主机访问类库

关于本书

本书提供了使用 HCL Z and I Emulator for Windows、Version 3.0 主机访问类库 (HACL) 所需的编程信息。在本书中，*Windows*® 是指 Windows® 7、Windows® 8、Windows® 8.1、Windows® 10、Windows® Server 2008 和 Windows® Server 2012。本书中所述的工作站是指所有受支持的个人计算机。当仅提到个人计算机的一种型号或体系结构时，仅指定该类型。

本书的读者对象

本书面向编写使用主机访问类库 (HACL) 函数的应用程序的程序员和开发人员。

其中假定程序员具备 Windows® 的工作知识。有关 Windows® 的信息，请参阅 到何处查找更多信息 (on page 2) 下的出版物列表。

本书假定您熟悉所使用的语言和编译器。有关如何编写、编译或链接编辑程序的信息，请参阅 到何处查找更多信息 (on page 2) 以获取要使用的特定语言的相应参考。

如何使用本书

本书的组织方式如下：

到何处查找更多信息

Z and I Emulator for Windows 资料库包含下列出版物：

- 安装指南
- *Quick Beginnings*
- *Emulator User's Reference*
- *Administrator's Guide and Reference*
- *Emulator Programming*
- 主机访问类库

除了出版的书籍外，还随“Z and I Emulator for Windows”提供了 HTML 文档：

Host Access Class Library for Java

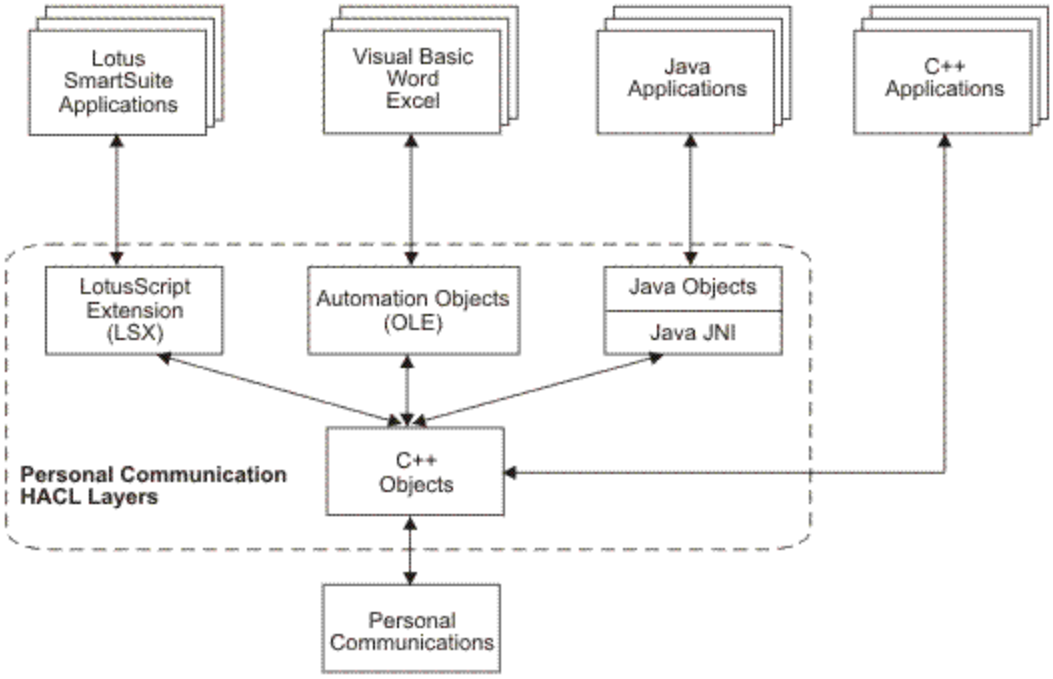
HACL Java HTML 介绍如何编写符合 ActiveX/OLE 2.0 的应用程序，以将 Z and I Emulator for Windows 用作嵌入式对象。这些文件可以从与 Z and I Emulator for Windows 产品文档一起提供的 Docs_Admin_Aids 压缩文件夹中访问，路径如下： *ZIEWin_3.0_Docs_Admin_Aids.zip\publications\zh_CN\doc\hacl*

第 1 章. 介绍

主机访问类库 (HACL) 是一组对象，允许应用程序程序员轻松快速地访问主机应用程序。通过支持多个不同的 HACL 层，HCL Z and I Emulator for Windows 为各种编程语言和环境提供支持：C++ 对象、Java™ 对象、Microsoft® 基于 COM 的自动化技术 (OLE)。每一层都提供相同的基本功能，但由于每个环境的语法和功能不同，每一层都会有一些不同之处。功能最强、最灵活的层是 C++ 层，它为所有其他层提供基础。

此分层概念允许将基本 HACL 功能用于各种编程环境，包括 Java™、Microsoft® Visual Basic®、Visual Basic® for Applications、Lotus® Notes™、Lotus® WordPro 和 Visual C++®。下图显示了 HACL 层。

图 1. HACL 层



C++ 对象

此 C++ 类库提供主机连接的完全面向对象的抽象，其中包括：读取和写入主机表示空间（屏幕），枚举屏幕上的字段，读取操作员指示符区 (OIA) 以获取状态信息，访问和更新有关可视仿真器窗口的信息，传输文件，以及执行重要事件的异步通知。

请参阅 Host Access Class Library C++ (on page 12) 以获取有关 C++ 对象的详细信息。

Java 对象

Java™ 对象提供 Java™ 包装，适用于所有 HACL 函数，类似于 Host-On-Demand V3。请参阅 Host Access Class Library for Java (on page 409) 以获取有关 HACL Java™ 类的详细信息。

自动化对象

主机访问类库自动化对象允许 Z and I Emulator for Windows 支持 Microsoft® 基于 COM 的自动化技术（以前称为 OLE 自动化）。HACL 自动化对象是一系列允许自动化控制器的自动化服务器，例如 Microsoft® Visual Basic®，以便以编程方式访问 Z and I Emulator for Windows 的数据和功能。换言之，支持控制自动化协议的应用程序（自动化控制器）可以控制一些 Z and I Emulator for Windows 操作（自动化服务器）。

注： HCL Z and I Emulator for Windows 提供的自动化对象本质上是 32 位。这些只能用于 32 位 Microsoft Office 程序。

请参阅 Host Access Class Library Automation Objects (on page 242) 以获取有关自动化对象层的详细信息。

ECL 概念

以下各节介绍了仿真器类库 (ECL) 的几个基本概念。了解这些概念将有助于您有效地使用库。

连接、句柄和名称

在 ECL 上下文中，连接是一个唯一的 Z and I Emulator for Windows 仿真器窗口。仿真器窗口可能实际连接到主机，也可能不连接，并且可能在屏幕上显示，也可能不显示。例如，Z and I Emulator for Windows 窗口可以处于已断开连接状态。连接通过其连接句柄或连接名称来区分。大多数 HACL 对象都与特定连接相关联。通常，对象在对象的构造函数上使用连接句柄或连接名称作为参数。对于不支持构造函数参数的 Visual Basic® 等语言，提供了成员函数来进行关联。构造后，对象就不能与任何其他连接相关联。例如，要创建与连接“B”关联的 ECLPS（表示空间）对象，将使用以下代码：

C++

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

Visual Basic®

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

HACL 连接名称是 A-Z 或 a-z 中的单个字符。最多可有 52 个连接名称，Z and I Emulator for Windows 当前限制为 52 个并发连接。连接的名称与其 EHLLAPI 短会话标识以及 Z and I Emulator for Windows 窗口标题和 OIA 上显示的会话标识相同。

HACL 句柄是代表单个连接的唯一 32 位数字。与连接名称不同，连接句柄不限于 52 个值，并且该值本身对应用程序没有意义。可以跨线程和进程使用连接句柄来引用相同的连接。

对于将来的扩展，应用程序应尽可能使用连接句柄。当需要标识连接时，大多数 HACL 对象接受句柄或名称。HACL 基类中有一些函数可用于将句柄转换为名称，以及将名称转换为句柄。这些函数可从任何 HACL 对象中使用。

注： 连接属性是动态属性。例如，如果将连接重新配置到其他主机，GetConnType 返回的连接类型可能会更改。通常，应用程序不应假定连接属性保持固定。

会话

在 ECL 的上下文中，会话对象 (ECLSession) 只是所有其他连接特定对象的容器。它为应用程序提供快捷方式，用于为特定连接创建一组完整的 HACL 对象。不能将术语会话与 Z and I Emulator for Windows 会话概念混淆。Z and I Emulator for Windows 会话是指屏幕上的物理仿真窗口。

创建或破坏 ECLSession 对象不会影响 Z and I Emulator for Windows 会话（窗口）。应用程序可以创建任意数量的引用相同或不同连接的 ECLSession 对象。

ECL 容器对象

多个 HACL 类充当其他对象的容器。例如，，ECLSession 对象包含 ECLPS、ECLOIA、ECLWinMetrics 和 ECLXfer 对象的实例。容器提供返回指向所包含对象的指针的方法。例如，ECLSession 对象具有 GetOIA 方法，该方法返回指向 OIA 对象的指针。包含的对象不是作为容器类的公用成员实现的，而是只能通过方法访问。

由于性能或其他原因，创建容器对象时可能会创建包含的对象，也可能不会创建包含的对象。类实现可选择推迟所包含对象的构造，直到应用程序第一次请求指向这些对象的指针。应用程序不应假定包含的对象与容器同时创建。例如，构造 ECLSession 对象时，可能不会构造 ECLPS 对象的实例。相反，ECLSession 类可能会将 ECLPS 对象的构造推迟到首次调用 GetPS 方法时。

当容器类被破坏时，所有包含的实例也将被破坏。返回到应用程序的任何指针都将无效，不得使用。

注： 某些 HACL 层（如自动化对象）可能会隐藏包含方案，或将其重新转换为不使用显式指针的命名方案

ECL 列表对象

多个 HACL 类提供列表迭代功能。例如，ECLConnList 类管理连接列表。ECL 列表类不会异步更新来反映列表内容中的更改。应用程序必须显式调用 Refresh 方法来更新列表的内容。这样，应用程序就可以迭代列表，而不必担心该列表在迭代过程中会更改。

事件

HACL 提供某些事件的异步通知功能。应用程序可以选择在发生特定事件时收到通知。例如，当启动新的 Z and I Emulator for Windows 连接时，应用程序会收到通知。当前，HACL 支持以下事件的通知：

- 连接启动/停止
- 通信连接/断开连接
- 操作员击键
- 表示空间或 OIA 更新

事件通知 由ECLNotify 抽象基类实现。每个事件类型都有单独的类。要获得事件通知，应用程序必须定义和创建从其中一个 ECLNotify 抽象基类派生的对象。接着，必须通过调用相应的 HACL 注册函数来注册该对象。注册应用程序对象后，每当发生感兴趣的事件时都会调用其 NotifyEvent 方法。

注:

1. 应用程序的 NotifyEvent 方法在单独的执行线程上异步调用。因此，NotifyEvent 方法应该可重入，如果它访问应用程序资源，则应使用适当的锁定或同步。
2. 某些 HACL 层（如自动化对象）可能不完全支持或实现 HACL 事件。

错误处理

在 C++ 层，HACL 使用 C++ 结构化异常处理。通常，通过使用 ECLErr 对象抛出 C++ 异常，向应用程序指示错误。要捕获错误，应用程序应将对 HACL 对象的调用包含在 try/catch 块中，例如：

```
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");
}
```

当捕获到 HACL 错误时，应用程序可以调用 ECLErr 对象上的方法来确定错误的确切原因。也可以调用 ECLErr 对象来构造完整的语言敏感错误消息。

在自动化对象层中，运行时错误会导致创建相应的脚本错误。应用程序可以使用 On Error 处理程序来捕获错误，查询有关错误的其他信息，并采取适当的措施。

寻址（行、列、位置）

HACL 提供两种在主机表示空间中对点（字符位置）进行寻址的方法。应用程序可以按行号/列号或单个线性位置值来进行字符寻址。无论使用哪种寻址方案，表示空间寻址始终基于 1（而不是从零开始）。

行/列寻址方案对于与主机数据的物理屏幕显示直接相关的应用程序非常有用。矩形坐标系（第 1 行第 1 列位于左上角）是在屏幕上对点进行寻址的自然方式。线性位置寻址方法（位置 1 位于左上角，从左到右，从上到下）对于将整个表示空间作为单个数据元素数组查看的应用程序非常有用，对于从使用此寻址方案的 EHLLAPI 接口移植的应用程序也非常有用。

在 C++ 层，通过为相同的方法调用不同的签名来选择不同的寻址方案。例如，要将主机光标移到给定的屏幕坐标，应用程序可以通过以下两个签名之一调用 ECLPS::SetCursorPos 方法：

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

如果将主机屏幕配置为每行 80 列，则这些语句具有相同的效果。此示例还指出了寻址方案的细微差别。如果应用程序对表示空间的每行字符数作出假设，线性位置方法可能会产生意外的结果。例如，示例中的第一行代码将光标放在为 132 列配置的表示空间中第 1 行的第 81 列。第二行代码会将光标置于第 2 行第 1 列，而不管表示空间的配置如何。

注： 某些 HACL 层可能仅公开一个编址方案。

从 EHLLAPI 迁移

当前写入仿真器高级语言 API (EHLLAPI) 的应用程序可以修改为使用主机访问类库。一般情况下，它需要大量的源代码更改或应用程序重组才能从 EHLLAPI 迁移到 HACL。HACL 提供与 EHLLAPI 不同的编程模型，通常需要不同的应用程序结构才会有效。

以下几节将帮助熟悉 EHLLAPI 的程序员了解 HACL 的相似性，以及 HACL 与 EHLLAPI 的不同之处。使用此信息，您可以了解如何修改特定应用程序以使用 HACL。

注： EHLLAPI 使用术语*会话*来表示与 HACL *连接*相同的内容。这些术语在本节中可以互换使用。

执行/语言接口

在最基本的层面上，EHLLAPI 和 HACL 在应用程序如何调用 API 的机制上有所不同。

EHLLAPI 作为具有多用途参数的单个调用点接口。DLL 中的单个入口点 (hllapi) 基于一组固定的四个参数提供所有函数。根据第四个命令参数的值，其中三个参数具有不同的含义。这个简单的接口使得从各种编程环境和语言调用 API 变得更容易。缺点是一个函数和四个参数非常复杂。

HACL 是一个面向对象的接口，提供一组编程对象，而不是显式入口点或函数。这些对象具有可用于操控主机连接的属性和方法。您不必关注结构打包和参数命令代码的详细信息，但可以专注于应用程序功能。只能从其中一个受支持的 HACL 层环境（C++ 或自动化对象）中使用 HACL 对象。这三个层可供大多数现代编程环境访问，例如 Microsoft® Visual C++®、Visual Basic® 和 Lotus® SmartSuite® 应用程序。

功能部件

在较高级别，HACL 提供许多在 EHLLAPI 级别不可用的功能。还有 EHLLAPI 的一些功能当前未在任何 HACL 类中实现。

HACL 的独特功能包括:

- 连接（会话）启动/停止功能
- 主机通信链路连接/断开的事件通知
- 连接（会话）启动/停止的事件通知
- 综合的错误捕获
- 特定于语言的错误消息文本生成
- 对连接（会话）的数量没有体系结构限制；目前，Z and I Emulator for Windows 限制为52
- 对多个并行连接（会话）和多线程应用程序的支持

- 主机表示空间的行/列寻址
- 表示空间的简化模型
- 字段和属性列表的自动生成
- 基于关键字的功能键字符串

HACL 中当前未实现的 EHLLAPI 功能包括：

- 结构化字段支持
- OIA 字符图像
- 锁定/解锁表示空间

会话标识

HACL 体系结构不限于 52 个会话。因此，单字符会话标识（如 EHLLAPI 中使用的会话标识）不恰当。HACL 使用连接句柄的概念，连接句柄是简单的 32 位值，对应用程序没有特殊意义。连接句柄唯一标识特定连接（会话）。可以跨线程和进程使用连接句柄来引用相同的连接。

所有需要引用特定连接的 HACL 对象和方法都接受连接句柄。此外，为了向后兼容并允许从仿真器用户界面（不显示句柄）引用，某些对象和方法也接受传统的会话标识。应用程序可以通过使用 ECLConnList 对象枚举连接来获取连接句柄。每个连接均由一个 ECLConnection 对象表示。ECLConnection::GetHandle 方法可用于检索与该特定连接关联的句柄。

强烈建议应用程序使用连接句柄，而不是连接名称（EHLLAPI 短会话标识）。HACL 的未来实现可能会阻止使用连接名称的应用程序访问超过 52 个会话。在某些情况下，可能需要使用该名称，例如当要求用户输入应用程序要使用的特定会话的名称时。在下面的 C++ 示例中，提供了会话的名称。那么，应用程序会在连接列表中查找连接，并为该会话创建 PS 和 OIA 对象：

```
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);
}
```

创建示例中显示的 PS 和 OIA 对象的第二种方法不是首选方法，因为使用的是会话名称而不是句柄。这将在代码的这一部分中创建一个隐式 52 会话限制。使用所示的第一个示例允许该代码段可用于任意数量的会话。

表示空间模型

HACL 表示空间模型比 EHLLAPI 更易于使用。HACL 表示空间由多多个平面组成，每个平面都包含一种类型的数据。这些平面包括：

- 文本
- 字段属性
- 颜色
- 扩展属性

这些平面大小都相同，并且在主机表示空间中的每个字符位置都包含一个字节。应用程序可以使用 ECLPS::GetScreen 方法获取任何感兴趣的平面。

此模型与 EHLLAPI 不同，在 EHLLAPI 中，文本和非文本表示空间数据通常在缓冲区中交错。应用程序必须设置 EHLLAPI 会话参数，才能指定要检索的数据类型，然后进行另一个调用，以将数据复制到缓冲区。HACL 模型允许应用程序在单个调用中获取感兴趣的数据，并且不同的数据类型不会在单个缓冲区中混合。

SendKey 接口

用于将击键发送到主机的 HACL 方法 (ECLPS::Sendkeys) 与 EHLLAPI SendKey 函数类似。但是，EHLLAPI 使用加密转义代码来表示非文本键，如 Enter、PF1 和 Backtab。ECLPS 对象使用带括号的关键字来表示这些击键。例如，以下 C++ 示例将在当前光标位置键入字符 ABC，然后按 Enter 键：

```
ECLPS  *PS;

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

事件

EHLLAPI 为应用程序提供一些接收特定事件异步通知的方法。但是，事件模型不一致（某些事件使用信号量，其他事件使用窗口系统消息），应用程序负责设置和管理事件线程。HACL 简化了所有事件处理，并使其对所有事件类型保持一致。应用程序不必显式创建多个执行线程，HACL 在内部处理线程。

但是，您必须注意，事件过程是在单独的执行线程上调用的。从事件过程访问动态应用程序数据时，必须同步对动态应用程序数据的访问。当应用程序注册事件时，将生成事件线程；当注销事件时，将终止事件线程。

PS 连接/断开连接和多线程

EHLLAPI 应用程序必须通过调用 ConnectPS 和 DisconnectPS EHLLAPI 函数来管理与不同会话的连接。应用程序必须仔细编码，以避免无限期地连接到会话，因为会话必须由所有 EHLLAPI 应用程序共享。在使用某些其他 EHLLAPI 函数之前，还必须确保应用程序已连接到会话。

HACL 不要求应用程序进行任何显式会话连接或断开连接。每个 HACL 对象在构造时都与特定的连接（会话）相关联。要访问不同的连接，应用程序只需要为每个连接创建对象。例如，以下示例会将击键 ABC 发送到会话 A，然后将 DEF 发送到会话 B，然后将 Enter 键发送到会话 A。在 EHLLAPI 程序中，应用程序必须连接/断开每个会话，因为它一次只能与一个会话交互。HACL 应用程序可以按所需的任何顺序使用对象：

```
ECLPS  *PSA, *PSB;

PSA = new ECLPS('A');
PSB = new ECLPS('B');

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

对于与多个连接（会话）交互的应用程序，这可以大大简化管理多个连接所需的代码。

除了单个工作会话外，EHLLAPI 还对应用程序的多线程性质进行了限制。当应用程序具有多个调用 EHLLAPI 接口的线程时，必须小心管理连接到表示空间和断开与表示空间的连接。即使有多个线程，应用程序也一次只能与一个会话交互。

ECLPS 不会对应用程序施加任何特定的多线程限制。应用程序可以与任意数量的线程上的任意数量的会话并发交互。

第 2 章. Host Access Class Library C++

此 C++ 类库提供主机连接的完全面向对象的抽象，其中包括：读取和写入主机表示空间（屏幕），枚举屏幕上的字段，读取操作员指示符区 (OIA) 以获取状态信息，访问和更新有关可视仿真器窗口的信息，传输文件，以及执行重要事件的异步通知。类库支持 Microsoft® Visual C++® 编译器。

主机访问类库 C++ 层由按类层次结构排列的许多 C++ 类组成。 图 2: 主机访问类对象 (on page 12) 说明了主机访问类库 C++ 层的 C++ 继承层次结构。在图中，每个对象都从其正上方的类继承。

图 2. 主机访问类对象

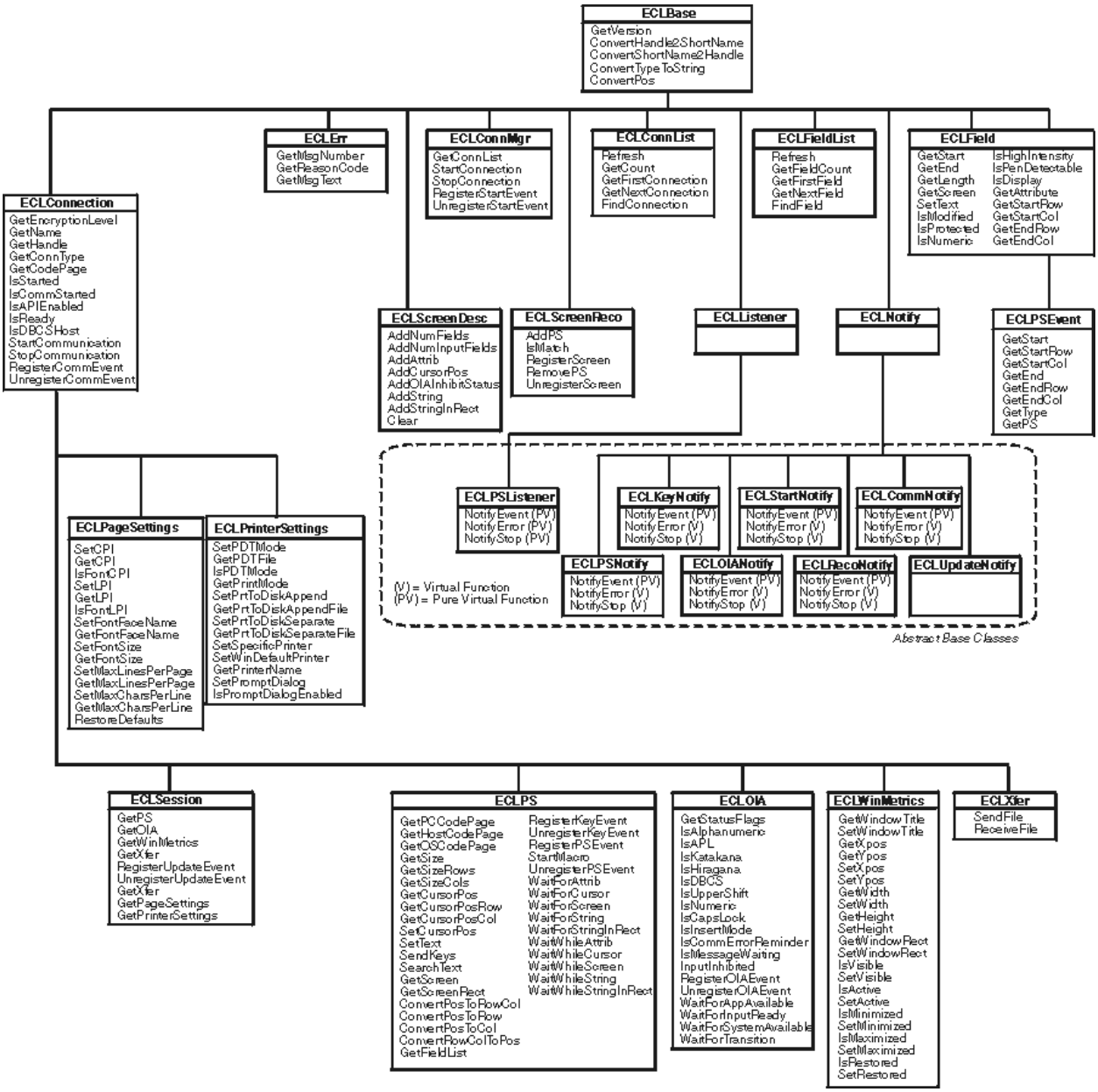


图 2: 主机访问类对象 (on page 12) 还显示每个类的所有成员函数。请注意，除了为每个类显示的函数之外，类还继承父类的所有函数。例如，函数 IsReady() 可用于 ECLSession, ECLPS、ECLOIA、ECLWinMetrics 和 ECLXfer 类。

