	0x08	高輝度フィールド
FATTR_DISPLAY	0x0C	可視フィールド
FATTR_ALPHA	0x10	英数字フィールド
FATTR_NUMERIC	0x10	数値のみフィールド
FATTR_PROTECTED	0x20	保護フィールド
FATTR_PRESENT	0x80	フィールド属性の表示
FATTR_52_BRIGHT	0x10	5250 高輝度フィールド

表 6. マスク値 (続く)

ニーモニック	マスク	説明
FATTR_52_DISP	0x40	5250 可視フィールド

ColorPlane

カラー・プレーンには、 表示スペースのそれぞれの文字のカラー情報が含まれています。それぞれの文字の前景および背景の色は、 ホスト・データ・ストリームで指定されたとおりに表現されます。カラー・プレーンの色は、エミュレーター・ウィンドウのカラー表示マッピングにより修正されることはありません。カラー・プレーンの各バイトには、以下のカラー情報が入ります。

表 7. カラー・プレーン情報

ビット位置 (0 が最下位のビット)	意味
7 - 4	**背景の文字色** **0x0** 空白 **0x1** 青 **0x2** 緑 **0x3** シアン **0x4** 赤 **0x5** マジェンタ **0x6** 茶 (3270)、黄色 (5250) **0x7** White
3-0	**前景の文字色** **0x0** 空白

表 7. カラー・プレーン情報 (続く)

ビット位置 (0 が最下位のビット)	意味
	0x1 青 **0x2** 緑 **0x3** シアン **0x4** 赤 **0x5** マジェンタ **0x6** 茶 (3270)、黄色 (5250) **0x7** 白 (通常の輝度) **0x8** グレー **0x9** 明るい青 **0xA** 明るい緑 **0xB** 明るいシアン **0xC** 明るい赤 **0xD** 明るいマジェンタ **0xE** 黄 **0xF** ホワイト (高輝度)

ExfieldPlane

このプレーンには、拡張文字属性データが入ります。

このプレーンは、拡張文字属性値の散在的な配列です。配列内の値は、ホストが拡張文字属性を指定した 表示スペースでの文字以外は、すべてバイナリー・ゼロです。拡張文字属性値の意味は、以下のとおりです。

表 8. 3270 拡張文字属性

ビット位置 (0 が最下位のビット)	意味
7, 6	**文字の強調表示** **0, 0** 正常 **0, 1** 明滅 **1, 0** 反転表示 **1, 1** 下線
5, 4, 3	**文字色** **0, 0, 0** デフォルト **0, 0, 1** 青 **0, 1, 0** 赤 **0, 1, 1** Pink **1, 0, 0** 緑 **1, 0, 1** ターコイズ **1, 1, 0** 黄

表 8. 3270 拡張文字属性 (続く)

ビット位置 (0 が最下位のビット)	意味
	1, 1, 1 White
2, 1	**文字属性** **00** デフォルト **11** 2 バイト文字
0	予約済み

表 9. 5250 拡張文字属性

ビット位置 (0 が最下位のビット)	意味
7	**0** 標準イメージ **1** 反転イメージ
6	**0** 下線なし **1** 下線
5	**0** 明滅なし **1** 明滅
4	**0** 桁分離線なし **1** 桁分離線あり
3, 2, 1, 0	予約済み

付録 C. 特記事項

本書は米国 HCL 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 HCL の営業担当員にお尋ねください。本書で HCL 製品、プログラム、またはサービスに言及していても、その HCL 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、HCL 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、HCL 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。

本書に記述されている主題事項に関して HCL が特許権 (特許出願を含む) を所有していることがあります。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。

HCL
330 Potrero Ave.
Sunnyvale, CA 94085
USA
注意: Office of the General Counsel

HCL TECHNOLOGIES LTD. 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。HCL 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 HCL 以外の文書または HCL 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、決してそれらの文書または 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 HCL 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。

HCL は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

HCL
330 Potrero Ave.
Sunnyvale, CA 94085
USA
注意: Office of the General Counsel

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、HCL 所定のプログラム契約の契約条項、HCL 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、HCL より提供されます。

本書に含まれるパフォーマンス・データは、特定の動作および環境条件下で得られたものです。実際の結果は、異なる可能性があります。

HCL 以外の製品に関する情報は、その製品の供給者もしくは公開されているその他のソースから入手したものです。HCL 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 HCL は確証できません。HCL 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、名称や住所が類似する企業が実在しているとしても、それは偶然にすぎません。

商標

HCL、HCL ロゴおよび hcl.com は、世界の多くの国で登録された HCL Technologies Ltd. の商標または登録商標です。他の製品名およびサービス名等は、それぞれ IBM® または各社の商標である場合があります。