以下各节简要介绍了每个类。有关更多详细信息，请参阅本章中的各个类描述。

ECLSAMPS.CPP 文件中提供了本章中显示的所有示例。此文件可用于使用任何支持的编译器编译和执行任何示例。

以下是主机访问类库 C++ 类的简要概述。每个类名都以 ECL 开头，ECL 是主机访问类库的通用前缀。

- 第 ECLBase 类 (on page 15) 页上的 ECLBase 是所有 ECL 对象的基类。它提供一些基本的实用程序方法，如连接名称和句柄的转换。由于所有 ECL 对象都继承自此类，因此这些方法可用于任何 ECL 对象。
- 第 ECLConnection 类 (on page 20) 页上的 ECLConnection 表示单个 Z and I Emulator for Windows 连接，并包含连接状态、连接类型（例如 3270 或 5250）以及连接的名称和句柄等连接信息。此类也是所有特定于连接的 ECL 对象（如 ECLPS 和 ECLOIA）的基类。
- 第 ECLConnList 类 (on page 34) 页上的 ECLConnList 包含创建对象或上次调用 Refresh 方法时存在的所有 Z and I Emulator for Windows 连接的列表。每个连接均由一个 ECLConnection 对象表示。
- 第 ECLConnMgr 类 (on page 42) 页上的 ECLConnMgr 使用 ECLConnList 对象枚举所有当前正在运行的 Z and I Emulator for Windows 连接（窗口）。还提供启动新连接和停止连接的方法。
- 第 ECLCommNotify 类 (on page 49) 页上的 ECLCommNotify 是一个通知类，应用程序可以使用该类在断开与主机的连接或连接到主机时收到通知。它可用于监控连接状态，并在连接意外断开时采取措施。
- 第 ECLErr 类 (on page 54) 页上的 ECLErr 提供从主机访问类库类返回运行时错误信息的方法。
- 第 ECLField 类 (on page 57) 页上的 ECLField 包含有关屏幕上单个字段的信息，例如字段属性、字段颜色、屏幕上的位置或长度。还提供了更新输入字段的方法。
- 第 ECLFieldList 类 (on page 74) 页上的 ECLFieldList 包含 ECLField 对象的集合。调用 Refresh 方法时，将检查当前主机屏幕，抽取字段列表，并用于构建 ECLField 对象列表。应用程序可以使用此集合来管理字段，而无需构建列表本身。
- 第 ECLKeyNotify 类 (on page 81) 页上的 ECLKeyNotify 是一个通知类，应用程序可以使用该类来获得击键事件的通知。应用程序可以过滤（删除）击键，将其替换为其他击键，或丢弃击键。
- 第 ECLListener 类 (on page 86) 页上的 ECLListener 是所有新 HACL 事件侦听器对象的基类。它为所有侦听器对象提供通用功能。
- 第 ECLOIA 类 (on page 86) 页上的 ECLOIA 提供对操作员状态信息的访问权，例如切换指示符、禁止输入的条件和通信错误。
- 第 ECLOIANotify 类 (on page 99) 页上的 ECLOIANotify 是抽象基类。应用程序创建从该类派生的对象以接收 OIA 更改通知。
- 第 ECLPS 类 (on page 102) 页上的 ECLPS 表示单个连接的表示空间（屏幕）。它包含以数据平面的形式获取屏幕内容副本的方法。每个平面均表示相关表示空间的特定方面，例如文本、字段属性和颜色属性。还提供用于搜索表示空间中的字符串、向主机发送键、获取和设置主机光标位置以及许多其他功能的方法。还提供 ECLFieldList 对象，可用于枚举屏幕上的字段列表。
- 第 ECLPSEvent 类 (on page 145) 页上的 ECLPSEvent 是一个事件对象，在更新表示空间时传递给 PS 事件侦听器。它包含有关事件的信息，包括导致更新的原因和屏幕已更新的部分。
- 第 ECLPSListener 类 (on page 150) 页上的 ECLPSListener 是抽象基类。应用程序创建从此类派生的对象，以接收表示空间更新事件，以及由 ECLPSEvent 对象提供的所有信息。
- 第 ECLPSNotify 类 (on page 153) 页上的 ECLPSNotify 是抽象基类。应用程序创建派生自此类的对象，以接收信息最少的表示空间更新通知。

- 第 ECLRecoNotify 类 (on page 155) 页上的 ECLRecoNotify 是抽象基类。应用程序创建从此类派生的对象，以接收屏幕识别通知。
- 第 ECLScreenDesc 类 (on page 159) 页上的 ECLScreenDesc 是用于描述单个主机屏幕的类。然后，屏幕描述类对象用于在所描述的主机屏幕出现时触发事件，或同步等待特定的主机屏幕。
- 第 ECLScreenReco 类 (on page 167) 页上的 ECLScreenReco 是一个类，用于收集一组屏幕描述对象，并在集合中的任何屏幕出现在表示空间中时生成异步事件。
- 第 ECLSession 类 (on page 173) 页上的 ECLSession 包含所有连接特定对象的集合。ECLSession 可用于为特定连接轻松创建一组完整的对象。
- 第 ECLStartNotify 类 (on page 181) 页上的 ECLStartNotify 是一个通知类，应用程序可以使用该类在启动或停止连接时收到通知。它可用于监控系统状态，并在连接意外关闭时采取措施。
- 第 ECLUpdateNotify 类 (on page 186) 页上的ECLUpdateNotify 是一个通知类，应用程序可以使用该类在更新主机屏幕或 OIA 时收到通知。
- 第 ECLWinMetrics 类 (on page 186) 页上的 ECLWinMetrics 表示正在运行仿真的物理窗口。提供了获取和设置窗口状态（最小值、最大值、已恢复）、窗口大小和可见性的方法。
- 第 ECLXfer 类 (on page 209) 页上的 ECLXfer 通过连接启动与主机之间的文件传输。
- 第 ECLPageSettings 类 (on page 215) 页上的 ECLPageSettings 会控制和检索仿真器会话**文件 > 页面设置**对话框的设置。
- 第 ECLPrinterSettings 类 (on page 227) 页上的 ECLPrinterSettings 会控制并检索仿真器会话**文件 > 打印机设置**对话框的设置。

构建 C++ ECL 程序

本节介绍有关如何构建使用 ECL 的 C++ 程序的机制。其中介绍了源代码的准备、编译和链接需求。

Microsoft Visual C++

以下各节介绍了如何准备、编译和链接使用 ECL 的 Microsoft® Visual C++ 应用程序。Z and I Emulator for Windows 当前支持 Microsoft® Visual C++ 编译器 V4.2 和更新版本。

源代码准备

使用 ECL 类的程序必须包含 ECL 头文件，以获取类定义和其他编译时信息。虽然可以仅包括应用程序所需的头文件子集，但为了简单起见，建议应用程序包括使用 ECLALL.HPP 文件的所有 ECL 头文件。

任何包含对 ECL 对象或定义的引用的 C++ 源文件，在第一次引用之前都应包含以下语句：

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

编译

必须指示编译器搜索包含 ECL 头文件的 ZIEWin 子目录。这可以使用 /I 编译器选项或 Developer Studio 的“项目设置”对话框来完成。

必须使用 /MT（用于可执行文件）或 /MD（用于 DLL）编译器选项编译应用程序，以进行多线程执行。

链接：

必须指示链接器包含 ECL 可链接库文件 (PCSECLVC.LIB)。这是通过在链接器命令行上指定库文件的全限定名称或通过使用“Developer Studio 项目设置”对话框来实现的。

正在执行

执行使用 ECL 的应用程序时，必须在系统路径中找到 ZIEWin 库。缺省情况下，会在 ZIEWin 安装期间将 ZIEWin 目录添加到系统路径。

ECLBase 类

ECLBase 是所有 ECL 对象的基类。它提供一些基本的实用程序方法，如连接名称和句柄的转换。由于所有 ECL 对象都继承自此类，因此这些方法可用于任何 ECL 对象。

应用程序不应直接创建此类的对象。

派生

无人

ECLBase 方法

下面展示了对 ECLBase 类有效的方法。

```
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

此方法会返回主机访问类库的版本。返回的值是十进制版本号乘以 100。例如，V1.02 将返回为 102。

原型

int GetVersion(void)

Parameters

无人

返回值

int

ECL 版本号乘以 100。

示例

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

此方法会返回指定的 ECL 连接句柄的名称（A-Z 或 a-z）。请注意，即使指定的连接不存在，此函数也可能会返回一个名称。

原型

char ConvertHandle2ShortName(long ConnHandle)

Parameters

long ConnHandle

ECL 连接的句柄。

返回值

字符型

范围 A–Z 或 a-z 内的 ECL 连接名称。

示例

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

此方法会返回具有指定名称的 ECL 连接的连接句柄。名称必须在范围 A–Z 或 a-z 内。请注意，即使指定的连接不存在，此函数也可能会返回一个句柄。

原型

char ConvertShortName2Handle(char Name)

Parameters

char Name

范围 A–Z 或 a-z 内的 ECL 连接名称。

返回值

字符型

ECL 连接的句柄。

示例

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

此方法会将由 ECLConnection::GetConnType() 返回的连接类型转换为以 null 结束的字符串。返回的字符串对语言不敏感。

ConnType	返回的字符串
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"
任何其他值	"UNKNOWN"

原型

void ConvertTypeToString(int ConnType,char *Buff)

Parameters

int ConnType

连接类型和必须是 ECLBASE.HPP 中定义的 HOSTTYPE_* 常量之一。

char *Buff

大小 TYPE_MAXSTRLEN 在 ECLBase.hpp 中定义并将返回字符串的缓冲区。

返回值

无人

示例

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

此方法是内嵌函数（宏），用于将 ECL 位置坐标转换为给定位置的行/列坐标和表示空间的宽度。对于已经知道（或假定）表示空间宽度的应用程序，此函数比使用 ECLPS::ConvertPosToRowCol() 更快。

原型

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

Parameters

ULONG Pos

要转换的线性位置坐标（输入）。

ULONG *Row

指向给定位置的返回行号的指针（输出）。

ULONG *Col

指向给定位置的返回列号的指针（输出）。

ULONG *PSCols

主机表示空间中的列数（输入）。

返回值

无人

示例

```
//---------------------------------------------------------------------
// 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 类

ECLConnection 包含给定连接的连接相关信息。此对象可以由应用程序直接创建，也可以由 ECLConnList 对象间接创建，或在创建从 ECLConnection 继承的任何对象（例如 ECLSession）时创建。

此对象的方法返回的信息在调用方法时是最新的。

ECLSession、ECLPS、ECLOIA、ECLWinMetrics 和 ECLXfer 将继承 ECLConnection。

派生

ECLBase > ECLConnection

ECLConnection 方法

下面展示了对 ECLConnection 类有效的方法。

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 构造函数

此方法会根据连接名称或句柄构造 ECLConnection 对象。

原型

ECLConnection(long ConnHandle)

ECLConnection(char ConnName)

Parameters

long ConnHandle

用于创建连接对象的连接的句柄。

char ConnName

用于创建连接对象的连接的名称（A-Z 或 a-z）。

返回值

无人

示例

```
//---------------------------------------------------------------------
// 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 析构函数

此方法会破坏 ECLConnection 对象。

原型

~ECLConnection()

Parameters

无人

返回值

无人

示例

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

此方法会返回配置了连接的主机代码页。

原型

unsigned int GetCodePage()

Parameters

无人

返回值

unsigned int

连接的主机代码页。

示例

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

此方法会返回连接的句柄。此句柄唯一标识连接，并可用于需要连接句柄的其他 ECL 函数。

原型

long GetHandle()

Parameters

无人

返回值

long

ECLConnection 对象的连接句柄。

示例

以下示例展示了如何返回连接列表中第一个连接的句柄。

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

此方法会返回连接类型。此连接类型可能会随着时间推移而更改（例如，可以为其他主机重新配置连接）。应用程序不应假定连接类型是固定的。有关返回的连接类型，请参阅下文。

注： ECLBase::ConvertTypeToString 函数会将连接类型转换为以 null 结束的字符串。

原型

int GetConn Type()

Parameters

无人

返回值

int

连接类型常量（来自 HOSTBASE.HPP 的 HOSTTYPE_*）。下表显示了返回的值及其含义。

返回值	意义
HOSTTYPE_3270DISPLAY	3270 显示
HOSTTYPE_3270PRINTER	3270 打印机
HOSTTYPE_5250DISPLAY	5250 显示
HOSTTYPE_5250PRINTER	5250 打印机
HOSTTYPE_VT	ASCII VT 显示
HOSTTYPE_UNKNOWN	未知连接类型

示例

以下示例展示了如何使用 GetConnType 方法返回连接类型。

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

此方法会返回连接的连接名称（A-Z 或 a-z 中的单个字母字符）。此名称也与 EHLLAPI 会话标识对应。

原型

char GetName()

Parameters

无人

返回值

字符型

连接短名称。

示例

以下示例展示了如何使用 GetName 方法返回连接名称。

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

此方法返回当前连接的加密级别。

原型

int GetEncryptionLevel()

Parameters

无人

返回值

int

加密级别常量。下表显示了返回的值及其含义。

返回值	意义
ENCRYPTION_NONE	无加密
ENCRYPTION_40BIT	40 位加密
ENCRYPTION_56BIT	56 位加密
ENCRYPTION_128BIT	128 位加密
ENCRYPTION_168BIT	168 位加密
ENCRYPTION_NOKEY	不使用密钥加密

示例

以下示例展示了如何使用 GetEncryptionLevel 方法返回加密级别。

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

此方法会指示连接是否已启动。启动的连接可能连接到主机，也可能不连接到主机。使用 IsCommStarted 函数确定连接当前是否连接到主机。

原型

BOOL IsStarted()

Parameters

无人

返回值

BOOL

如果连接已启动，则返回 TRUE；如果连接未启动，则返回 FALSE。

示例

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

此方法会指示连接当前是否连接到主机（例如，它会指示对于连接，主机通信是否处于活动状态）。如果未启动连接，则此函数会返回 FALSE 值（请参阅 IsStarted (on page 28)）。

原型

BOOL IsCommStarted()

Parameters

无人

返回值

BOOL

如果连接已连接到主机，则返回 TRUE 值；如果连接未连接到主机，则返回 FALSE 值。

示例

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

此方法会指示连接是否已启用 API。未启用 API 的连接不能与主机访问类库一起使用。如果未启动连接，则此函数会返回 FALSE 值。

原型

BOOL IsAPIEnabled()

Parameters

无人

返回值

BOOL

如果 API 已启用，则返回 TRUE 值；如果 API 未启用，则返回 FALSE 值。

示例

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

此方法会指示连接已就绪，这意味着连接已启动、已连接并已启用 API。此函数比调用 IsStarted、IsCommStarted 和 IsAPIEnabled 更快、更轻松。

原型

BOOL IsReady()

Parameters

无人

返回值

BOOL

如果连接已启动、为 CommStarted 且已启用 API，则返回 TRUE；否则返回 FALSE。

示例

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

此方法会将 ZIEWin 仿真器连接到主机数据流。这与进入 ZIEWin 仿真器“通信”菜单并选择“连接”具有相同的效果。

原型

void StartCommunication()

Parameters

无人

返回值

无人

示例

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

此方法会断开 ZIEWin 仿真器与主机数据流的连接。这与进入 ZIEWin 仿真器“通信”菜单并选择“断开连接”具有相同的效果。

原型

void StopCommunication()

Parameters

无人

返回值

无人

示例

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

此成员函数会注册一个应用程序对象，以接收所有通信链路连接/断开连接事件的通知。要使用此函数，应用程序必须创建从 ECLCommNotify 类派生的对象。接着将指向该对象的指针传递到此注册函数。实现限制：应用程序只能为通信事件通知注册一个对象。

使用此函数注册通知对象后，每当与主机的连接通信链接连接或断开连接时，都会调用该对象。对象将收到所有通信事件的通知，无论这些事件是由 StartCommunication() 函数导致还是由用户显式导致。不能将此事件与启动或停止新 ZIEWin 连接时触发的连接启动/停止事件相混淆。

可选的 InitEvent 参数会导致在注册对象时生成初始事件。这对于将事件对象与通信链路的当前状态同步非常有用。如果将 InitEvent 指定为 FALSE，则注册对象时不会生成初始事件。此参数的缺省值为 TRUE。

应用程序必须在破坏通知对象之前调用 UnregisterCommEvent()。如果注册此对象的 ECLConnection 对象被破坏，则将自动注销此对象。

有关更多信息，请参阅 ECLCommNotify 类 (on page 49) 的描述。

原型

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

Parameters

ECLCommNotify *NotifyObject

指向从 ECLCommNotify 类派生的对象的指针。

BOOL InitEvent

生成具有当前状态的初始事件。

返回值

无人

示例

请参阅 ECLCommNotify 类 (on page 49) 以获取 ECLConnection::RegisterCommEvent 示例。

UnregisterCommEvent

此成员函数会注销先前使用 RegisterCommEvent() 函数为通信事件注册的应用程序对象。如果未首先调用此函数来将其注销，则不能破坏已注册的应用程序通知对象。如果当前未注册通知对象，或者注册的对象不是传入的 NotifyObject，则此函数不执行任何操作（不抛出错误）。

注销通知对象后，将调用其 NotifyStop() 成员函数。

有关更多信息，请参阅 ECLCommNotify 类 (on page 49) 的描述。

原型

void UnregisterCommEvent(ECLCommNotify *NotifyObject)

Parameters

ECLCommNotify *NotifyObject

这是当前注册的应用程序通知对象。

返回值

无人

示例

请参阅 ECLCommNotify 类 (on page 49) 以获取 ECLConnection::UnregisterCommEvent 示例。

ECLConnList 类

ECLConnList 会获取有关给定机器上所有主机连接的信息。ECLConnList 对象包含系统中当前已知的所有连接的集合。

ECLConnList 对象包含 ECLConnection 对象的集合。集合的每个元素都包含有关单个连接的信息。此列表中的连接可能处于任何状态（例如，已停止或已断开连接）。所有已启动的连接都将显示在此列表中。ECLConnection 对象包含连接的状态。

此列表反映的是该对象创建时或最近一次调用 Refresh 方法时连接集的快照。列表并不随着连接的启动与停止而动态地进行更新。应用程序可以使用 ECLConnMgr 对象的 RegisterStartEvent 成员收到连接启动和停止事件的通知。

ELCConnList 对象可以由应用程序直接创建，也可以通过创建 ECLConnMgr 对象间接创建。

派生

ECLBase > ECLConnList

使用注意事项

ECLConnList 对象提供当前连接的静态快照。在构造 ECLConnList 对象时，将自动调用 Refresh 方法。如果在构造后立即使用 ECLConnList 对象，则该对象包含当时连接列表的准确表示。但是，如果自构造起已过了一段时间，则应在开始访问前在 ECLConnList 对象中调用 Refresh 方法。

应用程序可以使用 GetFirstConnection 和 GetNextConnection 方法迭代集合。GetFirstConnection 和 GetNextConnection 返回的对象指针仅在调用 Refresh 成员或破坏 ECLConnList 对象之前有效。应用程序可以使用 FindConnection 函数在列表中查找感兴趣的特定连接。与 GetNextConnection 类似，返回的指针仅在破坏下一个 Refresh 或 ECLConnList 对象之前有效。

连接列表中的连接顺序未定义。应用程序不应对列表顺序作出任何假设。在调用 Refresh 函数之前，列表中的连接顺序不会更改。

创建 ECLConnMgr 对象时，会自动创建 ECLConnList 对象。但是，可以在不使用 ECLConnMgr 对象的情况下创建 ECLConnList 对象。

ECLConnList 方法

下节介绍了对 ECLConnList 类有效的方法。

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

ECLConnList 构造函数

此方法会创建 ECLConnList 对象，并使用当前连接列表对其进行初始化。

原型

ECLConnList();

Parameters

无人

返回值

无人

示例

```
//-------------------------------------------------------------------
// 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 析构函数

此方法会破坏 ECLConnList 对象。

原型

~ECLConnList()

Parameters

无人

返回值

无人

示例

```
//-------------------------------------------------------------------
// 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 方法会返回指向 ECLConnList 集合中第一个连接信息对象的指针。请参阅 ECLConnection 类 (on page 20) 以获取其内容的详细信息。调用 ECLConnList Refresh 成员或破坏 ECLConnList 对象时，返回的指针将变为无效。应用程序不应删除返回的对象。如果列表中没有连接，则返回 NULL。

原型

ECLConnection *GetFirstConnection()

Parameters

无人

返回值

ECLConnection *

指向列表中的第一个 ECLConnection 对象的指针。如果列表中没有连接，则返回 null。

示例

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

此方法会返回指向 ECLConnList 集合中的列表中给定连接的下一个连接信息对象的指针。应用程序提供指向先前由此函数或 GetFirstConnection 返回的连接的指针。请参阅 ECLConnection 类 (on page 20) 以获取其内容的详细信息。在下一次 ECLConnList Refresh() 调用或 ECLConnList 对象被破坏后，返回的指针无效。如果试图读取列表结束之后的内容，则返回 NULL 指针。对此方法的连续调用（在每个调用上提供前一个指针）会在连接列表上迭代。返回最后一个连接后，后续调用将返回 NULL 指针。列表中的第一个连接可以通过为上一个连接提供 NULL 来获得。

原型

ECLConnection *GetNext Connection (ECLConnection *Prev)

Parameters

ECLConnection *Prev

之前调用此函数 GetFirstConnection() 时返回的指针或 NULL。

返回值

ECLConnection *

这是指向下一个 ECLConnection 对象的指针，列表结束时则为 NULL。

示例

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

此方法会在当前连接列表中搜索指定的连接。可以通过句柄或名称指定所需的连接。FindConnection 方法有两个签名。如果找到指定的连接，则返回指向 ECLConnection 对象的指针。如果指定的连接不在列表中，则返回 NULL。此函数不会自动刷新列表；如果自构建或刷新列表以来已启动新连接，则找不到该连接。返回的指针指向由 ECLConnList 对象维护的连接列表中的对象。在下一次 ECLConnList::Refresh 调用或 ECLConnList 对象被破坏后，返回的指针无效。

原型

ECLConnection *FindConnection(Long ConnHandle),

ECLConnection *FindConnection(char ConnName)

Parameters

Long ConnHandle

要在列表中查找的连接的句柄。

char ConnName

要在列表中查找的连接的名称。

返回值

ECLConnection *

指向请求的 ECLConnection 对象的指针。如果指定的连接不在列表中，则返回 NULL。

示例

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

此方法会返回 ECLConnList 集合中当前的连接数。

原型

ULONG GetCount()

Parameters

无人

返回值

ULONG

集合中的连接数。

示例

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

Refresh

此方法会使用系统中所有当前已知连接的列表更新 ECLConnList 集合。以前由 GetNextConnection、GetFirstConnection 和 FindConnection 返回的所有指针都将变为无效。

原型

void Refresh()

Parameters

无人

返回值

无人

示例

```
//-------------------------------------------------------------------
// 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 类

ECLConnMgr 会管理给定机器上的所有 Z and I Emulator for Windows 连接。它提供与连接管理相关的方法，例如启动和停止连接。它还会创建 ECLConnList 对象，以枚举系统上所有已知连接的列表（请参阅 ECLConnList 类 (on page 34)）。

派生

ECLBase > ECLConnMgr

ECLConnMgr 方法

下面展示了对 ECLConnMgr 类有效的方法。

ECLConnMgr()

~ECLConnMgr()

ECLConnList * GetConnList()

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

ECLConnMgr 构造函数

此方法会构造 ECLConnMgr 对象。

原型

ECLConnMgr()

Parameters

无人

返回值

无人

示例

```
//---------------------------------------------------------------------
// 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 析构函数

此方法会破坏 ECLConnMgr 对象。

原型

~ECLConnMgr()

Parameters

无人

返回值

无人

示例

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

此方法会返回指向 ECLConnList 对象的指针。请参阅 ECLConnList 类 (on page 34) 以获取更多信息。在破坏 ECLConnMgr 对象时，将破坏 ECLConnList 对象。

原型

ECLConnList * GetConnList()

Parameters

无人

返回值

ECLConnList *

指向 ECLConnList 对象的指针

示例

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

此方法会启动新的 Z and I Emulator for Windows 仿真器连接。ConfigParms 字符串包含 使用注意事项 (on page 45) 下所述的连接配置信息。

原型

void StartConnection(char *ConfigParms)

Parameters

char *ConfigParms

以 null 结束的连接配置字符串。

返回值

无人

使用注意事项

连接配置字符串特定于实现。主机访问类库的不同实现可能要求在配置字符串中具有不同的格式或信息。此调用本质上是异步的；当此调用返回时，可能尚未启动新连接。应用程序可以使用 RegisterStartEvent 函数在连接启动时收到通知。

对于 Z and I Emulator for Windows，配置字符串有以下格式：

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

可选参数用方括号 [] 括起。参数间至少用一个空格分隔。参数可以是大写、小写或大小写混合，并且可以按任意顺序显示。每个参数的含义如下所示：

PROFILE=<filename>

命名包含连接配置信息的 Z and I Emulator for Windows 工作站概要文件（.WS 文件）。此参数不是可选的；必须提供概要文件名称。如果文件名中包含空格，则必须用双引号将名称引起来。<filename> 值可以是不带扩展名的概要文件名，带 .WS 扩展名的概要文件名，或全限定概要文件名路径。

CONNNAME=<c>

指定新连接的连接名称（EHLLAPI 短会话标识）。此值必须是单个字母字符（A-Z 或 a-z）。如果未指定此值，则将自动分配下一个可用的连接名称。如果指定名称的连接已存在，则会抛出错误 (ERRMAJ_INVALID_SESSION)。

WINSTATE=<MAX|MIN|RESTORE|HIDE>

指定仿真器窗口的初始状态。如果未指定此参数，则缺省值为 RESTORE。

注： 由于此调用的异步性质，此函数可能会返回而不会出错，但连接无法启动。例如，如果在短时间内使用相同的连接名称启动两个连接，则第二个 StartConnection 不会失败，因为第一个连接尚未启动。但是，当第二个连接最终尝试注册其名称时，它无法启动，因为该名称已被第一个连接使用。为了最大限度地减少这种可能性，应在启动连接时尽可能不指定 CONNNAME 参数。

示例

以下是 StartConnection 方法的示例。

```
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

此方法会停止（终止）连接句柄标识的仿真器连接。请参阅 使用注意事项 (on page 47) 以获取 StopParms 字符串的内容。

原型

void StopConnection(Long ConnHandle, char *StopParms)

Parameters

Long ConnHandle

要停止的连接的句柄。

char * StopParms

以 null 结束的连接停止参数字符串。

返回值

无人

使用注意事项

连接停止参数字符串特定于实现。主机访问类库的不同实现可能需要参数字符串的不同格式和内容。对于 Z and I Emulator for Windows，字符串采用下列格式：

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

可选参数用方括号 [] 括起。参数间至少用一个空格分隔。参数可以是大写、小写或大小写混合，并且可以按任意顺序显示。SAVEPROFILE 参数的含义如下所示：

SAVEPROFILE=<YES|NO|DEFAULT> 控制将当前连接配置保存回工作站概要文件（.WS 文件）。这会导致使用您在连接中对配置所做的任何更改来更新概要文件。如果指定 NO，将停止连接且不更新概要文件。如果指定 YES，将停止连接并使用当前（可能已更改的）配置更新概要文件。如果指定 DEFAULT，则更新选项由**文件 -> 退出时保存**仿真器菜单选项控制。如果未指定该参数，则使用 DEFAULT。

示例

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

此方法会注册一个应用程序对象，以接收所有连接启动和停止事件的通知。要使用此函数，应用程序必须创建从 ECLStartNotify 类派生的对象。接着将指向该对象的指针传递到此注册函数。实现限制：一个应用程序只能注册一个对象来用于连接启动或停止通知。

使用此函数注册通知对象后，无论何时启动或停止 Z and I Emulator for Windows 连接，都会调用该对象。该对象会接收所有连接的通知，而无论这些连接是由 StartConnection 函数启动还是由您显式启动。不应将此事件与启动/停止通信事件混淆，通信事件是在连接连接主机系统或断开与主机系统的连接时触发的。

请参阅 ECLStartNotify 类 (on page 181) 以获取更多信息。

原型

void RegisterStartEvent(ECLStartNotify *NotifyObject)

Parameters

ECLStartNotify *NotifyObject

指向从 ECLStartNotify 类派生的对象的指针。

返回值

无人

示例

```
//-------------------------------------------------------------------
// ECLConnMgr::RegisterStartEvent
//
// See ECLStartNotify 类 (on page 181) for example of this method.
//-------------------------------------------------------------
```

UnregisterStartEvent

此方法会使用 RegisterStartEvent 函数注销先前为连接启动或停止事件注册的应用程序对象。如果未首先调用此函数来将其注销，则不能破坏已注册的应用程序通知对象。如果当前未注册通知对象，或者注册的对象不是传入的 NotifyObject，则此函数不执行任何操作（不抛出错误）。

注销通知对象时，将调用其 NotifyStop 方法。

请参阅 ECLStartNotify 类 (on page 181) 以获取更多信息。

原型

void UnregisterStartEvent(ECLStartNotify *NotifyObject)

Parameters

无人

返回值

无人

示例

```
//----------------------------------------------------------------------
// ECLConnMgr::UnregisterStartEvent
//
// See ECLStartNotify 类 (on page 181) for example of this method.
//----------------------------------------------------------------------
```

ECLCommNotify 类

ECLCommNotify 是抽象基类。应用程序无法直接创建此类的实例。要使用此类，应用程序必须定义其自己派生自 ECLCommNotify 的类。应用程序必须在其派生类中实现 NotifyEvent() 成员函数。它还可以选择性地实现 NotifyError() 和 NotifyStop() 成员函数。

ECLCommNotify 类用于允许应用程序在 ZIEWin 连接上发生通信连接/断开连接事件时收到通知。每当 ZIEWin 连接（窗口）与主机系统连接或断开连接时，都会生成连接/断开连接事件。

要收到通信连接/断开连接事件的通知，应用程序必须执行以下步骤：

1. 定义派生自 ECLCommNotify 的类。
2. 实现派生类并实现 NotifyEvent() 成员函数。
3. （可选）实现 NotifyError() 函数和/或 NotifyStop() 函数。
4. 创建派生类的实例。
5. 使用 ECLConnection::RegisterCommEvent() 函数注册实例。

显示的示例演示了如何执行此操作。完成上述步骤后，每次连接的通信链路连接或断开与主机的连接时，都将调用应用程序 NotifyEvent() 成员函数。

如果在事件生成过程中检测到错误，则使用 ECLErr 对象调用 NotifyError() 成员函数。发生错误后，可能会继续生成事件，也可能不会继续生成，具体取决于错误的性质。当事件生成终止时（由于错误，通过调用 ECLConnection::UnregisterCommEvent，或由于破坏 ECLConnection 对象），将调用 NotifyStop() 成员函数。但是，事件通知会被终止，NotifyStop() 成员函数总是被调用，并且应用程序对象会被注销。

如果应用程序不提供 NotifyError() 成员函数的实现，则使用缺省实现（向用户显示简单的消息框）。应用程序可以通过在应用程序派生类中实现 NotifyError() 函数来覆盖缺省行为。同样，如果应用程序不提供此函数（缺省行为是不执行任何操作），则使用缺省 NotifyStop() 函数。

请注意，应用程序还可以选择为派生类提供自己的构造函数和析构函数。如果应用程序要在类中存储某些特定于实例的数据，并将该信息作为参数传递到构造函数上，则此选项非常有用。例如，当发生通信事件时，应用程序可能希望将消息发布到应用程序窗口。应用程序不是将窗口句柄定义为全局变量（因此它对 NotifyEvent() 函数可见），而是可以为类定义构造函数，该类接受窗口句柄并将其存储在类成员数据区域中。

应用程序在注册接收事件时不能破坏通知对象。

实现限制：当前，ECLConnection 对象仅允许注册一个通知对象用于通信事件通知。如果已为该 ECLConnection 对象注册了通知对象，则 ECLConnection::RegisterCommEvent 将抛出错误。

派生

ECLBase > ECLNotify > ECLCommNotify

示例

```
//---------------------------------------------------------------------
// 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 方法

下节介绍了对 ECLCommNotify 类有效的方法：

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

此方法是“纯虚拟”成员函数（应用程序必须在派生自 ECLCommNotify 的类中实现此函数）。每当连接开始或停止并且为开始/停止事件注册了对象时，都会调用此函数。如果通信链路已连接，则 Connected BOOL 为 TRUE；如果通信链路未连接到主机，则为 FALSE。

原型

virtual void NotifyEvent (ECLConnection *ConnObj, BOOL Connected)

Parameters

ECLConnection *ConnObj

这是指向发生事件的 ECLConnection 对象的指针。

BOOL 已连接

如果连接了通信链路，则为 TRUE；如果已断开连接，则为 FALSE。

返回值

无人

NotifyError

每当 ECLConnection 对象在事件生成过程中检测到错误时，都会调用此方法。错误对象包含有关错误的信息（请参阅 ECLErr 类 (on page 54)）。根据错误的性质，出现错误后可能会继续生成事件。如果事件生成因错误而停止，则将调用 NotifyStop() 函数。应用程序可以选择实现此函数或允许 ECLCommNotify 基类处理错误。基类将使用 ECLErr::GetMsgText() 函数提供的文本在消息框中显示错误。如果应用程序在其派生类中实现此函数，它将覆盖基类函数。

原型

virtual void NotifyError (ECLConnection *ConnObj, ECLErr ErrObject)

Parameters

ECLConnection *ConnObj

这是指向发生错误的 ECLConnection 对象的指针。

ECLErr ErrObject

这是描述错误的 ECLErr 对象。

返回值

无人

NotifyStop

因任何原因（例如，由于错误条件或调 ECLConnection::UnregisterCommEvent 等）停止事件生成时，将调用此方法。

实现说明：原因代码当前未使用，因此将为零。

原型

virtual void NotifyStop (ECLConnection *ConnObj, int Reason)

Parameters

ECLConnection *ConnObj

这是指向正在停止通知的 ECLConnection 对象的 ptr。

int Reason

这未使用（零）。

返回值

无人

ECLErr 类

ECLErr 类提供从主机访问类库类返回运行时错误信息的方法。在错误情况下，将创建 ECLErr 对象并填充错误和诊断信息。接着，ECLErr 对象被作为 C++ 异常抛出。然后可以从捕获的 ECLErr 对象查询错误和诊断信息。

应用程序不应直接创建或抛出 ECLErr 对象。

派生

ECLBase > ECLErr

ECLErr 方法

下节介绍了对 ECLErr 类有效的方法。

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

GetMsgNumber

此方法会返回创建此 ECLErr 对象时设置的消息编号。错误消息编号在 ERRORIDS.HPP 中介绍。

原型

const int GetMsgNumber()

Parameters

无人

返回值

const int

错误消息编号。

示例

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

此方法会从 ECLErr 对象获取原因码（有时称为辅助或次要返回码）。此代码通常用于调试和诊断目的。主机访问类库的未来版本中可能会发生更改，因此不应以编程方式使用。原因码的描述可在 ERRORIDS.HPP 中找到。

原型

const int GetReasonCode()

Parameters

无人

返回值

const int

ECLErr 原因代码。

示例

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

此方法会返回与用于创建此 ECLErr 对象的错误代码关联的消息文本。消息文本将以当前安装 Z and I Emulator for Windows 的语言返回。

注： 删除 ECLErr 对象后，返回的指针无效。

原型

const char *GetMsgText()

Parameters

无人

返回值

char *

与属于此 ECLErr 对象的错误代码关联的消息文本。

示例

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

使用注意事项

将从 Z and I Emulator for Windows 消息工具中检索消息文本。

ECLField 类

ECLField 包含 ECLPS 对象包含的 ECLFieldList 对象中给定字段的信息。应用程序不能直接创建此类型的对象。ECLField 对象由 ECLFieldList 对象间接创建。

ECLField 对象描述主机表示空间的单个字段。它具有查询字段的各种属性和更新字段文本（例如，修改字段文本）的方法。不能修改字段属性。

派生

ECLBase > ECLField

副本构造函数和赋值运算符

此对象支持复制-构造和分配。这对于想要在主机屏幕上轻松捕获字段以供以后处理的应用程序非常有用。应用程序不需要分配文本缓冲区并复制字段的字符串内容，只需将字段存储在专用 ECLField 对象中即可。存储的副本会保留 ECLField 对象的所有函数，包括字段的文本值、属性、起始位置、长度等。例如，假设某个应用程序要捕获屏幕的第一个输入字段。 表 1: 复制-构造和分配示例 (on page 58) 显示了实现这一目标的两种方法。

表 1. 复制-构造和分配示例

将字段另存为字符串

```
#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
      SavePtr=malloc(Fld->Length() + 1);
      Fld->GetScreen(SavePtr, Fld->Length()+1);
  }

  // We now have captured the field text
```

将字段另存为 ECLField 对象

```
#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
      SaveFld = *Fld;
  }

  // We now have captured the field text
  // including text, position, attrib
```

使用 ECLField 对象而不是字符串来存储字段有几个优点：

- ECLField 对象执行字段文本缓冲区的所有存储管理；应用程序不必分配或自由文本缓冲区，也不必计算所需缓冲区的大小。
- 保存的字段保留原始字段的所有特性，包括其属性和起始位置。除了 SetText() 之外，所有常用的 ECLField 成员函数都可用于存储的字段。请注意，存储的字段是原始字段的副本。当主机屏幕更改或调用 ECLFieldList::Refresh() 函数时，其值不会更新。因此，该字段可以存储并在以后的应用程序中使用。

还为字符串和长整型值类型提供了分配运算符覆盖。这些覆盖使您可以轻松地为未受保护的字段分配新的字符串或数值。例如，以下设置了屏幕的前两个输入字段：

```
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;
```

注释：

1. 由复制-构造或分配初始化的 ECLField 对象是原始字段对象的只读副本。SetText() 方法对于此类对象无效，将导致抛出 ECLErr 异常。由于对象是副本，因此在更新或删除原始字段对象时不会更新或删除这些对象。应用程序负责删除不再需要的字段对象副本。
2. 调用未初始化的 ECLField 对象上的任何方法都将返回未定义的结果。
3. 应用程序创建的 ECLField 对象可以重新分配任意多次。
4. 只能从另一个 ECLField 对象、字符串或长整型值进行分配。将任何其他数据类型分配给 ECLField 对象无效。
5. 如果对当前属于 ECLFieldList 一部分的 ECLField 对象进行了分配，则只会更新字段的文本值。仅当字段对象是未受保护的字段时，才允许执行此操作。例如，以下内容将通过复制第一个输入字段中的值来修改屏幕的第二个输入字段：

   ```
   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;
   ```

   由于 Fld2 是 ECLFieldList 的一部分，因此上述分配与以下内容相同：

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

   请注意，如果 Fld2 受保护，这将抛出 ECLErr 异常。另请注意，仅更新 Fld2 的文本，而不更新其属性、位置或长度。
6. 为字段对象指定字符串等同于调用 SetText() 方法。您也可以指定数值，而无需先转换为字符串：

   ```
   *Field = 1087;
   ```

   这等同于将数字转换为字符串，然后调用 SetText() 方法。

ECLField 方法

下节介绍了对 ECLField 类有效的方法。

```
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()
```

以下方法对 ECLField 类有效：

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

GetStart

此方法会返回字段第一个字符在表示空间中的位置。GetStart 方法有两个签名。ULONG GetStart 以线性值的形式返回位置，其中表示空间的左上角为“1”。void GetStart(ULONG *Row, ULONG *Col) 以行和列坐标的形式返回位置。

原型

ULONG GetStart(),

void GetStart(ULONG *Row, ULONG *Col)

Parameters

ULONG *Row

此输出参数是指向要更新的行值的指针。

ULONG *Col

此输出参数是指向要更新的列值的指针。

返回值

ULONG

表示为线性数组的表示空间中的位置。

示例

以下示例展示了如何返回字段第一个字符在表示空间中的位置。

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

此方法会返回 ECLFieldList 集合中给定字段的起始行位置，用于与 ECLPS 对象关联的连接。

原型

ULONG GetStartRow()

Parameters

无人

返回值

ULONG

这是给定字段的起始行。

示例

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

此方法会返回 ECLFieldList 集合中给定字段的起始列位置，用于与 ECLPS 对象关联的连接。

原型

ULONG GetStartCol()

Parameters

无人

返回值

ULONG

这是给定字段的起始列。

示例

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

此方法会返回字段最后一个字符在表示空间中的位置。GetEnd 方法有两个签名。ULONG GetEnd 以线性值的形式返回位置，其中表示空间的左上角为“1”。void GetEnd(ULONG *Row, ULONG *Col) 以行和列坐标的形式返回位置。

原型

ULONG GetEnd()

void GetEnd(ULONG *Row, ULONG *Col)

Parameters

ULONG *Row

此输出参数是指向要更新的行值的指针。

ULONG *Col

此输出参数是指向要更新的列值的指针。

返回值

ULONG

表示为线性数组的表示空间中的位置。

示例

以下示例展示了如何返回字段最后一个字符在表示空间中的位置。

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

此方法会返回字段的结束行位置。

原型

ULONG GetEndRow()

Parameters

无人

返回值

ULONG

这是给定字段中的结束行。

示例

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

此方法会返回字段的结束列位置。

原型

ULONG GetEndCol()

Parameters

无人

返回值

ULONG

这是给定字段中的结束行。

示例

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

此方法会返回字段的名称。长度包括整个字段，即使它跨越表示空间的多行也是如此。它不包括启动字段的字段属性字符。

原型

ULONG GetLength()

Parameters

无人

返回值

ULONG

字段的长度。

示例

以下示例展示了如何返回字段的长度。

```
/--------------------------------------------------------------------
// 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 方法使用字段中的数据填充应用程序提供的缓冲区。复制到缓冲区的数据类型可通过可选的 Plane 参数进行选择。缺省情况下返回文本平面数据。返回的数据是创建此字段对象时就存在的字段；如果自调用 ECLFieldList::Refresh 函数以来该字段已更新，则不会反映该字段的当前内容。

所返回数据的长度是字段的长度（请参阅 GetLength (on page 67)）。复制 TextPlane 时，将在最后一个数据字节之后添加一个附加的以 null 结束的字节。因此，在获取文本平面时，应用程序应提供至少比字段长度多 1 个字节的缓冲区。如果应用程序缓冲区太小，返回的数据将被截断。复制到应用程序缓冲区的字节数作为函数结果返回（不包括文本平面副本的 null 终止符）。

无法使用此函数获取 FieldPlane。ECLField::GetAttribute 可用于获取字段属性值。

原型

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

Parameters

char * Buff

指向要使用字段数据填充的应用程序缓冲区的指针。

ULONG BuffLen

应用程序缓冲区的长度。

PS_PLANE Plane

可选参数。指示要检索的字段数据平面的枚举。必须是 TextPlane、ColorPlane 或 ExtendedFieldPlane 中的一个。

返回值

ULONG

复制到应用程序缓冲区的字节数，不包括 TextPlane 数据的尾随 null 字符。

示例

以下示例展示了如何返回指向 Plane 参数指示的字段数据的指针。

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

此方法会使用作为文本传递的字符串填充表示空间中的给定字段。如果文本超过字段的长度，文本将被截断。如果文本短于字段，则字段将采用 null 填充。

原型

void SetText(char *text)

Parameters

char *text

要在字段中设置的以 null 结束的字符串。

返回值

无人

示例

以下示例展示了如何使用作为文本传递的字符串填充表示空间中的给定字段。

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

此方法确定表示空间中的给定字段是否具有特定属性。如果字段具有属性，则方法返回 TRUE 值；如果字段没有属性，则返回 FALSE 值。

原型

BOOL IsModified()

BOOL IsProtected()

BOOL IsNumeric()

BOOL IsHighIntensity()

BOOL IsPenDetectable()

BOOL IsDisplay()

Parameters

无人

返回值

BOOL

如果属性存在，则返回 TRUE 值；如果属性不存在，则返回 FALSE 值。

示例

以下示例展示了如何确定给定字段是否具有属性。

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

此方法会返回字段的属性。返回的值包含每个可能的字段属性的位标识（已修改、受保护、数字、高亮度、光笔和显示）。请参阅 ECL 平面 - 格式和内容 (on page 414) 以获取有关这些位的更多详细信息。为每种类型的属性提供了一种方法（例如 IsModified 或 IsHighIntensity）。此方法可用于在单个调用中获取完整的属性信息。

原型

unsigned char GetAttribute()

Parameters

无人

返回值

unsigned char

字段的属性位。

示例

以下示例展示了如何返回字段的属性。

```
/ 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 类

ECLFieldList 类会对主机表示空间中的字段列表执行操作。应用程序不应直接创建 ECLFieldList 对象，而只能通过创建 ECLPS 对象来间接创建。

ECLFieldList 包含表示空间中所有字段的集合。集合的每个元素都是 ECLField 对象。有关其属性和方法的详细信息，请参阅 ECLField 类 (on page 57)。

ECLFieldList 对象提供调用 Refresh 方法时包含的表示空间的静态快照。如果在调用 Refresh() 后更新表示空间，则字段列表不会反映这些更改。应用程序必须显式调用 Refresh，才能刷新字段列表。

应用程序调用 Refresh 后，即可开始使用 GetFirstField 和 GetNextField 浏览字段集合。如果字段的位置已知，则可以使用 FindField 直接在列表中查找该字段。

注： 调用 Refresh 或破坏 ECLFieldList 对象时，GetFirstField、GetNextField 和 FindField 返回的所有 ECLField 对象指针都将无效。

派生

ECLBase > ECLFieldList

属性

无人

ECLFieldList 方法

下节介绍了对 ECLFieldList 类有效的方法。

```
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);
```

Refresh

此方法会获取表示空间中当前所有字段的快照。以前由此对象返回的所有 ECLField 对象指针都将变为无效。为了提高性能，可以将字段数据限制为感兴趣的平面。请注意，TextPlane 和 FieldPlane 始终可获得。

原型

void Refresh(PS_PLANE Planes=TextPlane)

Parameters

PS_PLANE Planes

为其构建字段的平面。有效值有 **TextPlane**、**ColorPlane**、**FieldPlane**、**ExfieldPlane** 和 **AllPlanes**（为所有项构建）。这是在 ECLPS.HPP 中定义的枚举。此可选参数缺省为 TextPlane。

返回值

无人

示例

以下示例展示了如何使用 Refresh 方法获取表示空间中当前的所有字段的快照。

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

此方法会返回 ECLFieldList 集合中存在的字段数（基于最近对 Refresh 方法的调用）。

原型

ULONG GetFieldCount()

Parameters

无人

返回值

ULONG

ECLFieldList 集合中的字段数。

示例

以下示例展示了如何使用 GetFieldCount 方法返回 ECLFieldList 集合中存在的字段数。

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

此方法会返回指向集合中第一个 ECLField 对象的指针。ECLFieldList 包含 ECLField 对象的集合。请参阅 ECLField 类 (on page 57) 以获取更多信息。如果集合中没有字段，则此方法会返回 NULL 指针。

原型

ECLField * GetFirstField();

Parameters

无人

返回值

ECLField *

指向 ECLField 对象的指针。如果连接中没有字段，则返回 null。

示例

以下示例展示了如何使用 GetFirstField 方法返回指向集合中第一个 ECLField 对象的指针。

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

此方法返回集合中给定对象之后的下一个 ECLField 对象。如果集合中给定对象之后再没有对象，则返回 NULL 指针。应用程序可以重复调用此方法，以迭代集合中的 ECLField 对象。

原型

ECLField *GetNextField(ECLField *Prev)

Parameters

ECLField *Prev

指向集合中任何 ECLField 对象的指针。返回的指针将是此指针之后的下一个对象。如果此值为 NULL，则返回指向集合中第一个对象的指针。此指针是由 GetFirstField、GetNextField 或 FindField 成员函数返回的指针。

返回值

ECLField *

指向集合中下一个对象的指针。如果集合中在 Prev 对象之后再没有对象，则返回 NULL。

示例

以下示例展示了如何使用 GetNextFieldInfo 方法返回指向集合中下一个 ECLField 对象的指针。

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

此方法使用文本或位置查找 ECLFieldList 集合中的字段。位置可以是线性位置，也可以是行、列位置。如果字段包含文本或位置，则返回指向该字段的 ECLField 对象的指针。返回的指针指向字段列表集合中的对象。如果未找到字段，则返回 NULL。搜索文本时，除非指定起始位置，否则搜索从 row1 column1 开始。同样对于文本，此方法将在列表中向前搜索作为缺省值；但是，可以指定显式搜索的方向。

注： 即使文本跨越多个字段，搜索文本也会成功。返回的字段对象将是找到的文本开始的字段。

原型

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);

Parameters

ULONG Pos

要搜索 OR 线性位置以开始文本搜索的线性位置。

ULONG Row

要搜索 OR 行以开始文本搜索的行位置。

ULONG Col

要搜索 OR 列以开始文本搜索的列位置。

char *text

要搜索的字符串

PS_DIR Dir

搜索方向

返回值

ECLField *

如果找到字段，则返回指向 ECLField 对象的指针。如果未找到字段，则返回 NULL。返回的指针在下次调用 Refresh 后无效。

示例

以下是 FindField 方法的示例。

```
//---------------------------------------------------------------------
// 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 类

ECLKeyNotify 是抽象基类。应用程序无法直接创建此类的实例。要使用此类，应用程序必须定义自己派生自 ECLKeyNotify 的类。应用程序必须在其派生类中实现 NotifyEvent() 成员函数。它还可以选择性地实现 NotifyError() 和 NotifyStop() 成员函数。

ECLKeyNotify 类用于允许应用程序收到击键事件通知。应用程序还可以选择过滤（删除）击键，以便不会将其发送到主机屏幕，或将其替换为其他击键。击键通知将排队，以便应用程序将始终收到每次击键的通知。此对象仅检测实际物理键盘的击键；其他 ECL 对象（如 ECLPS::SendKeys）将发送给主机的击键不会导致击键通知事件。

要获得击键事件通知，应用程序必须执行以下步骤：

1. 定义从 ECLKeyNotify 派生的类。
2. 实现派生类并实现 NotifyEvent() 成员函数。
3. （可选）实现 NotifyError() 和/或 NotifyStop() 函数。
4. 创建派生类的实例。
5. 向 ECLPS::RegisterKeyEvent() 函数注册实例。

显示的示例演示了如何执行此操作。完成上述步骤后，仿真器窗口中的每个击键都将导致调用应用程序 NotifyEvent() 成员函数。将向函数传递表示击键类型（纯 ASCII 键或特殊功能键）的参数，以及键的值（单个 ASCII 字符或表示功能键的关键字）。应用程序可以执行 NotifyEvent() 过程中所需的任何函数，包括调用其他 ECL 函数，如 ECLPS::SendKeys()。应用程序从 NotifyEvent() 返回值，以指示是否要过滤击键（返回 1 表示将过滤（放弃）击键，返回 0 表示将对其进行正常处理）。

如果击键在击键事件生成过程中检测到错误，则使用 ECLErr 对象调用 NotifyError() 成员函数。发生错误后，可能会继续生成击键事件，也可能不会继续生成，具体取决于错误的性质。当事件生成终止时（由于错误，通过调用 ECLPS::UnregisterKeyEvent，或由于破坏 ECLPS 对象），将调用 NotifyStop() 成员函数。但是，事件通知会被终止，NotifyStop() 成员函数总是被调用，并且应用程序对象会被注销。

如果应用程序不提供 NotifyError() 成员函数的实现，则使用缺省实现（向用户显示简单的消息框）。应用程序可以通过在应用程序派生类中实现 NotifyError() 函数来覆盖缺省行为。同样，如果应用程序不提供此函数（缺省行为是不执行任何操作），则使用缺省 NotifyStop() 函数。

请注意，应用程序还可以选择为派生类提供自己的构造函数和析构函数。如果应用程序要在类中存储某些特定于实例的数据，并将该信息作为参数传递到构造函数上，则此选项非常有用。例如，当发生击键时，应用程序可能希望将消息发布到应用程序窗口。应用程序不是将窗口句柄定义为全局变量（因此它对 NotifyEvent() 函数可见），而是可以为类定义构造函数，该类接受窗口句柄并将其存储在类成员数据区域中。

应用程序在注册接收事件时不能破坏通知对象。

可以向多个 ECLPS 对象注册击键通知对象的同一实例，以接收多个连接的击键。因此，应用程序可以使用此对象的单个实例来处理任意数量会话上的击键。将向成员函数传递指向发生事件的 ECLPS 对象的指针，以便应用程序可以区分不同连接上的事件。显示的样本使用同一对象处理两个连接上的击键。

实现限制：当前，ECLPS 对象仅允许为给定连接注册一个通知对象。如果已为该 ECLPS 对象注册通知对象，则 ECLPS::RegisterKeyEvent 将抛出错误。

派生

ECLBase > ECLNotify > ECLKeyNotify

示例

以下示例展示了如何构造和使用 ECLKeyNotify 对象。

```
// 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 方法

下节介绍了对 ECLKeyNotify 类有效的方法。

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

此方法是“纯虚拟”成员函数（应用程序*必须*在派生自 ECLKeyNotify 的类中实现此函数）。每当发生击键事件并且为击键事件注册了对象时，都会调用此函数。返回值指示击键的处置（返回 1 表示放弃，0 表示处理）。

原型

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

Parameters

ECLPS *PSObj

这是指向发生事件的 ECLPS 对象的 ptr。

char const KeyType[2]

这是表示键类型的以 null 结束的 1 字符字符串：

A = 纯 ASCII 击键
M = 助记符关键字

const char * const KeyString

这是包含击键或助记符关键字的以 null 结束的字符串。关键字始终为小写（例如“[enter]”）。请参阅 Sendkeys 助记符关键字 (on page 411) 以获取助记符关键字列表。

返回值

int

这是过滤器指示符。

1 = 过滤（放弃）击键
0 = 处理击键（发送到主机）

NotifyError

每当 ECLPS 对象在击键事件生成过程中检测到错误时，都会调用此方法。错误对象包含有关错误的信息（请参阅 ECLErr 类 (on page 54)）。根据错误的性质，出现错误后可能会继续生成击键事件。如果击键事件生成因错误而停止，则将调用 NotifyStop() 函数。

原型

virtual void NotifyError (ELLPS *PSobj, ECLErr ErrObject)

Parameters

ECLPS *PSObj

这是指向发生错误的 ECLPS 对象的 ptr。

ECLErr ErrObject

这是描述错误的 ECLErr 对象。

返回值

无人

NotifyStop

因任何原因（例如，由于错误条件、调用 ECLPS::UnregisterKeyEvent、破坏 ECLPS 对象等）停止击键事件生成时，将调用此方法。

原型

virtual void NotifyStop (ELLPS *PSObj, int Reason)

Parameters

ECLPS *PSObj

这是指向事件正在停止的 ECLPS 对象的 ptr。

int Reason

这未使用（零）。

返回值

无人

ECLListener 类

ECLListener 是所有 HACL“侦听器”对象的基类。侦听器是注册为接收特定类型的异步事件的对象。发生事件或检测到错误时，将调用侦听器对象上的方法。

ECLListener 类上没有公用方法。

派生

ECLBase > ECLListener

使用注意事项

应用程序不直接使用此类，而是创建从其派生的类的实例（例如，ECLPSListener）。

ECLOIA 类

ECLOIA 提供操作员息区域 (OIA) 服务。

由于 ECLOIA 是从 ECLConnection 派生的，因此可以获取 ECLConnection 对象中包含的所有信息。请参阅 ECLConnection 类 (on page 20) 以获取更多信息。

为构造时标识的连接创建 ECLOIA 对象。可以通过传递连接名称（A-Z 或 a-z 中的单个字母字符）或连接句柄（通常从 ECLConnList 对象获得）来创建 ECLOIA 对象。一次只能打开一个具有给定名称或句柄的 Z and I Emulator for Windows 连接。

派生

ECLBase > ECLConnection > ECLOIA

使用注意事项

ECLSession 类会创建此对象的实例。如果应用程序不需要其他服务，则可以直接创建此对象。否则，考虑使用 ECLSession 对象来创建所需的所有对象。

ECLOIA 方法

下节介绍了对 ECLOIA 类有效的方法。

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 构造函数

此方法会根据连接名称（A-Z 或 a-z 中的单个字母字符）或连接句柄创建 ECLOIA 对象。使用给定名称只能启动一条 Z and I Emulator for Windows 连接。

原型

ECLOIA(char ConnName)

ECLOIA(long ConnHandle)

Parameters

char ConnName

连接的单字符短名称（A-Z 或 a-z）。

long ConnHandle

ECL 连接的句柄。

返回值

无人

示例

以下示例展示了如何使用连接名称创建 ECLOIA 对象。

```
// 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

此方法会检查以确定 OIA 是否指示游标位于字母数字位置。

原型

BOOL IsAlphanumeric()

Parameters

无人

返回值

BOOL

如果键盘处于字母数字方式，则返回 TRUE；如果键盘未处于字母数字方式，则返回 FALSE。

示例

以下示例展示了如何确定 OIA 是否指示键盘处于字母数字方式。

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

此方法会检查以确定 OIA 是否指示键盘处于 APL 方式。

原型

BOOL IsAPL()

Parameters

无人

返回值

BOOL

如果键盘处于 APL 方式，则返回 TRUE；如果键盘未处于 APL 方式，则返回 FALSE。

示例

以下示例展示了如何确定 OIA 是否指示键盘处于 APL 方式。

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

此方法会检查以确定 OIA 是否指示键盘处于上 Shift 方式。

原型

BOOL IsUpperShift()

Parameters

无人

返回值

BOOL

如果键盘处于上 Shift 方式，则返回 TRUE；如果键盘未处于上 Shift 方式，则返回 FALSE。

示例

以下示例展示了如何确定 OIA 是否指示键盘处于上 Shift 方式。

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

此方法会检查以确定 OIA 是否指示游标位于仅数字位置。

原型

BOOL IsNumLock()

Parameters

无人

返回值

BOOL

如果 Numeric 为 on，则返回 TRUE；如果非 Numeric，则返回 FALSE。

示例

以下示例展示了如何确定 OIA 是否指示游标位于数字位置。

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

此方法会检查以确定 OIA 是否指示键盘已打开 Caps Lock。

原型

BOOL IsCapsLock()

Parameters

无人

返回值

BOOL

如果 Caps Lock 已打开，则返回 TRUE；如果 Caps Lock 未打开，则返回 FALSE。

示例

以下示例展示了如何确定 OIA 是否指示键盘已打开 Caps Lock。

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

此方法会检查以确定 OIA 是否指示键盘处于插入方式。

原型

BOOL IsInsertMode()

Parameters

无人

返回值

BOOL

如果键盘处于插入方式，则返回 TRUE；如果键盘未处于插入方式，则返回 FALSE。

示例

以下示例展示了如何确定 OIA 是否指示键盘处于插入方式。

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

此方法会检查以确定 OIA 是否指示存在通信错误提醒条件。

原型

BOOL IsCommErrorReminder()

Parameters

无人

返回值

BOOL

如果条件存在，则返回 TRUE；如果条件不存在，则返回 FALSE。

示例

以下示例展示了如何确定 OIA 是否指示存在通信错误提醒条件。

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

此方法会检查以确定 OIA 是否指示消息等待指示符已打开。这仅发生在 5250 会话中。

原型

BOOL IsMessageWaiting()

Parameters

无人

返回值

BOOL

如果消息等待指示符已打开，则返回 TRUE；如果该指示符未打开，则返回 FALSE。

示例

以下示例展示了如何确定 OIA 是否指示消息等待指示符已打开。

```
---------------------------------------------------------------------
// 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 方法会等至与 autECLOIA 对象关联的连接的 OIA 指示连接可以接受键盘输入。

原型

BOOL WaitForInputReady(long nTimeOut = INFINITE)

Parameters

long nTimeOut

要等待的最长时间长度（以毫秒为单位），此参数为可选。缺省值是 INFINITE。

返回值

如果符合条件，方法会返回 TRUE；如果已超过 nTimeOut（以毫秒为单位），则返回 FALSE。

WaitForSystemAvailable

WaitForSystemAvailable 方法会等至与 ECLOIA 对象关联的会话的 OIA 指示会话已连接到主机系统。

原型

BOOL WaitForSystemAvailable(long nTimeOut = INFINITE)

Parameters

long nTimeOut

要等待的最长时间长度（以毫秒为单位），此参数为可选。缺省值是 INFINITE。

返回值

如果符合条件，方法会返回 TRUE；如果已超过 nTimeOut（以毫秒为单位），则返回 FALSE。

WaitForAppAvailable

所连接会话的 OIA 指示应用程序已初始化并可供使用时，WaitForAppAvailable 方法会等待。

原型

BOOL WaitForAppAvailable(long nTimeOut = INFINITE)

Parameters

long nTimeOut

要等待的最长时间长度（以毫秒为单位），此参数为可选。缺省值是 INFINITE。

返回值

如果符合条件，方法会返回 TRUE；如果已超过 nTimeOut（以毫秒为单位），则返回 FALSE。

WaitForTransition

WaitForTransition 方法会等待所连接会话的 OIA 中指定位置的值发生更改。

原型

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

Parameters

BYTE nIndex

要监控的 OIA 的 1 字节十六进制位置。此参数是可选参数。缺省值为 3。

long nTimeOut

要等待的最长时间长度（以毫秒为单位），此参数为可选。缺省值是 INFINITE。

返回值

如果符合条件，方法会返回 TRUE；如果已超过 nTimeOut（以毫秒为单位），则返回 FALSE。

InputInhibited

此方法会返回枚举值，指示输入是否被禁止。如果输入被禁止，则可以确定禁止的原因。如果输入因多个原因而被禁止，则返回最高值枚举（例如，如果存在通信错误和协议编程错误，则返回 ProgCheck 值）。

原型

INHIBIT_REASON InputInhibited ()

Parameters

无人

返回值

INHIBIT_REASON

返回 ECLOIA.HPP 中定义的其中一个 INHIBIT_REASON 值。如果输入当前未被禁止，则返回值 NotInhibited。

示例

以下示例展示如何确定输入是否被禁止。

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

此方法会返回一组表示各种 OIA 指示符的状态位。此方法可用于在单个调用中收集一组 OIA 指示符，而不是调用多个不同的 IsXXX 方法。返回的每个位都表示单个 OIA 指示符，其中值 1 表示该指示符已打开 (TRUE)，0 表示它已关闭 (FALSE)。ECLOIA.HPP 头文件中定义了一组位掩码常量，用于隔离返回的 32 位值中的各个指示符。

原型

ULONG GetStatusFlags()

Parameters

无人

返回值

ULONG

按如下方式定义的位标志集：

位位置	掩码常量	描述
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		<保留>
3-0	OIAFLAG_INHIBMASK	InputInhibited： 0=NotInhibited 1=SystemWait 2=CommCheck 3=ProgCheck 4=MachCheck 5=OtherInhibit

RegisterOIAEvent

此成员函数会注册应用程序对象，以接收 OIA 更新事件的通知。要使用此函数，应用程序必须创建从 ECLOIANotify 派生的对象。接着将指向该对象的指针传递到此注册函数。可以同时注册任意数量的通知对象。未定义多个侦听器接收事件的顺序，因此不应假定该顺序。

使用此函数注册 ECLOIANotify 对象后，每当发生 OIA 更新时，都会调用其 NotifyEvent() 方法。在短时间内对 OIA 的多次更新可聚合为一个事件。

应用程序必须先注销通知对象，然后才能将其破坏。如果 ECLOIA 对象被破坏，将自动注销该对象。

原型

void RegisterOIAEvent(ECLOIANotify * notify)

Parameters

ECLOIANotify *

指向要注册的 ECLOIANotify 对象的指针。

返回值

无人

UnregisterOIAEvent

此成员函数会注销以前在 RegisterOIAEvent 函数中注册的应用程序对象。如果未先调用此函数将其注销，则不应破坏注册为接收事件的对象。如果当前未注册特定对象，则不会执行任何操作，也不会出现错误。

注销 ECLOIANotify 对象时，将调用其 NotifyStop() 方法。

原型

void UnregisterOIAEvent(ECLOIANotify * notify)

Parameters

ECLPSNotify *

指向要注销的 ECLOIANotify 对象的指针。

返回值

无人

ECLOIANotify 类

ECLOIANotify 是抽象基类。应用程序无法直接创建此类的实例。要使用此类，应用程序必须定义自己派生自 ECLOIANotify 的类。应用程序必须在其派生类中实现 NotifyEvent() 成员函数。它还可以选择性地实现 NotifyError() 和 NotifyStop() 成员函数。

ECLOIANotify 类用于允许应用程序收到操作员信息区域更新的通知。每当更新 OIA 上的任何指示符时，都会生成事件。

派生

ECLBase > ECLNotify > ECLOIANotify

使用注意事项

要使用此类获得 OIA 更新通知，应用程序必须执行以下步骤：

1. 定义从 ECLOIANotify 派生的类。
2. 实现 ECLOIANotify 派生类的 NotifyEvent 方法。
3. （可选）实现 ECLOIANotify 的其他成员函数。
4. 创建派生类的实例。
5. 使用 ECLOIA::RegisterOIAEvent() 方法注册实例。

注册完成后，对 OIA 指示符的更新将导致调用 ECLOIANotify 派生类的 NotifyEvent() 方法。

请注意，短时间内发生的多次 OIA 更新可能会聚合到单个事件通知中。

应用程序可以选择为派生类提供自己的构造函数和析构函数。如果应用程序需要在类中存储某些特定于实例的数据，并将该信息作为参数传递到构造函数上，则此选项非常有用。

如果在事件注册过程中检测到错误，则使用 ECLErr 对象调用 NotifyError() 成员函数。发生错误后，可能会继续生成事件，也可能不会继续生成。当事件生成终止（由于错误或其他原因）时，调用 NotifyStop() 成员函数。NotifyError() 的缺省实现将向用户显示一个消息框，其中显示从 ECLErr 对象检索到的错误消息文本。

当事件通知出于任何原因（错误或调用 ECLOIA::UnregisterOIAEvent）停止时，将调用 NotifyStop() 成员函数。NotifyStop() 的缺省实现不执行任何操作。

ECLOIANotify 方法

下节介绍了对 ECLOIANotify 类及其派生的所有类有效的方法。

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

NotifyEvent

此方法是纯虚拟成员函数（应用程序**必须**在派生自 ECLOIANotify 的类中实现此函数）。只要更新 OIA 并且注册此对象以接收更新事件，就会调用此方法。

多个 OIA 更新可能会聚集到单个事件中，从而导致仅调用此方法一次。

原型

virtual void NotifyEvent(ECLOIA * OIAObj) = 0

Parameters

ECLOIA *

指向生成此事件的 ECLOIA 对象的指针。

返回值

无人

NotifyError

每当 ECLOIA 对象在事件生成过程中检测到错误时，都会调用此方法。错误对象包含有关错误的信息（请参阅 ECLErr 类描述）。根据错误的性质，出现错误后可能会继续生成事件。如果事件生成因错误而停止，则将调用 NotifyStop() 方法。

应用程序可以选择实现此功能或允许基本 ECLOIANotify 类处理此功能。缺省实现将使用 ECLErr::GetMsgText() 方法提供的文本在消息框中显示错误。如果应用程序在其派生类中实现此函数，则会覆盖此行为。

原型

virtual void NotifyError(ECLOIA * OIAObj, ECLErr ErrObj)

Parameters

ECLOIA *

指向生成此事件的 ECLOIA 对象的指针。

ECLErr

描述错误的 ECLErr 对象。

返回值

无人

NotifyStop

因任何原因（例如，由于错误条件或调用 ECLOIA::UnregisterOIAEvent）停止事件生成时，将调用此方法。

原因码参数当前未在使用，将为零。

此函数的缺省实现不会执行任何操作。

原型

virtual void NotifyStop(ECLOIA * OIAObj, int Reason)

Parameters

ECLOIA *

指向生成此事件的 ECLOIA 对象的指针。

int

事件生成已停止的原因（当前未使用，因此将为零）。

返回值

无人

ECLPS 类

ECLPS 类对主机表示空间执行操作。

为构造时标识的连接创建 ECLPS 对象。可以通过传递连接名称（A-Z 中的单个字母字符）或连接句柄（通常从 ECLConnection 对象获得）来创建 ECLPS 对象。一次只能打开一个具有给定名称或句柄的 Z and I Emulator for Windows 连接。

派生

ECLBase > ECLConnection > ECLPS

属性

无人

使用注意事项

ECLSession 类会创建此对象的实例。如果应用程序不需要其他服务，则可以直接创建此对象。否则，您可能需要考虑使用 ECLSession 对象来创建所需的所有对象。

ECLPS 方法

下节介绍了可用于 ECLPS 的方法。

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)

以下方法可用于 ECLPS：

```
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 构造函数

此方法会使用连接名称或句柄来创建 ECLPS 对象。

原型

ECLPS(char ConnName)

ECLPS(long ConnHandle)

Parameters

char ConnName

连接的单字符短名称（A-Z 或 a-z）。

long ConnHandle

ECL 连接的句柄。

返回值

无人

示例

以下示例展示了如何使用连接名称创建 ECLPS 对象。

```
//---------------------------------------------------------------------
// 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 析构函数

此方法会破坏 ECLPS 对象。

原型

~ECLPS()

Parameters

无人

返回值

无人

示例

以下示例展示了如何破坏 ECLPS 对象。

```
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 方法会检索指定个人计算机有效代码页的编号。

原型

int GetPCCodePage()

Parameters

无人

返回值

int

代码页的编号。

GetHostCodePage

GetHostCodePage 方法会检索指定主机计算机有效代码页的编号。

原型

int GetHostCodePage()

Parameters

无人

返回值

int

代码页的编号。

GetOSCodePage

GetOSCodePage 方法会在个人计算机上检索指定操作系统有效代码页的编号。

原型

int GetOSCodePage()

Parameters

无人

返回值

int

代码页的编号。

GetSize

此方法会返回与 ECLPS 对象关联的连接的表示空间的大小。GetSize 方法有两个签名。使用 ULONG GetSize()，大小作为线性值返回，表示表示表示空间中的字符总数。使用 void GetSize(ULONG *Rows, ULONG *Cols)，将返回表示空间的行数和列数。

原型

ULONG GetSize()

void GetSize(ULONG *Rows, ULONG *Cols)

Parameters

ULONG *Rows

此输出参数是表示空间中的行数。

ULONG *Cols

此输出参数是表示空间中的列数。

返回值

ULONG

表示为线性值的表示空间大小。

示例

下面是使用 GetSize 方法的示例。

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

此方法会返回表示空间中的行数，用于与 ECLPS 对象关联的连接。

原型

ULONG GetSizeRows()

Parameters

无人

返回值

ULONG

这是表示空间中的行数。

示例

下面是使用 GetSizeRows 方法的示例。

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

此方法会返回表示空间中的列数，用于与 ECLPS 对象关联的连接。

原型

ULONG GetSizeCols()

Parameters

无人

返回值

ULONG

这是表示空间中的列数。

示例

以下是使用 GetSizeCols 方法的示例。

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

此方法会返回游标在与 ECLPS 对象关联的连接的表示空间中的位置。GetCursorPos 方法有两个签名。使用 ULONG GetCursorPos()，该位置会作为线性（基于 1）位置返回。使用 void GetCursorPos(ULONG *Row, ULONG * Col)，该位置会作为行和列坐标返回。

原型

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

Parameters

ULONG *Row

此输出参数是主机游标的行坐标。

ULONG *Col

此输出参数是主机游标的列坐标。

返回值

ULONG

表示为线性值的游标位置。

示例

以下是使用 GetCursorPos 方法的示例。

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

此方法会返回游标在表示空间中的行位置，用于与 ECLPS 对象关联的连接。

原型

ULONG GetCursorPosRow()

Parameters

无人

返回值

ULONG

这是游标在表示空间中的行位置。

示例

以下是使用 GetCursorPosRow 方法的示例。

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

此方法会返回游标在表示空间中的列位置，用于与 ECLPS 对象关联的连接。

原型

ULONG GetCursorPosCol()

Parameters

无人

返回值

ULONG

这是游标在表示空间中的列位置。

示例

以下是使用 GetCursorPosCol 方法的示例。

```
//-------------------------------------------------------------------
// 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 会设置游标在与 ECLPS 对象关联的连接的表表示空间中的位置。SetCursorPos 方法有两个签名。可以使用 void SetCursorPos(ULONG pos) 将位置指定为线性（基于 1）位置，也可以使用 void SetCursorPos(ULONG Row, ULONG Col) 将位置指定为行和列坐标。

原型

void SetCursorPos(ULONG pos),

void SetCursorPos(ULONG Row, ULONG Col)

Parameters

ULONG pos

表示为线性位置的游标位置。

ULONG Row

游标行坐标。

ULONG Col

游标列坐标。

返回值

无人

示例

以下是使用 SetCursorPos 方法的示例。

```
--
// 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 方法会将以 null 结束的键字符串发送到表示空间，用于与 ECLPS 对象关联的连接。SendKeys 方法有三个签名。如果未指定位置，则从当前主机光标位置开始输入击键。可以指定位置（线性位置或行和列坐标），在这种情况下，主机光标首先移到给定位置。

文本字符串可能包含纯文本字符，这些字符将完全按照给定的方式写入表示空间。此外，该字符串可以包含嵌入式关键字（助记符），这些关键字表示各种控制键击，如 3270 Enter 键和 5250 PageUp 键。关键字括在方括号中（例如 [enter]）。在字符串中遇到此类关键字时，会将其转换为正确的仿真器命令并发送。文本字符串可以包含任意数量的纯字符和嵌入式关键字。从左到右处理关键字，直至到达字符串的末尾。例如，以下字符串将导致在当前光标位置输入字符 ABC，然后输入 3270 擦除字段结束击键，接着输入 3270 Tab 击键，最后输入 XYZ 和 PF1 键：

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

注： 字符串中的空白字符将像任何其他纯文本字符一样写入主机表示空间。因此，不应使用空格来分隔关键字或文本。

要将左方括号或右方括号字符发送给主机，必须在文本字符串中将其加倍（例如，它必须出现两次才能写入单个方括号）。以下示例将字符串 “A [:]” 写入表示空间。

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

如果尝试将击键写入屏幕上受保护的位置，键盘锁定和剩余的击键将被丢弃。

有关关键字列表，请参阅 Sendkeys 助记符关键字 (on page 411)。

原型

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

Parameters

Char *text

要发送到表示空间的键字符串。

ULONG AtPos

开始写入击键的位置。

ULONG AtRow

开始写入击键的行。

ULONG AtCol

开始写入击键的列。

返回值

无人

示例

以下是使用 SendKeys 方法的示例。

```
//-------------------------------------------------------------------
// 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 方法会在与 ECLPS 对象关联的连接的表示空间中搜索文本。此方法会返回找到文本的线性位置；如果未找到文本，则返回零。使用可选的 Dir 参数，可以向前（从左到右，从上到下）或向后（从右到左，从下到上）进行搜索。使用可选的 FoldCase 参数，搜索可以区分大小写，也可以折叠大小写（不区分）。

如果未提供起始位置，则搜索从向前搜索的屏幕开头开始，或从向后搜索的屏幕结尾开始。起始位置可以使用线性位置或行和列坐标提供。如果提供了起始位置，则表示开始搜索的位置。向前搜索从起始位置（包括）搜索到屏幕的最后一个字符。向后搜索从起始位置（包括）搜索到屏幕的第一个字符。

搜索字符串必须完全存在于搜索区域内，搜索才能成功（例如，如果搜索字符串横跨指定的起始位置，则不会找到它）。

返回的线性位置可以使用基类 ConvertPosToRowCol 方法转换为行和列坐标。

原型

```
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)
```

Parameters

char *text

要搜索的以 null 结束的字符串。

PS_DIR Dir

指示搜索方向的可选参数。如果指定，则必须是 **SrchForward** 或 **SrchBackward** 中的一个。缺省值为 **SrchForward**。

BOOL FoldCase

指示搜索区分大小写的可选参数。如果指定为 FALSE，则文本字符串必须与表示空间完全匹配，包括大写和小写字符的使用。如果指定为 TRUE，则查找文本字符串时不考虑大小写。缺省值为 FALSE。

ULONG StartPos

指示搜索的起始线性位置。此位置将包括在搜索中。

ULONG StartRow

指示要开始搜索的行。

ULONG StartCol

指示要开始搜索的列。

返回值

ULONG

找到的字符串的线性位置，未找到时则为零。

示例

以下是使用 SearchText 方法的示例。

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

此方法会从与 ECLPS 对象关联的连接的表示空间中检索数据。数据以字节值的线性数组的形式返回，每个表示空间字符位置一个字节。除非从 TextPlane 检索数据，否则数组不以 null 结束，在这种情况下，将追加一个 null 终止字节。

应用程序必须为返回的数据提供缓冲区，并提供缓冲区的长度。如果请求的数据不适合缓冲区，则会被截断。对于 TextPlane 数据，缓冲区必须包含至少一个以 null 结束的额外字节。该方法会返回复制到应用程序缓冲区的字节数（不包括 TextPlane 副本的终止 null 值）。

应用程序必须指定要从表示空间检索的数据字节数。如果起始位置加上此长度超过表示空间的大小，则会抛出错误。如果未指定起始位置，则从给定起始位置或第 1 行、第 1 列开始返回数据。返回的数据以线性方式从表示空间从左到右、从上到下复制，横跨多行，直至指定的长度。如果应用程序要获取屏幕矩形区域的屏幕数据，则应使用 GetScreenRect 方法。

应用程序可以指定要检索数据的任何平面。如果未指定平面，将检索 TextPlane。有关不同 ECL 平面的详细信息，请参阅 ECL 平面 - 格式和内容 (on page 414)。

原型

```
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)
```

Parameters

char *Buff

指向至少为 BuffLen 大小的应用程序提供的缓冲区的指针。

ULONG BuffLen

提供的缓冲区中的字节数。

ULONG StartPos

在表示空间中要开始复制的线性位置。

ULONG StartRow

在表示空间中要开始复制的行。

ULONG StartCol

在表示空间中要开始复制的列。

ULONG Length

要从表示空间复制的线性字节数。

PS_PLANE plane

可选参数，指定要复制的表示空间平面。如果指定，必须是 **TextPlane**、**ColorPlane**、**FieldPlane** 和 **ExfieldPlane** 中的一个。缺省值为 **TextPlane**。有关不同 ECL 平面的内容和格式，请参阅 ECL 平面 - 格式和内容 (on page 414)。

返回值

ULONG

从表示空间复制的数据字节数。此值不包括 TextPlane 副本的尾随 null 字节。

示例

以下是使用 GetScreen 方法的示例。

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

此方法会从与 ECLPS 对象关联的连接的表示空间中检索数据。数据以字节值的线性数组的形式返回，每个表示空间字符位置一个字节。数组不是以 null 结束。

应用程序提供表示空间中的起始和结束坐标。这些坐标构成矩形区域的相对角点。矩形区域内的表示空间仅作为单个线性数组复制到应用程序缓冲区。起点和终点可以是彼此之间的任何空间关系。复制被定义为从包含最上点的行开始到包含最下点的行，以及从最左边的列开始到最右边的列。两个坐标都必须在表示空间大小的边界内，否则会抛出错误。可以根据线性位置或行号和列号来指定坐标。

提供的应用程序缓冲区必须至少大到足以包含矩形中的字节数。如果缓冲区太小，则不会复制任何数据，并返回零作为方法结果。否则，该方法将返回复制的字节数。

应用程序可以指定要检索数据的任何平面。如果未指定平面，将检索 TextPlane。有关不同 ECL 平面的详细信息，请参阅 ECL 平面 - 格式和内容 (on page 414)。

原型

```
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)
```

Parameters

char *Buff

指向至少为 BuffLen 大小的应用程序提供的缓冲区的指针。

ULONG BuffLen

提供的缓冲区中的字节数。

ULONG StartPos

复制矩形一角的表示空间中的线性位置。

ULONG EndPos

复制矩形一角的表示空间中的线性位置。

ULONG StartRow

复制矩形一角的表示空间中的行。

ULONG StartCol

复制矩形一角的表示空间中的列。

ULONG EndRow

复制矩形一角的表示空间中的行。

ULONG EndCol

复制矩形一角的表示空间中的列。

PS_PLANE plane

可选参数，指定要复制的表示空间平面。如果指定，必须是 **TextPlane**、**ColorPlane**、**FieldPlane** 或 **ExfieldPlane** 中的一个。缺省值为 **TextPlane**。有关不同 ECL 平面的内容和格式，请参阅 ECL 平面 - 格式和内容 (on page 414)。

返回值

ULONG

从表示空间复制的数据字节数。

示例

以下是使用 GetScreenRect 方法的示例。

```
------------------------------
// 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 方法为与 ECLPS 对象相关联的连接发送一个字符数组至“表示空间”。尽管这与 SendKeys 方法很相似，但不同的是 SetText 不发送助记符击键（例如 [enter] 或 [pf1]）。

如果未指定位置，则从当前游标位置开始写入文本。

原型

void SetText(char *text);

void SetText(char *text, ULONG AtPos);

void SetText(char *text, ULONG AtRow, ULONG AtCol);

Parameters

char *text

要复制到表示空间的字符的以 null 结束的字符串。

ULONG AtPos

在表示空间中要开始复制的线性位置。

ULONG AtRow

在表示空间中要开始复制的行。

ULONG AtCol

在表示空间中要开始复制的列。

返回值

无人

示例

以下是使用 SetText 方法的示例。

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

此方法会将表示空间中从给定位置开始的指定长度文本复制到剪贴板。所复制文本的长度将是指定的长度，如果未指定长度，则复制直到表示空间末尾的文本。如果未指定位置，所复制文本将从表示空间中当前光标位置开始。如果未指定任何参数，会将整个表示空间复制到剪贴板。

原型

void CopyText ();

void CopyText (ULONG Long Len);

void CopyText (ULONG AtPos, ULONG Long Len);

void CopyText (ULONG AtRow, ULONG AtCol, ULONG Long Len);

Parameters

ULONG Long Len

要从表示空间复制的线性字节数。

ULONG AtPos

在表示空间中要开始复制的线性位置。

ULONG AtRow

在表示空间中要开始复制的行。

ULONG AtCol

在表示空间中要开始复制的列。

返回值

无人

示例

以下是使用 CopyText 方法的示例。

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

粘贴文本

此方法将指定长度的文本从剪贴板粘贴到演示空间中的给定位置。粘贴文本的长度是指定的长度，如果未指定长度，则粘贴剪贴板中的整个文本，直到到达演示空间的末尾。如果未指定位置，则文本将粘贴到演示空间中的当前光标位置。如果演示空间是字段格式的，并且在粘贴剪贴板内容时，当存在制表符“\t”时，剩余的粘贴内容将移动到下一个可写字段。

原型

void PasteText ();

void PasteText (ULONG Long Len);

void PasteText (ULONG AtPos, ULONG Long Len);

void PasteText (ULONG AtRow, ULONG AtCol, ULONG Long Len);

Parameters

ULONG Long Len

要从表示空间粘贴的线性字节数。

ULONG AtPos

在表示空间中要开始粘贴的线性位置。

ULONG AtRow

在表示空间中要开始粘贴的行。

ULONG AtCol

在表示空间中要开始粘贴的列。

返回值

无人

示例

以下是使用 PasteText 方法的示例。

```
//-------------------------------------------------------------------
// 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 方法会将表示为线性数组的表示空间中的位置转换为表示空间中以行和列坐标给出的位置。转换后的位置位于与 ECLPS 对象关联的连接的表示空间中。

原型

void ConvertPosToRowCol(ULONG pos, ULONG *row, ULONG *col)

Parameters

ULONG pos

要在表示为线性数组的表示空间中转换的位置。

ULONG *row

表示空间中已转换的行坐标。

ULONG *col

表示空间中已转换的列坐标。

返回值

无人

示例

以下示例展示了如何将表示为线性数组的表示空间中的位置转换为以行和列坐标显示的位置。

```
///--------------------------------------------------------------------
// 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 方法会将表示空间中以行和列坐标表示的位置转换为表示为线性数组的表示空间中的位置。转换后的位置位于与 ECLPS 对象关联的连接的表示空间中。

原型

ULONG ConvertRowColToPos(ULONG row, ULONG col)

Parameters

ULONG row

表示空间中要转换的行坐标。

ULONG col

表示空间中要转换的列坐标。

返回值

ULONG

表示为线性数组的表示空间中的转换后位置。

示例

以下示例展示了如何将以行和列坐标显示的表示空间中的位置转换为线性数组位置。

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

此方法会在表示空间中获取线性位置值，并返回与 ECLPS 对象关联的连接所在的行。

原型

ULONG ConvertPosToRow(ULONG Pos)

Parameters

ULONG Pos

这是表示空间中要转换的线性位置。

返回值

ULONG

这是线性位置的行位置。

示例

以下是使用 ConvertPosToRow 方法的示例。

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

此方法会在表示空间中获取线性位置值，并返回与 ECLPS 对象关联的连接所在的列。

原型

ULONG ConvertPosToCol(ULONG Pos)

Parameters

ULONG Pos

这是表示空间中要转换的线性位置。

返回值

ULONG

这是线性位置的列位置。

示例

以下是使用 ConvertPosToCol 方法的示例。

```
///----------------------------------------------------------------------
/// 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 函数会注册应用程序提供的对象以接收操作员击键事件的通知。应用程序必须构造从 ECLKeyNotify 抽象基类派生的对象。当出现操作员击键时，将调用应用程序提供的对象的 NotifyEvent() 方法。应用程序可以选择过滤或传递并以常规方式处理的击键。请参阅 ECLKeyNotify 类 (on page 81) 以获取更多详细信息。

实现限制：一次只能注册一个对象以接收击键事件。

原型

void RegisterKeyEvent(ECLKeyNotify *NotifyObject)

Parameters

ECLKeyNotify *NotifyObject

从 ECLKeyNotify 类派生的应用程序对象。

返回值

无人

示例

以下示例展示了如何注册应用程序提供的对象以接收操作员击键事件的通知。请参阅 ECLKeyNotify 类 (on page 81) 以获取 RegisterKeyEvent 示例。

```
// 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 方法使用 RegisterKeyEvent 函数注销先前为击键事件注册的应用程序对象。如果未首先调用此函数来将其注销，则不能破坏已注册的应用程序通知对象。如果当前未注册通知对象，或者注册的对象不是传入的 NotifyObject，则此函数不执行任何操作（不抛出错误）。

原型

virtual UnregisterKeyEvent(ECLKeyNotify *NotifyObject)

Parameters

ECLKeyNotify *NotifyObject

当前为击键事件注册的对象。

返回值

无人

示例

请参阅 ECLKeyNotify 类 (on page 81) 以获取 UnregisterKeyEvent 示例。

GetFieldList

此方法会返回指向 ECLFieldList 对象的指针。字段列表对象可用于迭代主机表示空间中的字段列表。破坏 ECLPS 对象时，将自动破坏此函数返回的 ECLFieldList 对象。请参阅 ECLFieldList 类 (on page 74) 以获取有关此对象的更多信息。

原型

ECLFieldList *GetFieldList()

Parameters

无人

返回值

ECLFieldList *

指向 ECLFieldList 对象的指针。

示例

以下示例展示了如何返回指向 ECLFieldList 对象的指针。

```
// 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 方法会等待与 ECLPS 对象关联的连接的表示空间中的游标位于指定位置。

原型

BOOL WaitForCursor(int Row, int Col, long nTimeOut=INFINITE,
    BOOL bWaitForIR=TRUE)

Parameters

int Row

光标所在的行位置。如果为负值，则此值表示从 PS 底部开始的行位置。

int Col

光标所在的列位置。如果为负值，则此值指示从 PS 边缘开始的游标位置。

long nTimeOut

以毫秒为单位的最长等待时间。此参数是可选参数。缺省值是 INFINITE。

BOOL bWaitForIR

如果此值为 True，在符合等待条件之后，函数将等至 OIA 指示 PS 已准备好接受输入。此参数是可选参数，缺省为 TRUE。

返回值

如果符合条件，方法会返回 TRUE；如果已超过 nTimeOut（以毫秒为单位），则返回 FALSE。

注： 如果 nTimeOut 是缺省值 (INFINITE)，当测试条件返回 FALSE 时，此方法将阻止。

示例

```
// 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

当与 ECLPS 对象关联的连接的表示空间中的游标位于指定位置时，WaitWhileCursor 方法会等待。

原型

BOOL WaitWhileCursor(int Row, int Col, long nTimeOut=INFINITE,
BOOL bWaitForIR=TRUE)

Parameters

int Row

光标所在的行位置。如果为负值，则此值表示从 PS 底部开始的行位置。

int Col

光标所在的列位置。如果为负值，则此值指示从 PS 边缘开始的游标位置。

long nTimeOut

以毫秒为单位的最长等待时间。此参数是可选参数。缺省值是 INFINITE。

BOOL bWaitForIR

如果此值为 True，在符合等待条件之后，函数将等至 OIA 指示 PS 已准备好接受输入。此参数是可选参数，缺省为 TRUE。

返回值

如果符合条件，方法会返回 TRUE；如果已超过 nTimeOut（以毫秒为单位），则返回 FALSE。

注： 如果 nTimeOut 是缺省值 (INFINITE)，当测试条件返回 FALSE 时，此方法将阻止。

示例

```
// 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 方法会等待指定字符串出现在与 ECLPS 对象关联的连接的表示空间中。如果使用了可选的 Row 和 Column 参数，则字符串必须从指定位置开始。如果对 Row 和 Col 指定 0,0，则方法将搜索整个 PS。

原型

```
BOOL WaitForString( char* WaitString, int Row=0, int Col=0, long nTimeOut=INFINITE,
    BOOL bWaitForIR=TRUE, BOOL bCaseSens=TRUE)
```

Parameters

char* WaitString

将作为等待主体的字符串。

int Row

光标所在的行位置。如果为负值，则此值表示从 PS 底部开始的行位置。缺省值是零。

int Col

光标所在的列位置。如果为负值，则此值指示从 PS 边缘开始的游标位置。缺省值是零。

long nTimeOut

以毫秒为单位的最长等待时间。此参数是可选参数。缺省值是 INFINITE。

BOOL bWaitForIR

如果此值为 True，在符合等待条件之后，函数将等至 OIA 指示 PS 已准备好接受输入。此参数是可选参数，缺省为 TRUE。

BOOL bCaseSens

如果此值为 True，则等待条件被验证为区分大小写。此参数是可选参数。缺省值为 TRUE。

返回值

如果符合条件，方法会返回 TRUE；如果已超过 nTimeOut（以毫秒为单位），则返回 FALSE。

注： 如果 nTimeOut 是缺省值 (INFINITE)，当测试条件返回 FALSE 时，此方法将阻止。

示例

```
// set up PS
ECLPS ps = new ECLPS('A');

// do the wait
BOOL waitOK = ps.WaitForString("LOGON");

// do the processing for the screen
```

WaitWhileString

当指定字符串位于与 ECLPS 对象关联的连接的表示空间中时，WaitWhileString 方法会等待。如果使用了可选的 Row 和 Column 参数，则字符串必须从指定位置开始。如果对 Row 和 Col 指定 0,0，则方法将搜索整个 PS。

原型

```
BOOL WaitWhileString(char* WaitString, int Row=0, int Col=0,
                     long nTimeOut=INFINITE,
                     BOOL bWaitForIR=TRUE, BOOL bCaseSens=TRUE)
```

Parameters

char* WaitString

将作为等待主体的字符串。

int Row

字符串的起始行位置。如果为负值，则此值表示从 PS 底部开始的行位置。缺省值是零。

int Col

字符串的起始列位置。如果为负值，则此值指示从 PS 边缘开始的游标位置。缺省值是零。

long nTimeOut

以毫秒为单位的最长等待时间。此参数是可选参数。缺省值是 INFINITE。

BOOL bWaitForIR

如果此值为 True，在符合等待条件之后，函数将等至 OIA 指示 PS 已准备好接受输入。此参数是可选参数，缺省为 TRUE。

BOOL bCaseSens

如果此值为 True，则等待条件被验证为区分大小写。此参数是可选参数。缺省值为 TRUE。

返回值

如果符合条件，方法会返回 TRUE；如果已超过 nTimeOut（以毫秒为单位），则返回 FALSE。

注： 如果 nTimeOut 是缺省值 (INFINITE)，当测试条件返回 FALSE 时，此方法将阻止。

示例

```
// 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 方法会等待指定字符串出现在与指定矩形中的 ECLPS 对象关联的连接的表示空间中。

原型

```
BOOL WaitForStringInRect(char* WaitString, int sRow, int sCol, int eRow,int eCol,
    long nTimeOut=INFINITE, BOOL bWaitForIR=TRUE, BOOL bCaseSens=TRUE)
```

Parameters

char* WaitString

将作为等待主体的字符串。

int Row

矩形的起始行位置。

int Col

矩形的起始列位置。

int eRow

搜索矩形的结束行位置。

int eCol

搜索矩形的结束列位置。

long nTimeOut

以毫秒为单位的最长等待时间。此参数是可选参数。缺省值是 INFINITE。

BOOL bWaitForIR

如果此值为 True，在符合等待条件之后，函数将等至 OIA 指示 PS 已准备好接受输入。此参数是可选参数，缺省为 TRUE。

BOOL bCaseSens

如果此值为 True，则等待条件被验证为区分大小写。此参数是可选参数。缺省值为 TRUE。

返回值

如果符合条件，方法会返回 TRUE；如果已超过 nTimeOut（以毫秒为单位），则返回 FALSE。

注： 如果 nTimeOut 是缺省值 (INFINITE)，当测试条件返回 FALSE 时，此方法将阻止。

示例

```
// 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

当指定字符串位于与指定矩形中的 ECLPS 对象关联的连接的表示空间时，WaitWhileStringInRect 方法会等待。

原型

```
BOOL WaitWhileStringInRect(char* WaitString, int sRow, int sCol, int eRow,int eCol,
    long nTimeOut=INFINITE, BOOL bWaitForIR=TRUE, BOOL bCaseSens=TRUE)
```

Parameters

char* WaitString

将作为等待主体的字符串。

int Row

矩形的起始行位置。

int Col

矩形的起始列位置。

int eRow

搜索矩形的结束行位置。

int eCol

搜索矩形的结束列位置。

long nTimeOut

以毫秒为单位的最长等待时间。此参数是可选参数。缺省值是 INFINITE。

BOOL bWaitForIR

如果此值为 True，在符合等待条件之后，函数将等至 OIA 指示 PS 已准备好接受输入。此参数是可选参数，缺省为 TRUE。

BOOL bCaseSens

如果此值为 True，则等待条件被验证为区分大小写。此参数是可选参数。缺省值为 TRUE。

返回值

如果符合条件，方法会返回 TRUE；如果已超过 nTimeOut（以毫秒为单位），则返回 FALSE。

注： 如果 nTimeOut 是缺省值 (INFINITE)，当测试条件返回 FALSE 时，此方法将阻止。

示例

```
// 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 方法将等待，直到指定的属性值出现在与指定行/列位置的 ECLPS 对象相关联连接的表示空间中。可选的 MaskData 参数可用于控制您正在查找哪个属性值。可选的平面参数允许您选择四个 PS 平面中的任一个。

原型

```
BOOL WaitForAttrib(int Row, int Col, unsigned char AttribDatum,
    unsigned char MskDatum= 0xFF, PS_PLANE plane = FieldPlane,
    long TimeOut = INFINITE, BOOL bWaitForIR = TRUE)
```

Parameters

int Row

属性的行位置。

int Col

属性的列位置。

unsigned char AttribDatum

要等待的属性的 1 字节十六进制值。

unsigned char MskDatum

作为属性掩码使用的 1 字节十六进制值。此参数是可选参数。缺省值为 0xFF。

PS_PLANE plane

要获取的属性的平面。该平面可有下列值：**TextPlane**、**ColorPlane**、**FieldPlane** 和 **ExfieldPlane**。有关不同 ECL 平面的内容和格式，请参阅 ECL 平面 - 格式和内容 (on page 414)。

此参数是可选参数。缺省值为 FieldPlane。

long nTimeOut

以毫秒为单位的最长等待时间。此参数是可选参数。缺省值是 INFINITE。

BOOL bWaitForIR

如果此值为 True，在符合等待条件之后，函数将等至 OIA 指示 PS 已准备好接受输入。此参数是可选参数，缺省为 TRUE。

返回值

如果符合条件，方法会返回 TRUE；如果已超过 nTimeOut（以毫秒为单位），则返回 FALSE。

注： 如果 nTimeOut 是缺省值 (INFINITE)，当测试条件返回 FALSE 时，此方法将阻止。

示例

```
// 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

当指定的属性值出现在与指定行/列位置的 ECLPS 对象相关联连接的表示空间中时，WaitWhileAttrib 方法将等待。可选的 MaskData 参数可用于控制您正在查找哪个属性值。可选的平面参数允许您选择四个 PS 平面中的任一个。

原型

```
BOOL WaitWhileAttrib(int Row, int Col, unsigned char AttribDatum,
    unsigned char MskDatum= 0xFF, PS_PLANE plane = FieldPlane,
    long TimeOut = INFINITE, BOOL bWaitForIR = TRUE)
```

Parameters

int Row

属性的行位置。

int Col

属性 unsigned 的列位置。

char AttribDatum

要等待的属性的 1 字节十六进制值。

unsigned char MskDatum

作为属性掩码使用的 1 字节十六进制值。此参数是可选参数。缺省值为 0xFF。

PS_PLANE plane

要获取的属性的平面。该平面可有下列值：**TextPlane**、**ColorPlane**、**FieldPlane** 和 **ExfieldPlane**。有关不同 ECL 平面的内容和格式，请参阅 ECL 平面 - 格式和内容 (on page 414)。

此参数是可选参数。缺省值为 FieldPlane。

long nTimeOut

以毫秒为单位的最长等待时间。此参数是可选参数。缺省值是 INFINITE。

BOOL bWaitForIR

如果此值为 True，在符合等待条件之后，函数将等至 OIA 指示 PS 已准备好接受输入。此参数是可选参数，缺省为 TRUE。

返回值

如果符合条件，方法会返回 TRUE；如果已超过 nTimeOut（以毫秒为单位），则返回 FALSE。

注： 如果 nTimeOut 是缺省值 (INFINITE)，当测试条件返回 FALSE 时，此方法将阻止。

示例

```
// 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

同步等待 ECLScreenDesc 参数描述的屏幕出现在表示空间中。

原型

BOOL WaitForScreen(ECLScreenDesc* screenDesc, long TimeOut = INFINITE)

Parameters

ECLScreenDesc

描述屏幕的 screenDesc 对象（请参阅 ECLScreenDesc 类 (on page 159)）。

long nTimeOut

以毫秒为单位的最长等待时间。此参数是可选参数。缺省值是 INFINITE。

返回值

如果符合条件，方法会返回 TRUE；如果已超过 nTimeOut（以毫秒为单位），则返回 FALSE。

注： 如果 nTimeOut 是缺省值 (INFINITE)，当测试条件返回 FALSE 时，此方法将阻止。

示例

```
// 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

同步等待至 ECLScreenDesc 参数描述的屏幕不再位于表示空间中。

原型

BOOL WaitWhileScreen(ECLScreenDesc* screenDesc, long TimeOut = INFINITE)

Parameters

ECLScreenDesc

描述屏幕的 screenDesc 对象（请参阅 ECLScreenDesc 方法 (on page 159)）。

long nTimeOut

以毫秒为单位的最长等待时间。此参数是可选参数。缺省值是 INFINITE。

返回值

如果符合条件，方法会返回 TRUE；如果已超过 nTimeOut（以毫秒为单位），则返回 FALSE。

注： 如果 nTimeOut 是缺省值 (INFINITE)，当测试条件返回 FALSE 时，此方法将阻止。

示例

```
// 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

此成员函数会注册应用程序对象，以接收 PS 更新事件的通知。要使用此函数，应用程序必须创建从 ECLPSNotify 或 ECLPSListener 派生的对象。接着将指向该对象的指针传递到此注册函数。可以同时注册任意数量的通知或侦听器对象。未定义多个侦听器接收事件的顺序，因此不应假定该顺序。

此函数的不同原型允许生成不同类型的更新事件，以及有关更新的不同级别的详细信息。会使用 ECLPSNotify 对象注册最简单的更新事件。注册类型会为每个 PS 更新生成一个事件。不会生成有关更新的信息。有关更多信息，请参阅 ECLPSNotify 对象的描述。

对于需要有关更新的更多信息的应用程序，可以注册 ECLPSListener 对象。注册此对象使应用程序能够忽略某些类型的更新（例如，击键等本地终端功能），并确定更新的屏幕区域。有关更多信息，请参阅 ECLPSListener 对象的描述。注册 ECLPSListener 对象时，应用程序可以选择指定将导致事件的更新类型。

使用此函数注册 ECLPSNotify 或 ECLPSListener 对象后，每当更新表示空间时，都会调用其 NotifyEvent() 方法。在短时间内对 PS 的多次更新可聚合为一个事件。

应用程序必须先注销通知/侦听器对象，然后才能将其破坏。如果 ECLPS 对象被破坏，将自动注销该对象。

原型

void RegisterPSEvent(ECLPSNotify * notify)
void RegisterPSEvent(ECLPSListener * listener)
void RegisterPSEvent(ECLPSListener * listener, int type)

Parameters

ECLPSNotify *

指向要注册的 ECLPSNotify 对象的指针。

ECLPSListener *

指向要注册的 ECLPSListener 对象的指针。

int

将导致发生事件的更新类型：

- USER_EVENTS（本地终端函数）
- HOST_EVENTS（主机更新）
- ALL_EVENTS（所有更新）

返回值

无人

StartMacro

StartMacro 方法会运行 MacroName 参数指示的 Z and I Emulator for Windows 宏文件。

原型

void StartMacro(String MacroName)

Parameters

String MacroName

位于 Z and I Emulator for Windows 用户类应用程序数据目录（在安装时指定）的宏文件的名称，不带文件扩展名。此方法不支持长文件名。

返回值

无人

使用注意事项

必须将短文件名用于宏名。此方法不支持长文件名。

示例

以下示例展示了如何启动宏。

```
Dim PS as Object

Set PS = CreateObject("ZIEWin.autECLPS")
PS.StartMacro "mymacro"
```

UnregisterPSEvent

此成员函数会注销以前在 RegisterPSEvent 函数中注册的应用程序对象。如果未先调用此函数将其注销，则不应破坏注册为接收事件的对象。如果当前未注册特定对象，则不会执行任何操作，也不会出现错误。

注销 ECLPSNotify 或 ECLPSListener 对象时，将调用其 NotifyStop() 方法。

原型

void UnregisterPSEvent(ECLPSNotify * notify)

void UnregisterPSEvent(ECLPSListener * listener)

void UnregisterPSEvent(ECLPSListener * listener, int type)

Parameters

ECLPSNotify *

指向要注销的 ECLPSNotify 对象的指针。

ECLPSListener *

指向要注销的 ECLPSListener 对象的指针。

int

已注册的更新类型：

- USER_EVENTS（本地终端函数）
- HOST_EVENTS（主机更新）
- ALL_EVENTS（所有更新）

返回值

无人

ECLPSEvent 类

更新表示空间后，会将 ECLPSEvent 对象传递到 ECLListener 对象。此事件对象代表表示空间更新事件，并包含有关更新的信息。

应用程序可以使用两组功能来确定已更新的表示空间区域。GetStart() 和 GetEnd() 方法会返回线性位置，指示更新区域在表示空间中的起始位置和结束位置。对于最左上角的字符，线性寻址从 1 开始，并从左到右逐行换行。一组相应的函数（GetStartRow、GetStartCol、GetEndRow、GetEndCol）会以行/列坐标返回相同的信息。

更新区域包括从起始字符到结束字符（包括）的所有 PS 字符。如果开始和结束位置不在同一行上，则更新区域将从一行的末尾换行到下一行的第一列。请注意，更新区域（通常）不是矩形。如果起始位置大于结束位置，则更新区域从起始位置开始，从屏幕的最后一个字符换行到第一个字符，并继续到结束位置。

请注意，更新区域可能包含的内容超过表示空间的实际更改部分，但保证至少包含更改的区域。当在短时间内发生多次 PS 更新时，更改可能会聚合到单个事件中，其中更新区域跨越所有更新的总和。

派生

ECLBase > ECLEvent > ECLPSEvent

使用注意事项

应用程序不直接使用此类。应用程序会创建在 ECLListener::NotifyEvent 方法上接收 ECLPSEvent 对象的 ECLListener 派生对象。

ECLPSEvent 方法

下节介绍了对 ECLPSEvent 类及其派生的所有类有效的方法。

ECLPS * GetPS()
int GetType()
ULONG GetStart()
ULONG GetEnd()
ULONG GetStartRow()
ULONG GetStartCol()
ULONG GetEndRow()
ULONG GetEndCol()

GetPS

此方法会返回生成此事件的 ECLPS 对象。

原型

ECLPS * GetPS()

Parameters

无人

返回值

ECLPS *

指向生成事件的 ECLPS 对象的指针。

GetType

此方法会返回生成此事件的表示空间更新的类型。返回值为 USER_EVENTS 或 HOST_EVENTS 的 on。用户事件定义为作为本地终端功能发生的任何 PS 更新（例如，用户或编程 API 输入的击键）。主机事件是发生在主机出站数据流中的 PS 更新。

原型

int GetType()

Parameters

无人

返回值

int

返回 USER_EVENTS 或 HOST_EVENTS 常量。

GetStart

此方法会返回更新区域开始位置的表示空间中的线性位置。请注意，此位置的行/列坐标取决于当前为表示空间定义的列数。如果此值大于 GetEnd() 返回的值，则更新区域从此位置开始，在屏幕末尾换行至屏幕开头，并继续到结束位置。

原型

ULONG GetStart()

Parameters

无人

返回值

ULONG

更新区域开始的线性位置。

GetEnd

此方法会返回更新区域结束位置的表示空间中的线性位置。请注意，此位置的行/列坐标取决于当前为表示空间定义的列数。如果此值小于 GetStart() 返回的值，则更新区域从 GetStart() 位置开始，在屏幕末尾换行至屏幕开头，并继续到此位置。

原型

ULONG GetEnd()

Parameters

无人

返回值

ULONG

更新区域结束的线性位置。

GetStartRow

此方法会返回更新区域开始位置的表示空间中的行号。如果起始行/列的位置大于结束行/列的位置，则更新区域从此位置开始，在屏幕末尾换行至屏幕开头，并继续到结束位置。

原型

ULONG GetStartRow()

Parameters

无人

返回值

ULONG

更新区域开始的行号。

GetStartCol

此方法会返回更新区域开始位置的表示空间中的列号。如果起始行/列的位置大于结束行/列的位置，则更新区域从此起始行/列开始，在屏幕末尾换行至屏幕开头，并继续到结束位置。

原型

ULONG GetStartCol()

Parameters

无人

返回值

ULONG

更新区域开始的列号。

GetEndRow

此方法会返回更新区域结束位置的表示空间中的行号。如果起始行/列的位置大于结束行/列的位置，则更新区域从此起始行/列开始，在屏幕末尾换行至屏幕开头，并继续到结束行/列。

原型

ULONG GetEndRow()

Parameters

无人

返回值

ULONG

更新区域结束的行号。

GetEndCol

此方法会返回更新区域结束位置的表示空间中的列号。如果起始行/列的位置大于结束行/列的位置，则更新区域从此起始行/列开始，在屏幕末尾换行至屏幕开头，并继续到结束行/列。

原型

ULONG GetEndCol()

Parameters

无人

返回值

ULONG

更新区域结束的列号。

ECLPSListener 类

ECLPSListener 是抽象基类。应用程序无法直接创建此类的实例。要使用此类，应用程序必须定义其自己的类，该类派生自 ECLPSListener。应用程序必须实现此类中的所有方法。

ECLPSListener 类用于允许应用程序收到有关表示空间更新的通知。更新主机屏幕时会生成事件（在任何平面中更改表示空间中的任何数据）。

此类与 ECLPSNotify 类相似，因为它用于接收 PS 更新的通知。但是，它的不同之处在于会收到比 ECLPSNotify 类多得多的与更新原因和范围相关的信息。通常，使用此类在处理时间和内存方面会更昂贵，因为必须为每个事件生成更多信息。对于需要高效更新主机屏幕可视表示的应用程序，此类比每次更新时重新绘制表示更高效。使用此类，应用程序只能更新已更改的可视表示的部分。

此类也不同于 ECLPSNotify，因为所有方法都纯虚拟，因此必须由应用程序实现（没有任何方法的缺省实现）。

派生

ECLBase > ECLListener > ECLPSListener

使用注意事项

要使用此类获得 PS 更新通知，应用程序必须执行以下步骤：

1. 定义从 ECLPSListener 派生的类。
2. 实现 ECLPSListener 派生类的所有方法。
3. 创建派生类的实例。
4. 使用 ECLPS::RegisterPSEvent() 方法注册实例。

注册完成后，对表示空间的更新将导致调用 ECLPSListener 派生类的 NotifyEvent() 方法。接着，应用程序可以使用方法调用中提供的 ECLPSEvent 对象，确定导致 PS 更新的原因以及受影响的屏幕区域。

请注意，短时间内发生的多次 PS 更新可能会聚合到单个事件通知中。

应用程序可以选择为派生类提供自己的构造函数和析构函数。如果应用程序需要在类中存储某些特定于实例的数据，并将该信息作为参数传递到构造函数上，则此选项非常有用。

如果在事件注册过程中检测到错误，则使用 ECLErr 对象调用 NotifyError() 成员函数。发生错误后，可能会继续生成事件，也可能不会继续生成。当事件生成终止（由于错误或其他原因）时，调用 NotifyStop() 成员函数。

ECLPSListener 方法

下节介绍了对 ECLPSListener 类及其派生的所有类有效的方法。请注意，除构造函数和析构函数之外的所有方法都是纯虚拟方法。

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

此方法是纯虚拟成员函数（应用程序必须在派生自 ECLPSListener 的类中实现此函数）。只要更新表示空间并且注册此对象以接收更新事件，就会调用此方法。作为参数传递的 ECLPSEvent 对象包含有关事件的信息，包括已修改的屏幕区域。请参阅 ECLPSEvent 类 (on page 145) 以获取详细信息。

多个 PS 更新可能会聚集到单个事件中，从而导致仅调用此方法一次。包含在 ECLPSEvent 对象中的已更改区域将包含所有修改的总和。

通过在 ECLPS::RegisterPSEvent() 方法上提供相应的参数，可以将事件限制为仅特定类型的 PS 更新。例如，应用程序可能会选择仅针对来自主机的更新收到通知，而不是针对本地击键收到通知。

原型

virtual void NotifyEvent(ECLPSEvent * event) = 0

Parameters

ECLPSEvent *

指向表示 PS 更新的 ECLPSEvent 对象的指针。

返回值

无人

NotifyError

每当 ECLPS 对象在事件生成过程中检测到错误时，都会调用此方法。错误对象包含有关错误的信息（请参阅 ECLErr 类 (on page 54)）。根据错误的性质，出现错误后可能会继续生成事件。如果事件生成因错误而停止，则将调用 NotifyStop() 方法。

这是应用程序必须实现的纯虚拟方法。

原型

virtual void NotifyError(ECLPS * PSObj, ECLErr ErrObj) = 0

Parameters

ECLPS *

指向生成此事件的 ECLPS 对象的指针。

ECLErr

描述错误的 ECLErr 对象。

返回值

无人

NotifyStop

因任何原因（例如，由于错误条件或调用 ECLPS::UnregisterPSEvent）停止事件生成时，将调用此方法。

这是应用程序必须实现的纯虚拟方法。

原因码参数当前未在使用，将为零。

原型

virtual void NotifyStop(ECLPS * PSObj, int Reason) = 0

Parameters

ECLPS *

指向生成此事件的 ECLPS 对象的指针。

int

事件生成已停止的原因（当前未使用，因此将为零）。

返回值

无人

ECLPSNotify 类

ECLPSNotify 是抽象基类。应用程序无法直接创建此类的实例。要使用此类，应用程序必须定义其自己的类，该类派生自 ECLPSNotify。应用程序必须在其派生类中实现 NotifyEvent() 成员函数。它还可以选择性地实现 NotifyError() 和 NotifyStop() 成员函数。

ECLPSNotify 类用于允许应用程序收到有关表示空间更新的通知。更新主机屏幕时会生成事件（在任何平面中更改表示空间中的任何数据）。

此类与 ECLPSListener 类相似，因为它用于接收 PS 更新的通知。但是，它的不同之处在于不会收到与 ECLPSNotify 类之外的更新原因和范围相关的信息。通常，使用此类在处理时间和内存方面会更高效，因为不必为每个事件生成任何信息。此类可用于仅需要更新通知而不需要事件原因或所更新屏幕部分的详细信息的应用程序。

此类与 ECLPSListener 的不同之处在于为 NotifyError() 和 NotifyStop() 方法提供了缺省实现。

派生

ECLBase > ECLNotify > ECLPSNotify

使用注意事项

要使用此类获得 PS 更新通知，应用程序必须执行以下步骤：

1. 定义从 ECLPSNotify 派生的类。
2. 实现 ECLPSNotify 派生类的 NotifyEvent 方法。
3. （可选）实现 ECLPSNotify 的其他成员函数。
4. 创建派生类的实例。
5. 使用 ECLPS::RegisterPSEvent() 方法注册实例。

注册完成后，对表示空间的更新将导致调用 ECLPSNotify 派生类的 NotifyEvent() 方法。

请注意，短时间内发生的多次 PS 更新可能会聚合到单个事件通知中。

应用程序可以选择为派生类提供自己的构造函数和析构函数。如果应用程序需要在类中存储某些特定于实例的数据，并将该信息作为参数传递到构造函数上，则此选项非常有用。

如果在事件注册过程中检测到错误，则使用 ECLErr 对象调用 NotifyError() 成员函数。发生错误后，可能会继续生成事件，也可能不会继续生成。当事件生成终止（由于错误或其他原因）时，调用 NotifyStop() 成员函数。NotifyError() 的缺省实现将向用户显示一个消息框，其中显示从 ECLErr 对象检索到的错误消息文本。

当事件通知出于任何原因（错误或调用 ECLPS::UnregisterPSEvent）停止时，将调用 NotifyStop() 成员函数。NotifyStop() 的缺省实现不执行任何操作。

ECLPSNotify 方法

下节介绍了对 ECLPSNotify 类及其派生的所有类有效的方法。

ECLPSNotify()=0
~ECLPSNotify()
virtual void NotifyEvent(ECLPS * PSObj)
virtual void NotifyError(ECLPS * PSObj, ECLErr ErrObj)
virtual void NotifyStop(ECLPS * PSObj, int Reason)

NotifyEvent

此方法是纯虚拟成员函数（应用程序**必须**在派生自 ECLPSNotify 的类中实现此函数）。只要更新表示空间并且注册此对象以接收更新事件，就会调用此方法。

多个 PS 更新可能会聚集到单个事件中，从而导致仅调用此方法一次。

原型

virtual void NotifyEvent(ECLPS * PSObj)

Parameters

ECLPS *

指向生成此事件的 ECLPS 对象的指针。

返回值

无人

NotifyError

每当 ECLPS 对象在事件生成过程中检测到错误时，都会调用此方法。错误对象包含有关错误的信息（请参阅 ECLErr 类 (on page 54)）。根据错误的性质，出现错误后可能会继续生成事件。如果事件生成因错误而停止，则将调用 NotifyStop() 方法。

应用程序可以选择实现此功能或允许基本 ECLPSNotify 类处理此功能。缺省实现将使用 ECLErr::GetMsgText() 方法提供的文本在消息框中显示错误。如果应用程序在其派生类中实现此函数，则会覆盖此行为。

原型

virtual void NotifyError(ECLPS * PSObj, ECLErr ErrObj) = 0

Parameters

ECLPS *

指向生成此事件的 ECLPS 对象的指针。

ECLErr

描述错误的 ECLErr 对象。

返回值

无人

NotifyStop

因任何原因（例如，由于错误条件或调用 ECLPS::UnregisterPSEvent）停止事件生成时，将调用此方法。

原因码参数当前未在使用，将为零。

此函数的缺省实现不会执行任何操作。

原型

virtual void NotifyStop(ECLPS * PSObj, int Reason) = 0

Parameters

ECLPS *

指向生成此事件的 ECLPS 对象的指针。

int

事件生成已停止的原因（当前未使用，因此将为零）。

返回值

无人

ECLRecoNotify 类

ECLRecoNotify 可用于实现将接收和处理 ECLScreenReco 事件的对象。只要 PS 中的任何屏幕与 ECLScreenReco 中的 ECLScreenDesc 对象匹配，就会生成事件。当事件生成停止时，以及事件生成过程中发生错误时，将生成特殊事件。

要获得 ECLScreenReco 事件的通知，应用程序必须执行以下步骤：

1. 定义从 ECLRecoNotify 类派生的类。
2. 实现 NotifyEvent()、NotifyStop() 和 NotifyError() 方法。

3. 创建新类的实例。
4. 使用 ECLScreenReco::RegisterScreen() 方法注册实例。

有关示例，请参阅ECLScreenReco 类 (on page 167)。

派生

ECLBase > ECLNotify > ECLRecoNotify

ECLRecoNotify 方法

下面列出了 ECLRecoNotify 的有效方法：

ECLRecoNotify()
~ECLRecoNotify()
void NotifyEvent(ECLPS *ps, ECLScreenDesc *sd)
void NotifyStop(ECLPS *ps, ECLScreenDesc *sd)
void NotifyError(ECLPS *ps, ECLScreenDesc *sd, ECLErr e)

ECLRecoNotify 构造函数

创建 ECLRecoNotify 的空实例。

原型

ECLRecoNotify()

Parameters

无人

返回值

无人

示例

有关示例，请参阅ECLScreenReco 类 (on page 167)。

ECLRecoNotify 析构函数

破坏 ECLRecoNotify 的实例

原型

~ECLRecoNotify()

Parameters

无人

返回值

无人

示例

有关示例，请参阅ECLScreenReco 类 (on page 167)。

NotifyEvent

当使用 ECLScreenReco 上的 ECLRecoNotify 对象注册 ECLScreenDesc 时调用。

原型

void NotifyEvent(ECLPS *ps, ECLScreenDesc *sd)

Parameters

ECLPS ps

您注册的 ECLPS 对象。

ECLScreenDesc sd

您注册的 ECLScreenDesc。

返回值

无人

示例

有关示例，请参阅ECLScreenReco 类 (on page 167)。

NotifyStop

当 ECLScreenReco 对象停止监控其已注册的 ECLScreenDesc 对象的 ECLPS 对象时调用。

原型

void NotifyStop(ECLPS *ps, ECLScreenDesc *sd)

Parameters

ECLPS ps

您注册的 ECLPS 对象。

ECLScreenDesc sd

您注册的 ECLScreenDesc。

返回值

无人

示例

有关示例，请参阅ECLScreenReco 类 (on page 167)。

NotifyError

ECLScreenReco 对象遇到错误时调用。

原型

void NotifyError(ECLPS *ps, ECLScreenDesc *sd, ECLErr e)

Parameters

ECLPS ps

您注册的 ECLPS 对象。

ECLScreenDesc sd

您注册的 ECLScreenDesc。

ECLErr e

包含错误信息的 ECLErr 对象。

返回值

无人

示例

有关示例，请参阅ECLScreenReco 类 (on page 167)。

ECLScreenDesc 类

ECLScreenDesc 类用于描述主机访问类库屏幕识别技术的屏幕。它使用表示空间的所有四个主要平面来描述它（TEXT、FIELD、EXFIELD、COLOR），以及游标位置。

使用此对象上提供的方法，程序员可以设置主机端应用程序中给定屏幕外观的详细描述。创建并设置 ECLScreenDesc 对象后，可以将其传递到同步 ECLPS 上提供的 WaitFor... 方法，也可以将其传递到 ECLScreenReco，后者会在 PS 中显示与 ECLScreenDesc 对象匹配的屏幕时触发异步事件。

派生

ECLBase > ECLScreenDesc

ECLScreenDesc 方法

下面列出了 ECLScreenDesc 的有效方法：

```
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 构造函数

创建 ECLScreenDesc 的空实例。

原型

ECLScreenDesc()

Parameters

无人

返回值

无人

示例

```
// 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 析构函数

破坏 ECLScreenDesc 的实例。

原型

~ ECLScreenDesc()

Parameters

无人

返回值

无人

示例

```
// 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

将给定位置的属性值添加到屏幕描述。

原型

```
void AddAttrib(BYTE attrib, UINT pos, PS_PLANE plane=FieldPlane);
void AddAttrib(BYTE attrib, UINT row, UINT col, PS_PLANE plane=FieldPlane);
```

Parameters

BYTE attrib

要添加的属性值。

int row

行位置。

int col

列位置。

PS_PLANE plane

属性所在的平面。有效值包括：TextPlane、ColorPlane、FieldPlane、Exfield Plane、GridPlane。**TextPlane**、**ColorPlane**、**FieldPlane** 和 **ExfieldPlane**。有关不同 ECL 平面的内容和格式，请参阅 ECL 平面 - 格式和内容 (on page 414)。

返回值

无人

示例

```
// 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

将屏幕描述的游标位置设置为给定位置。

原型

void AddCursorPos(uint row, uint col)

Parameters

uint row

行位置。

uint col

列位置。

返回值

无人

示例

```
// 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

将输入字段数添加到屏幕描述。

原型

void AddNumFields(uint num)

Parameters

uint num

字段数。

返回值

无人

示例

```
// 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

将输入字段数添加到屏幕描述。

原型

void AddNumInputFields(uint num)

Parameters

uint num

输入字段数。

返回值

无人

示例

```
// 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

设置屏幕描述的 OIA 监控类型。

原型

void AddOIAInhibitStatus(OIAStatus type=NOTINHIBITED)

Parameters

OIAStatus type

OIA 状态的类型。当前有效值为 DONTCARE 和 NOTINHIBITED。

返回值

无人

示例

```
// 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

将给定位置的字符串添加到屏幕描述。如果未提供行和列，则字符串可能会出现在 PS 中的任何位置。

注： 负值是 PS 底部的绝对位置。例如，row=-2 是 24 行中的第 23 行。

原型

void AddString(LPCSTR s, UINT row, UINT col, BOOL caseSensitive=TRUE)

Parameters

LPCSTR s

要添加的字符串。

uint row

行位置。

uint col

列位置。

BOOL caseSense

如果此值为 TRUE，则将字符串添加为区分大小写。此参数是可选参数。缺省值为 TRUE。

返回值

无人

示例

```
// 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

将给定矩形中的字符串添加到屏幕描述。

原型

void AddStringInRect(char * str, int Top, int Left, int Bottom, int Right,
BOOL caseSense=TURE)

Parameters

char * str

要添加的字符串。

int Top

左上行位置。此参数是可选参数。缺省值为第一行。

int Left

左上列位置。此参数是可选参数。缺省值为第一列。

int Bottom

右下行位置。此参数是可选参数。缺省值为最后一行。

int Right

右下列位置。此参数是可选参数。缺省值为最后一列。

BOOL caseSense

如果此值为 TRUE，则将字符串添加为区分大小写。此参数是可选参数。缺省值为 TRUE。

返回值

无人

示例

```
// 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());
```

Clear

从屏幕描述中删除所有描述元素。

原型

void Clear()

Parameters

无人

返回值

无人

示例

```
// 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 类

ECLScreenReco类 是主机访问类库屏幕识别系统的引擎。它包含添加和删除屏幕描述的方法。它还包含用于识别这些屏幕以及异步回调这些屏幕的处理程序代码的逻辑。

将 ECLScreenReco 类的对象视为唯一识别集。对象可以有多个 ECLPS 对象，它可以监视屏幕，多个要查找的屏幕，以及当它在任何 ECLPS 对象中看到屏幕时要调用的多个回调点。

您只需在应用程序启动时设置 ECLScreenReco 对象，当任何要监控的 ECLPS 中出现任何屏幕时，ECLScreenReco 就将调用代码。您完全无需监控屏幕！

以下是一个常用实现示例：

```
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 方法

以下方法对 ECLScreenReco 有效：

ECLScreenReco()

~ECLScreenReco()

AddPS(ECLPS*)

IsMatch(ECLPS*, ECLScreenDesc*)

RegisterScreen(ECLScreenDesc*, ECLRecoNotify*)

RemovePS(ECLPS*)

UnregisterScreen(ECLScreenDesc*)

ECLScreenReco 构造函数

创建 ECLScreenReco 的空实例

原型

ECLScreenReco()

Parameters

无人

返回值

无人

示例

请参阅 ECLScreenReco 类 (on page 167) 中提供的常见实现示例。

ECLScreenReco Destructor

破坏 ECLScreenReco 的实例

原型

~ECLScreenReco()

Parameters

无人

返回值

无人

示例

请参阅 ECLScreenReco 类 (on page 167) 中提供的常见实现示例。

AddPS

添加要监控的表示空间对象。

原型

AddPS(ECLPS*)

Parameters

ECLPS*

要监控的 PS 对象。

返回值

无人

示例

请参阅 ECLScreenReco 类 (on page 167) 中提供的常见实现示例。

IsMatch

静态成员方法，可用于传递 ECLPS 对象和 ECLScreenDesc 对象，并确定屏幕描述是否与 PS 匹配。它作为静态方法提供，因此任何例程都可以在不创建 ECLScreenReco 对象的情况下调用它。

原型

IsMatch(ECLPS*, ECLScreenDesc*)

Parameters

ECLPS*

要比较的 ECLPS 对象。

ECLScreenDesc*

要比较的 ECLScreenDesc 对象。

返回值

如果 PS 中的屏幕匹配，则返回 TRUE，否则返回 FALSE。

示例

```
// 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

开始监控添加到给定屏幕描述的屏幕识别对象的所有 ECLPS 对象。如果屏幕出现在 PS 中，则将调用 ECLRecoNotify 对象上的 NotifyEvent 方法。

原型

RegisterScreen(ECLScreenDesc*, ECLRecoNotify*)

Parameters

ECLScreenDesc*

要注册的屏幕描述对象。

ECLRecoNotify*

包含屏幕描述的回调代码的对象。

返回值

无人

示例

请参阅 ECLScreenReco 类 (on page 167) 中提供的常见实现示例。

RemovePS

从屏幕识别监控中删除 ECLPS 对象。

原型

RemovePS(ECLPS*)

Parameters

ECLPS*

要删除的 ECLPS 对象。

返回值

无人

示例

请参阅 ECLScreenReco 类 (on page 167) 中提供的常见实现示例。

UnregisterScreen

从屏幕识别监控中删除屏幕描述及其回调代码。

原型

UnregisterScreen(ECLScreenDesc*)

Parameters

ECLScreenDesc*

要删除的屏幕描述对象。

返回值

无人

示例

请参阅 ECLScreenReco 类 (on page 167) 中提供的常见实现示例。

ECLSession 类

ECLSession 会提供与仿真器连接相关的一般服务，并包含指向主机访问类库中其他对象实例的指针。

派生

ECLBase > ECLConnection > ECLSession

属性

无人

使用注意事项

由于 ECLSession 是从 ECLConnection 派生的，因此可以获取 ECLConnection 对象中包含的所有信息。请参阅 ECLConnection 类 (on page 20) 以获取更多信息。

尽管 ECLSession 包含的对象能够独立运行，但指向它们的指针存在于 ECLSession 类中。创建 ECLSession 对象时，还会创建 ECLPS、ECLOIA、ECLXfer 和 ECLWinMetrics 对象。

ECLSession 方法

下节介绍了对 ECLSession 类有效的方法：

```
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 构造函数

此方法会根据连接名称（A-Z 或 a-z 中的单个字母字符）或连接句柄创建 ECLSession 对象。给定名称只能有一个打开的 Z and I Emulator for Windows 连接。例如，一次只能有一个打开的连接“A”。

原型

ECLSession(char Name)

ECLSession(long Handle)

Parameters

char Name

连接的单字符短名称（A-Z 或 a-z）。

long Handle

ECL 连接的句柄。

返回值

无人

示例

```
//---------------------------------------------------------------------
// 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 Destructor

此方法会破坏 ECLSession 对象。

原型

~ECLSession();

Parameters

无人

返回值

无人

示例

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

此方法会返回指向 ECLSession 对象中包含的 ECLPS 对象的指针。使用此方法可访问 ECLPS 对象方法。请参阅 ECLPS 类 (on page 102) 以获取更多信息。

原型

ECLPS *GetPS()

Parameters

无人

返回值

ECLPS *

ECLPS 对象指针。

示例

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

此方法会返回指向 ECLSession 对象中包含的 ECLOIA 对象的指针。使用此方法可访问 ECLOIA 方法。请参阅 ECLOIA 类 (on page 86) 以获取更多信息。

原型

ECLOIA *GetOIA()

Parameters

无人

返回值

ECLOIA *

ECLOIA 对象指针。

示例

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

此方法会返回指向 ECLSession 对象中包含的 ECLXfer 对象的指针。使用此方法可访问 ECLXfer 方法。请参阅 ECLXfer 类 (on page 209) 以获取更多信息。

原型

ECLXfer *GetXfer()

Parameters

无人

返回值

ECLXfer *

ECLXfer 对象指针。

示例

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

此方法会返回指向 ECLSession 对象中包含的 ECLWinMetrics 对象的指针。使用此方法可访问 ECLWinMetrics 方法。请参阅 ECLWinMetrics 类 (on page 186) 以获取更多信息。

原型

ECLWinMetrics *GetWinMetrics()

Parameters

无人

返回值

ECLWinMetrics *

ECLWinMetrics 对象指针。

示例

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

此方法会返回指向 ECLSession 对象中包含的 ECLPageSettings 对象的指针。使用此方法可访问 ECLPageSettings 方法。请参阅 ECLPageSettings 类 (on page 215) 以获取更多信息。

原型

ECLPageSettings *GetPageSettings() const;

Parameters

无人

返回值

ECLPageSettings *

ECLPageSettings 对象指针。

示例

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

此方法会返回指向 ECLSession 对象中包含的 ECLPrinterSettings 对象的指针。使用此方法可访问 ECLPrinterSettings 方法。请参阅 ECLPageSettings 类 (on page 215) 以获取更多信息。

原型

ECLPrinterSettings *GetPrinterSettings() const;

Parameters

无人

返回值

ECLPrinterSettings *

ECLPrinterSettings 对象指针。

示例

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

已弃用。请参阅 RegisterPSEvent (on page 143) 中的 ECLPS::RegisterPSEvent。

UnregisterUpdateEvent

已弃用。请参阅 UnregisterPSEvent (on page 145) 中的 ECLPS::UnregisterPSEvent。

ECLStartNotify 类

ECLStartNotify 是抽象基类。应用程序无法直接创建此类的实例。要使用此类，应用程序必须定义自派生自 ECLStartNotify 的类。应用程序必须在其派生类中实现 NotifyEvent() 成员函数。它还可以选择性地实现 NotifyError() 和 NotifyStop() 成员函数。

ECLStartNotify 类用于允许应用程序在 ZIEWin 连接启动和停止时收到通知。无论何时以任何方式（包括 ECLConnMgr 启动/停止方法）启动或停止 ZIEWin 连接（窗口），都会生成启动/停止事件。

要收到启动/停止事件的通知，应用程序必须执行以下步骤：

1. 定义派生自 ECLStartNotify 的类。
2. 实现派生类并实现 NotifyEvent() 成员函数。
3. （可选）实现 NotifyError() 和/或 NotifyStop() 函数。
4. 创建派生类的实例。
5. 使用 ECLConnMgr::RegisterStartEvent() 函数注册实例。

显示的示例演示了如何执行此操作。完成上述步骤后，每次启动或停止连接时，都将调用应用程序 NotifyEvent() 成员函数。会向该函数传递两个参数，以提供连接的句柄和 BOOL 启动/停止指示符。应用程序可以执行 NotifyEvent() 过程中所需的任何函数，包括调用其他 ECL 函数。请注意，应用程序不能阻止连接停止；通知是在会话已停止后发出的。

如果在事件生成过程中检测到错误，则使用 ECLErr 对象调用 NotifyError() 成员函数。发生错误后，可能会继续生成事件，也可能不会继续生成，具体取决于错误的性质。当事件生成终止时（由于错误，通过调用

ECLConnMgr::UnregisterStartEvent，或由于破坏 ECLConnMgr 对象），将调用 NotifyStop() 成员函数。但是，事件通知会被终止，NotifyStop() 成员函数总是被调用，并且应用程序对象会被注销。

如果应用程序不提供 NotifyError() 成员函数的实现，则使用缺省实现（向用户显示简单的消息框）。应用程序可以通过在应用程序派生类中实现 NotifyError() 函数来覆盖缺省行为。同样，如果应用程序不提供此函数（缺省行为是不执行任何操作），则使用缺省 NotifyStop() 函数。

请注意，应用程序还可以选择为派生类提供自己的构造函数和析构函数。如果应用程序要在类中存储某些特定于实例的数据，并将该信息作为参数传递到构造函数上，则此选项非常有用。例如，当发生启动/停止事件时，应用程序可能希望将消息发布到应用程序窗口。应用程序不是将窗口句柄定义为全局变量（因此它对 NotifyEvent() 函数可见），而是可以为类定义构造函数，该类接受窗口句柄并将其存储在类成员数据区域中。

应用程序在注册接收事件时不能破坏通知对象。

实现限制：目前，ECLConnMgr 对象仅允许为一个启动/停止事件通知注册一个通知对象。如果已为该 ECLConnMgr 对象注册了通知对象，则 ECLConnMgr::RegisterStartEvent 将抛出错误。

派生

ECLBase > ECLNotify > ECLStartNotify

示例

```
//---------------------------------------------------------------------
// 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 方法

下节介绍了对 ECLStartNotify 类有效的方法。

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

此方法是纯虚拟成员函数（应用程序必须在派生自 ECLStartNotify 的类中实现此函数）。每当连接开始或停止并且为开始/停止事件注册了对象时，都会调用此函数。如果连接已启动，则 Started BOOL 为 TRUE；如果已停止，则为 FALSE。

原型

virtual int NotifyEvent (ECLConnMgr *CMObj, long ConnHandle,
BOOL Started) = 0

Parameters

ECLConnMgr *CMObj

这是指向发生事件的 ECLConnMgr 对象的指针。

long ConnHandle

这是已启动或已停止的连接的句柄。

BOOL Started

如果连接已启动，则这为 TRUE；如果连接已停止，则为 FALSE。

返回值

无人

NotifyError

每当 ECLConnMgr 对象在事件生成过程中检测到错误时，都会调用此方法。错误对象包含有关错误的信息（请参阅 ECLErr 类描述）。根据错误的性质，出现错误后可能会继续生成事件。如果事件生成因错误而停止，则将调用 NotifyStop() 函数。

ConnHandle 包含与错误相关的连接句柄。如果错误与任何特定连接无关，则此值可能为零。

应用程序可以选择实现此函数或允许 ECLStartNotify 基类处理错误。基类将使用 ECLErr::GetMsgText() 函数提供的文本在消息框中显示错误。如果应用程序在其派生类中实现此函数，它将覆盖基类函数。

原型

virtual void NotifyError（ECLConnMgr *CMObj, long ConnHandle,
ECLErr ErrObject)

Parameters

ECLConnMgr *CMObj

这是指向出错的 ECLConnMgr 对象的 ptr。

long ConnHandle

这是与错误或零相关的连接的句柄。

ECLErr ErrObject

这是描述错误的 ECLErr 对象。

返回值

无人

NotifyStop

因任何原因（例如，由于错误条件或调用 ECLConnMgr::UnregisterStartEvent）停止事件生成时，将调用此方法。

原型

virtual void NotifyStop （ECLConnMgr *CMObj int Reason)

Parameters

ECLConnMgr *CMObj

这是指向正在停止通知的 ECLConnMgr 对象的 ptr。

int Reason

这是未使用的零。

返回值

无人

ECLUpdateNotify 类

已弃用。请参阅ECLPSListener 类 (on page 150)和ECLOIA 类 (on page 86)中的类描述。

ECLWinMetrics 类

ECLWinMetrics 类在 Z and I Emulator for Windows 连接窗口上执行操作。它允许您执行窗口矩形和位置操控（例如 SetWindowRect、GetXpos 或 SetWidth），以及窗口状态操控（例如 SetVisible 或 IsRestored）。

派生

ECLBase > ECLConnection > ECLWinMetrics

属性

无人

使用注意事项

由于 ECLWinMetrics 是从 ECLConnection 派生的，因此可以获取 ECLConnection 对象中包含的所有信息。请参阅 ECLConnection 类 (on page 20) 以获取更多信息。

为构造时标识的连接创建 ECLWinMetrics 对象。可以通过传递连接标识（A-Z 或 a-z 中的单个字母字符）或连接句柄（通常从 ECLConnection 对象获得）来创建 ECLWinMetrics 对象。一次只能打开一个具有给定名称或句柄的 Z and I Emulator for Windows 连接。

注： 在 ECLSession 类中存在指向 ECLWinMetrics 对象的指针。如果只想处理连接窗口，请自行创建 ECLWinMetrics。如果要执行更多操作，可能需要创建 ECLSession 对象。

ECLWinMetrics 方法

以下方法适用于 ECLWinMetrics 类。

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 构造函数

此方法会根据连接名称或连接句柄创建 ECLWinMetrics 对象。给定名称只能有一个打开的 Z and I Emulator for Windows 连接。例如，一次只能有一个打开的连接“A”。

原型

ECLWinMetrics(char Name)

ECLWinMetrics(long Handle)

Parameters

char Name

连接的单字符短名称（A-Z 或 a-z）。

long Handle

ECL 连接的句柄。

返回值

无人

示例

```
//---------------------------------------------------------------------
// 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 析构函数

此方法会破坏 ECLWinMetrics 对象。

原型

~ECLWinMetrics()

Parameters

无人

返回值

无人

示例

```
//---------------------------------------------------------------------
// 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 方法会返回指向 null 终止字符串的指针，该字符串包含当前位于与 ECLWinMetrics 对象关联的连接的标题栏中的标题。不要假定返回的字符串在一段时间内是持久的。您必须复制字符串，或者在每次需要时调用此方法。

原型

const char *GetWindowTitle()

Parameters

无人

返回值

指向包含标题的以 null 结束的字符串的指针。

示例

```
//-------------------------------------------------------------------
// 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 方法会将与 ECLWinMetrics 对象关联的连接的标题栏中的当前标题更改为在输入参数中传递的标题。null 字符串可用于将标题重置为缺省标题。

原型

void SetWindowTitle(char *NewTitle)

Parameters

char *NewTitle

以 null 结束的标题字符串。

返回值

无人

示例

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

使用注意事项

如果 NewTitle 是空字符串，SetWindowTitle 会将窗口标题恢复为其原始设置。

GetXpos

GetXpos 方法会返回连接窗口矩形左上点的 *x* 位置。

原型

long GetXpos()

Parameters

无人

返回值

long

连接窗口的 *x* 位置。

示例

```
//---------------------------------------------------------------------
// 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 方法会设置连接窗口矩形左上点的 *x* 位置。

原型

void SetXpos(long NewXpos)

Parameters

long NewXpos

窗口矩形的新 *x* 坐标。

返回值

无人

示例

```
//---------------------------------------------------------------------
// 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 方法会返回连接窗口矩形左上点的 *y* 位置。

原型

long GetYpos()

Parameters

无人

返回值

long

连接窗口的 *y* 位置。

示例

```
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 方法会设置连接窗口矩形左上点的 *y* 位置。

原型

void SetYpos(long NewYpos)

Parameters

long NewYpos

窗口矩形的新 *y* 坐标。

返回值

无人

示例

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

此方法会返回连接窗口矩形的宽度。

原型

long GetWidth()

Parameters

无人

返回值

long

连接窗口的宽度。

示例

```
//---------------------------------------------------------------------
// 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 方法会设置连接窗口矩形的宽度。

原型

void SetWidth(long NewWidth)

Parameters

long NewWidth

窗口矩形的新宽度。

返回值

无人

示例

```
//-------------------------------------------------------------------
// 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 方法会返回连接窗口矩形的高度。

原型

long GetHeight()

Parameters

无人

返回值

long

连接窗口的高度。

示例

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

此方法会设置连接窗口矩形的高度。

原型

void SetHeight(Long NewHeight)

Parameters

long NewHeight

窗口矩形的新高度。

返回值

无人

示例

以下示例展示了如何使用 SetHeight 方法设置连接窗口矩形的高度。

```
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

此方法会返回连接窗口矩形的边界点。

原型

void GetWindowRect(Long *left, Long *top, Long *right, Long *bottom)

Parameters

long *left

此输出参数设置为窗口矩形的左坐标。

long *top

此该输出参数设置为窗口矩形的顶坐标。

long *right

此输出参数设置为窗口矩形的右坐标。

long *bottom

此输出参数设置为窗口矩形的底坐标。

返回值

无人

示例

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

此方法会设置连接窗口矩形的边界点。

原型

void SetWindowRect(long left, long top, long right, long bottom)

Parameters

long left

窗口矩形的左坐标。

long top

窗口矩形的顶坐标。

long right

窗口矩形的右坐标。

long bottom

窗口矩形的底坐标。

返回值

无人

示例

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

此方法会返回连接窗口的可见性状态。

原型

BOOL IsVisible()

Parameters

无人

返回值

可见性状态。如果窗口可见，则返回 TRUE 值；如果窗口不可见，则返回 FALSE 值。

示例

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

此方法会设置连接窗口的可见性状态。

原型

void SetVisible(BOOL SetFlag)

Parameters

BOOL SetFlag

TRUE 表示可见，FALSE 表示不可见。

返回值

无人

示例

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

此方法会返回连接窗口的焦点状态。

原型

BOOL Active()

Parameters

无人

返回值

BOOL

焦点状态。如果处于活动状态，则返回 TRUE；如果未处于活动状态，则返回 FALSE。

示例

```
// 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

此方法会设置连接窗口的焦点状态。

原型

void SetActive(BOOL SetFlag)

Parameters

Bool SetFlag

新状态。如果处于活动状态，则返回 TRUE；如果处于非活动状态，则返回 FALSE。

返回值

无人

示例

以下是 SetActive 方法的示例。

```
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

此方法会返回连接窗口的最小化状态。

原型

BOOL IsMinimized()

Parameters

无人

返回值

BOOL

最小化状态。如果窗口已最小化，则返回 TRUE 值；如果窗口未最小化，则返回 FALSE 值。

示例

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

此方法会将连接窗口设置为 minimized

原型

void SetMinimized()

Parameters

无人

返回值

无人

示例

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

此方法会返回连接窗口的最大化状态。

原型

BOOL IsMaximized()

Parameters

无人

返回值

BOOL

最大化状态。如果窗口已最大化，则返回 TRUE 值；如果窗口未最大化，则返回 FALSE 值。

示例

```
// 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

此方法会将连接窗口设置为 maximized。

原型

void SetMaximized()

Parameters

无人

返回值

无人

示例

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

此方法会返回连接窗口的恢复状态。

原型

BOOL IsRestored()

Parameters

无人

返回值

BOOL

恢复状态。如果窗口已恢复，则返回 TRUE 值；如果窗口未恢复，则返回 FALSE 值。

示例

```
//---------------------------------------------------------------------
// 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 方法会将连接窗口设置为 restored。

原型

void SetRestored()

Parameters

无人

返回值

无人

示例

```
//---------------------------------------------------------------------
// 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 类

ECLXfer 提供文件传输服务。

派生

ECLBase > ECLConnection > ECLXfer

属性

无人

使用注意事项

由于 ECLXfer 是从 ECLConnection 派生的，因此可以获取 ECLConnection 对象中包含的所有信息。请参阅 ECLConnection 类 (on page 20) 以获取更多信息。

为构造时标识的连接创建 ECLXfer 对象。可以通过传递连接标识（A-Z 或 a-z 中的单个字母字符）或连接句柄（通常从 ECLConnList 对象获得）来创建 ECLXfer 对象。一次只能打开一个具有给定名称或句柄的 Z and I Emulator for Windows 连接。

注： 在 ECLSession 类中存在指向 ECLXfer 对象的指针。如果只想处理连接窗口，请自行创建 ECLXfer 对象。如果要执行更多操作，可能需要创建 ECLSession 对象。

ECLXfer 方法

下节介绍了对 ECLXfer 类有效的方法：

ECLXfer(char Name)
ECLXfer(long Handle)
~ECLXfer()
int SendFile(char *PCFile, char *HostFile, char *Options)
int ReceiveFile(char *PCFile, char *HostFile, char *Options)

ECLXfer 构造函数

此方法会根据连接标识（A-Z 或 a-z 中的单个字母字符）或连接句柄创建 ECLXfer 对象。给定标识只能有一个打开的 Z and I Emulator for Windows 连接。例如，一次只能有一个打开的连接“A”。

原型

ECLXfer(char Name)

ECLXfer(long Handle)

Parameters

char Name

连接的单字符短名称（A-Z 或 a-z）。

long Handle

ECL 连接的句柄。

返回值

无人

示例

```
//---------------------------------------------------------------------
// 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 析构函数

此方法将破坏 ECLXfer 对象。

原型

~ECLXfer();

Parameters

无人

返回值

无人

示例

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

此方法会将文件从工作站发送到主机。

原型

int SendFile(char *PCFile, char *HostFile, char *Options)

Parameters

char *PCFile

指向包含要发送到主机的工作站文件名的字符串的指针。

char *HostFile

指向包含要在主机上创建或更新的主机文件名的字符串的指针。

char *Options

指向包含要在传输过程中使用的选项的字符串的指针。

返回值

int

SendFile EHLLAPI 函数的 EHLLAPI 返回码，如 *Emulator Programming* 中所述。

示例

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

使用注意事项

文件传输选项取决于主机。以下是 VM/CMS 主机的一些有效主机选项的列表：

ASCII
CRLF
APPEND
LRECL
RECFM
CLEAR/NOCLEAR
PROGRESS
QUIET

请参阅 *Emulator Programming* 以获取受支持的主机和关联文件传输选项的列表。

ReceiveFile

此方法会从主机接收文件并将文件发送到工作站。

原型

int ReceiveFile(char *PCFile, char *HostFile, char *Options)

Parameters

char *PCFile

指向包含要发送到主机的工作站文件名的字符串的指针。

char *HostFile

指向包含要在主机上创建或更新的主机文件名的字符串的指针。

char *Options

指向包含要在传输过程中使用的选项的字符串的指针。

返回值

int

ReceiveFile EHLLAPI 函数的 EHLLAPI 返回码，如 *Emulator Programming* 中所述。

示例

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

使用注意事项

文件传输选项取决于主机。以下是 VM/CMS 主机的一些有效主机选项的列表：

ASCII
CRLF
APPEND
LRECL
RECFM
CLEAR/NOCLEAR
PROGRESS
QUIET

请参阅 *Emulator Programming* 以获取受支持的主机和关联文件传输选项的列表。

ECLPageSettings 类

ECLPageSettings 类会对会话页面设置执行操作。它让您可检索和配置**文件 → 页面设置**对话框设置，如 CPI、LPI 和字体名称。仅支持对话框的**文本**选项卡中的设置。

派生

ECLBase > ECLConnection > ECLPageSettings

属性

无人

限制

与每个方法关联的连接都必须处于特定状态，方法才能成功。如果未满足这些限制，则会引发相应的例外。

以下限制在调用 ECLPageSettings 类的任何方法时适用。如果未满足这些限制，则会抛出异常。

- 连接**页面设置**和**打印机设置**对话框不能在使用中。
- 连接不得在打印。
- 关联的连接不能处于 PDT 方式。

可能对每种特定方法都有附加限制。

使用注意事项

由于 ECLPageSettings 是从 ECLConnection 派生的，因此可以获取 ECLConnection 对象中包含的所有信息。请参阅 ECLConnection 类 (on page 20) 以获取更多信息。

为构造时标识的连接创建 ECLPageSettings 对象。可以通过传递连接标识（**A** 至 **Z** 中的单个字母字符）或连接句柄（通常从 ECLConnection 对象获得）来创建 ECLPageSettings 对象。一次只能打开一个具有给定名称或句柄的 Z and I Emulator for Windows 连接。

ECLSession 类会创建此对象的实例。如果应用程序不需要 ECLSession 提供的其他服务，则可以独立创建此对象。否则，请考虑创建 ECLSession 对象并使用由 ECLSession 创建的对象。请参阅 ECLSession 类 (on page 173) 以获取更多信息。

每种方法仅支持与 ECLPageSettings 对象关联的连接的特定连接类型。每个方法部分都提供受支持的连接类型。如果在不受支持的连接上调用方法，则会抛出异常。使用 GetConnType 方法确定连接类型。

CPI、LPI 和 FontSize 取决于属性 FaceName。因此，如果在设置 FaceName 之前设置了 CPI、LPI 或 FontSize，并且值对 FaceName 属性无效，则可能会在连接中重新配置不同的 CPI、LPI 或 FontSize 值。您应该在设置 CPI、LPI 或 FontSize 之前设置 FaceName 值。或者，您可以在每次设置 FaceName 时查询 CPI、LPI 和 FontSize，以确保它们使用所需的值。

ECLPageSettings 方法

以下各节介绍了对 ECLPageSettings 类有效的方法。

```
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
```

连接类型

ECLPageSettings 方法的有效连接类型如下所示：

连接类型	字符串值
3270 显示	HOSTTYPE_3270DISPLAY
5250 显示	HOSTTYPE_5250DISPLAY
3270 打印机	HOSTTYPE_3270PRINTER
VT (ASCII) 仿真	HOSTTYPE_VT

ECLPageSettings 构造函数

此方法会使用连接名称或句柄来创建 ECLPageSettings 对象。

原型

ECLPageSettings(char Name)

ECLPageSettings(long Handle)

Parameters

char Name

连接的单字符短名称。有效值为 A–Z。

long Handle

ECL 连接的句柄。

返回值

无人

示例

以下示例展示了如何使用连接名称和连接句柄创建 ECLPageSettings 对象。

```
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

此方法会设置连接中的 CPI（每英寸字符数）值。如果调用此方法时不带任何参数，则会在连接中设置字体 CPI。

原型

void SetCPI(ULONG CPI=FONT_CPI);

Parameters

ULONG CPI

字符/英寸。此参数是可选参数。缺省值为 FONT_CPI。

返回值

无人

示例

```
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

此方法会返回连接的 CPI（每英寸字符数）值。即使在关联连接中选择**字体 CPI**，此方法也会返回为关联连接中的字体选择的 CPI 值。

如果在连接中配置了**字体 CPI**，则此方法不会返回常量 FONT_CPI。使用 IsFontCPI 方法确定是否在连接中设置**字体 CPI**。

原型

ULONG GetCPI() const;

Parameters

无人

返回值

ULONG CPI

字符/英寸。

示例

```
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

此方法会返回是否在连接中设置**字体 CPI** 的指示。

原型

BOOL IsFontCPI();

Parameters

无人

返回值

BOOL

有效值如下：

- 如果在连接中设置了**字体 CPI**，则返回 TRUE。
- 如果未在连接中设置**字体 CPI**，则返回 FALSE。

示例

```
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

此方法会设置连接中的 LPI（每英寸行数）值。如果调用此方法时不带任何参数，则会在连接中设置字体 LPI。

原型

void SetLPI(ULONG LPI=FONT_LPI);

Parameters

ULONG LPI

行/英寸。此参数是可选参数。缺省值为 FONT_LPI。

返回值

无人

示例

```
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

此方法会返回连接的 LPI（每英寸行数）值。即使在关联连接中选择**字体 LPI**，此方法也会返回为关联连接中的字体选择的 LPI 值。

如果在连接中配置了**字体 LPI**，则此方法不会返回常量 FONT_LPI。使用 IsFontLPI 方法确定是否在连接中设置**字体 LPI**。

原型

ULONG GetLPI() const;

Parameters

无人

返回值

ULONG LPI

行/英寸。

示例

```
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

此方法会返回是否在关联连接中设置**字体 LPI** 的指示。

原型

BOOL IsFontLPI();

Parameters

无人

返回值

BOOL

有效值如下：

- 如果在连接中设置了**字体 LPI**，则返回 TRUE。
- 如果未在连接中设置**字体 LPI**，则返回 FALSE。

示例

```
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

此方法会设置连接中的字体。

原型

void SetFontFaceName(const char *const FaceName);

Parameters

char *FaceName

包含字体名称的以 null 结束的字符串。

返回值

无人

示例

```
void Sample111() {

   ECLPageSettings PgSet('A');
   const char *Face;

   PgSet.SetFontFaceName("Courier New");
```

```
   Face = PgSet.GetFontFaceName();
   printf("FaceName = %s\n", Face);
} // end sample
```

GetFontFaceName

此方法会返回指向以 null 结束的字符串的指针。该字符串包含当前在页面设置中为与 ECLPageSettings 对象关联的连接选择的字体名称。此方法可能不会每次都返回相同的字符串。

该字符串仅在对象的生命周期内有效。您必须复制字符串，或者在每次需要时调用此方法。

原型

const char *GetFontFaceName() const;

Parameters

无人

返回值

char *

指向包含字体名称的以 null 结束的字符串的指针。

示例

```
void Sample111() {

   ECLPageSettings PgSet('A');
   const char *Face;

   PgSet.SetFontFaceName("Courier New");
   Face = PgSet.GetFontFaceName();
   printf("FaceName = %s\n", Face);
} // end sample
```

SetFontSize

此方法会设置字体的大小。

原型

void SetFontSize(ULONG FontSize);

Parameters

ULONG FontSize

要在连接中设置的字体大小。

返回值

无人

SetMaxLinesPerPage

此方法会设置每页可打印的最大行数。

原型

void SetMaxLinesPerPage(ULONG MPL);

Parameters

ULONG MPL

每页最大行数（最大打印行数）。有效值范围为 1-255。

返回值

无人

示例

```
void Sample113() {

   ECLPageSettings PgSet('A');

   PgSet.SetMaxLinesPerPage(40);
   ULONG MPL = PgSet.GetMaxLinesPerPage();
   printf("MaxLinesPerPage = %ld\n", MPL);
} // end sample
```

GetMaxLinesPerPage

此方法会返回每页可打印的最大行数。

原型

ULONG GetMaxLinesPerPage() const;

Parameters

无人

返回值

ULONG

每页最大行数（最大打印行数）。

示例

```
void Sample113() {

   ECLPageSettings PgSet('A');

   PgSet.SetMaxLinesPerPage(40);
   ULONG MPL = PgSet.GetMaxLinesPerPage();
   printf("MaxLinesPerPage = %ld\n", MPL);
} // end sample
```

SetMaxCharsPerLine

此方法会设置每行可打印的最大字符数。

原型

void SetMaxCharsPerLine(ULONG MPP);

Parameters

ULONG MPP

每行可打印的最大字符数（最大打印位置）。有效值范围为 1-255。

返回值

无人

示例

```
void Sample114() {

   ECLPageSettings PgSet('A');

   PgSet.SetMaxCharsPerLine(50);
   ULONG MPP = PgSet.GetMaxCharsPerLine();
   printf("MaxCharsPerLine=%ld\n", MPP);
} // end sample
```

GetMaxCharsPerLine

此方法会返回每行可打印的最大字符数。

原型

ULONG GetMaxCharsPerLine() const;

Parameters

无人

返回值

ULONG

每行可打印的最大字符数（最大打印位置）。

示例

```
void Sample114() {

   ECLPageSettings PgSet('A');

   PgSet.SetMaxCharsPerLine(50);
   ULONG MPP = PgSet.GetMaxCharsPerLine();
   printf("MaxCharsPerLine=%ld\n", MPP);
} // end sample
```

RestoreDefaults

此方法会恢复在 PageSetup 面板的 nFlags 字段中指定的属性页的系统缺省值。这等同于单击连接**页面设置**对话框属性页面中的**缺省值**按钮。

原型

void RestoreDefaults(ULONG Flags=PAGE_TEXT) const;

Parameters

ULONG Flags

此参数是可选参数。以下标志说明指定的**页面设置**对话框属性页面的名称。此标志可以按位进行 OR 运算，以恢复属性页面（在 PCSAPI32.H 中定义）。

PAGE_TEXT

此标志说明“文本”属性页面。这是当前支持的唯一属性页面。

返回值

无人

示例

```
void Sample115() {

   ECLPageSettings PgSet('A');

   PgSet.RestoreDefaults(PAGE_TEXT);
} // end sample
```

ECLPrinterSettings 类

ECLPrinterSettings 类对 Z and I Emulator for Windows 连接的打印机设置执行操作。它让您可检索和配置**文件 → 打印机设置**对话框设置，如打印机和 PDT 方式。

派生

ECLBase > ECLConnection > ECLPrinterSettings

属性

无人

限制

与每个方法关联的连接都必须处于特定状态，方法才能成功。如果未满足这些限制，则会引发相应的例外。

以下限制在调用 ECLPrinterSettings 类的任何方法时适用。如果未满足这些限制，则会抛出异常。

- 连接**页面设置**和**打印机设置**对话框不能在使用中。
- 连接不得在打印。

可能对每种特定方法都有附加限制。

使用注意事项

由于 ECLPrinterSettings 是从 ECLConnection 派生的，因此可以获取 ECLConnection 对象中包含的所有信息。请参阅 ECLConnection 类 (on page 20) 以获取更多信息。

为构造时标识的连接创建 ECLPrinterSettings 对象。可以通过传递连接标识（**A** 至 **Z** 中的单个字母字符）或连接句柄（通常从 ECLConnection 对象获得）来创建 ECLPrinterSettings 对象。一次只能打开一个具有给定名称或句柄的 Z and I Emulator for Windows 连接。

ECLSession 类会创建此对象的实例。如果应用程序不需要 ECLSession 提供的其他服务，则可以独立创建此对象。否则，请考虑创建 ECLSession 对象并使用由 ECLSession 创建的对象。请参阅 ECLSession 类 (on page 173) 以获取更多信息。

ECLPrinterSettings 方法

以下各节介绍了对 ECLPrinterSettings 类有效的方法。

```
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 构造函数

此方法会使用连接名称或句柄来创建 ECLPrinterSettings 对象。

原型

ECLPrinterSettings(char Name)

ECLPrinterSettings(long Handle)

Parameters

char Name

连接的单字符短名称。有效值为 A–Z。

long Handle

ECL 连接的句柄。

返回值

无人

示例

以下示例展示了如何使用连接名称和连接句柄创建 ECLPrinterSettings 对象。

```
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

此方法会在 PDT 方式下设置与给定 PDT 文件的连接，或者在非 PDT 方式（GDI 方式）下设置连接。

注： 如果在 PDTMode 设置为 FALSE 的情况下调用此方法，则关联连接的 PrintMode 必须已为 SpecificPrinter 或 WinDefaultPrinter。

原型

void SetPDTMode(BOOL PDTMode=TRUE, const char *const PDTFile = NULL);

Parameters

BOOL PDTMode

此参数是可选参数。有效值如下：

- **TRUE**，将连接设置为 PDT 方式。这是缺省值。
- **FALSE**，将连接设置为非 PDT 方式。

char *PDTFile

包含 PDT 文件名的以 null 结束的字符串。

此参数是可选参数。仅当 PDTMode 为 TRUE 时才使用此参数。如果 PDTMode 为 FALSE，则忽略该参数。

有效值如下：

- NULL

  会使用连接中配置的 PDT 文件。如果尚未在连接中配置任何 PDT 文件，则此方法将失败并出现异常。这是缺省值。
- 不带路径的文件名

  使用 Z and I Emulator for Windows 安装路径中 PDFPDT 子文件夹中的 PDTFile。
- 文件的全限定路径名

  如果 PDTFile 不存在，则此方法将失败并出现异常。

返回值

无人

示例

```
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

此方法会返回在连接中配置的 PDT 文件。此方法可能不会每次都返回相同的字符串。

该字符串仅在对象的生命周期内有效。您必须复制字符串，或者在每次需要时调用此方法。

原型

const char *GetPDTFile() const;

Parameters

无人

返回值

char *

有效值如下：

- 包含连接的 PDT 文件的全限定路径名的以 null 结束的字符串。
- 如果未在连接中配置 PDT 文件，则返回 **NULL**。

示例

```
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

此方法会返回连接的 PDT 方式的状态。

原型

BOOL IsPDTMode() const;

Parameters

无人

返回值

BOOL

有效值如下：

- 如果连接处于 PDT 方式，则返回 TRUE。
- 如果连接未处于 PDT 方式，则返回 FALSE。

示例

```
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

此方法会返回指示连接的 PrintMode 的枚举值。枚举数据类型 ECLPrinterSettings::PrintMode 在 ECLPRSET.HPP 中定义。

PrintMode 可以是以下一项：

- **PrtToDskAppend**（**打印到磁盘 - 附加**方式）

  这相当于在主机会话的**打印机设置 → 打印机 → 打印到磁盘**对话框中选择**追加**选项。
- **PrtToDskSeparate**（**打印到磁盘 - 单独**方式）

  这相当于在主机会话的**打印机设置 → 打印机 → 打印到磁盘**对话框中选择**单独**选项。
- **WinDefaultPrinter**（**Windows 缺省打印机**方式）

  这相当于在主机会话**打印机设置**对话框中选择**使用 Windows 缺省打印机**选项。

- **SpecificPrinter（特定打印机方式）**

  这相当于在主机会话**打印机设置**对话框中选择打印机，而不选中**使用 Windows 缺省打印机。**

原型

ECLPrinterSettings::PrintMode GetPrintMode() const;

Parameters

无人

返回值

ECLPrinterSettings::PrintMode

在 ECLPRSET.HPP 中定义的一个 PrintMode 值。

示例

```
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

此方法会将 PrintMode 设置为**打印到磁盘 - 附加**方式，并为此方式设置相应的文件。

注:

1. 关联的连接必须处于 PDT 方式。
2. 要设置此文件的文件夹必须具有写入访问权。否则，此方法将失败并出现异常。
3. 如果该文件存在，则将使用它。否则，将在打印完成时创建此文件。

原型

void SetPrtToDskAppend(const char *const FileName = NULL);

Parameters

char *FileName

包含**打印到磁盘 - 附加**文件的名称的以 null 结束的字符串。此参数是可选参数。
有效值如下：

- NULL

  使用连接中当前为此 PrintMode 配置的文件。如果尚未在连接中配置文件，则此方法将失败并出现异常。这是缺省值。
- 文件名，不带路径

  用户类应用程序数据目录路径将用于查找文件。
- 文件的全限定路径名

  该目录必须存在于路径中，否则该方法将失败并出现异常。文件不需要存在于路径中。

返回值

无人

示例

```
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

此方法会返回配置为**打印到磁盘 - 附加**方式的文件。此文件称为**打印到磁盘 - 附加**文件。此方法可能不会每次都返回相同的字符串。

该字符串仅在对象的生命周期内有效。您必须复制字符串，或者在每次需要时调用此方法。

原型

const char *GetPrtToDskAppendFile();

Parameters

无人

返回值

char *

有效值如下：

- 包含连接的**打印到磁盘 - 附加**文件的全限定路径名的以 null 结束的字符串。
- 如果未在连接中配置**打印到磁盘 - 附加**文件，则返回 NULL。

示例

```
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

此方法会将连接设置为**打印到磁盘 - 单独**方式，并为此方式设置相应的文件。

注：

1. 关联的连接必须处于 PDT 方式。
2. 要设置此文件的文件夹必须具有写入访问权。否则，此方法将失败并出现异常。
3. 文件名不能包含扩展名。如果包含扩展名，则该方法将失败并出现异常。

原型

void SetPrtToDskSeparate(const char *const FileName = NULL);

Parameters

char *FileName

包含**打印到磁盘 - 单独**文件的名称的以 null 结束的字符串。此参数是可选参数。
有效值如下：

- NULL

  使用连接中当前为此 PrintMode 配置的文件。如果尚未在连接中配置文件，则此方法将失败并出现异常。这是缺省值。
- 文件名，不带路径

  用户类应用程序数据目录路径将用于查找文件。
- 文件的全限定路径名

  该目录必须存在于路径中，否则该方法将失败并出现异常。文件不需要存在于路径中。

返回值

无人

示例

```
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

此方法会返回配置为**打印到磁盘 - 单独** 方式的文件。此文件称为**打印到磁盘 - 单独**文件。此方法可能不会每次都返回相同的字符串。

该字符串仅在对象的生命周期内有效。您必须复制字符串，或者在每次需要时调用此方法。

原型

const char *GetPrtToDskSeparateFile();

Parameters

无人

返回值

char *

有效值如下：

- 包含**打印到磁盘 - 单独**文件的全限定路径名的以 null 结束的字符串。
- 如果未在连接中配置**打印到磁盘 - 单独**文件，则返回 NULL。

示例

```
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

此方法会设置在 SpecificPrinter 方式下与在 Printer 参数中指定的打印机的连接。

原型

void SetSpecificPrinter(const char *const Printer);

Parameters

char *Printer

包含打印机名称和端口名的以 null 结束的字符串。如果打印机不存在，则此方法将失败并出现异常。
值必须采用以下格式：

```
<Printer name> on <Port Name>
```

例如：

- `HP LaserJet 4050 Series PCL 6 on LPT1`

返回值

无人

示例

```
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

此方法会在 WinDefaultPrinter 方式下设置连接，即建立连接以使用 Windows® 缺省打印机。如果未在机器中配置 Windows 缺省打印机，则此方法将失败并出现异常。

原型

void SetWinDefaultPrinter();

Parameters

无人

返回值

无人

示例

```
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

此方法会返回 NULL 或连接中配置的打印机名称。此方法可能不会每次都返回相同的字符串。

该字符串仅在对象的生命周期内有效。您必须复制字符串，或者在每次需要时调用此方法。

PrinterName 必须采用以下格式：

```
<Printer name> on <Port Name>
```

例如：

- `HP LaserJet 4050 Series PCL 6 on LPT1`

原型

const char *GetPrinterName();

Parameters

无人

返回值

char *

有效值如下：

- 如果连接的 PrintMode 是 SpecificPrinter，则为包含特定打印机名称的以 null 结束的字符串。
- 如果连接的 PrintMode 为 WinDefaultPrinter，则为包含 Windows 缺省打印机名称的以 null 结束的字符串。
- 如果连接中未配置打印机，或者如果连接的 PrintMode 是 PrtToDskAppend 或 PrtToDskSeparate，则为 NULL。

示例

```
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

此方法会设置或重置选项，以在打印前显示“打印机设置”对话框。

原型

void SetPromptDialog(BOOL bPrompt=TRUE);

Parameters

BOOL bPrompt

此参数是可选参数。有效值如下：

- TRUE：在打印前显示“打印机设置”对话框。这是缺省值。
- FALSE：在打印前不显示“打印机设置”对话框。

返回值

无人

示例

```
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

此方法会检查是否在打印前显示“打印机设置”对话框。

原型

BOOL IsPromptDialogEnabled();

Parameters

无人

返回值

BOOL

有效值如下：

- 如果打印前显示“打印机设置”对话框，则返回 TRUE。
- 如果打印前未显示“打印机设置”对话框，则返回 FALSE。

示例

```
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. Host Access Class Library Automation Objects

主机访问类库自动化对象允许 Z and I Emulator for Windows 产品支持 Microsoft® 基于 COM 的自动化技术（以前称为 OLE 自动化）。ECL 自动化对象是一系列允许自动化控制器的自动化服务器，例如 Microsoft® Visual Basic®，以便以编程方式访问 Z and I Emulator for Windows 的数据和功能。

例如，将键发送到 Z and I Emulator for Windows 表示空间。这可以通过在 Z and I Emulator for Windows 窗口中手动输入键来实现，但也可以通过相应的 Z and I Emulator for Windows 自动化服务器（在本例中为 autECLPS）实现自动化。使用 Visual Basic®，可以创建 autECLPS 对象，然后使用要放置在表示空间中的字符串调用该对象中的 SendKeys 方法。

换言之，支持控制自动化协议的应用程序（自动化控制器）可以控制一些 Z and I Emulator for Windows 操作（自动化服务器）。Z and I Emulator for Windows 支持 Visual Basic® 脚本，该脚本使用 ECL 自动化对象。请参阅 Z and I Emulator for Windows 宏/脚本支持以获取更多详细信息。

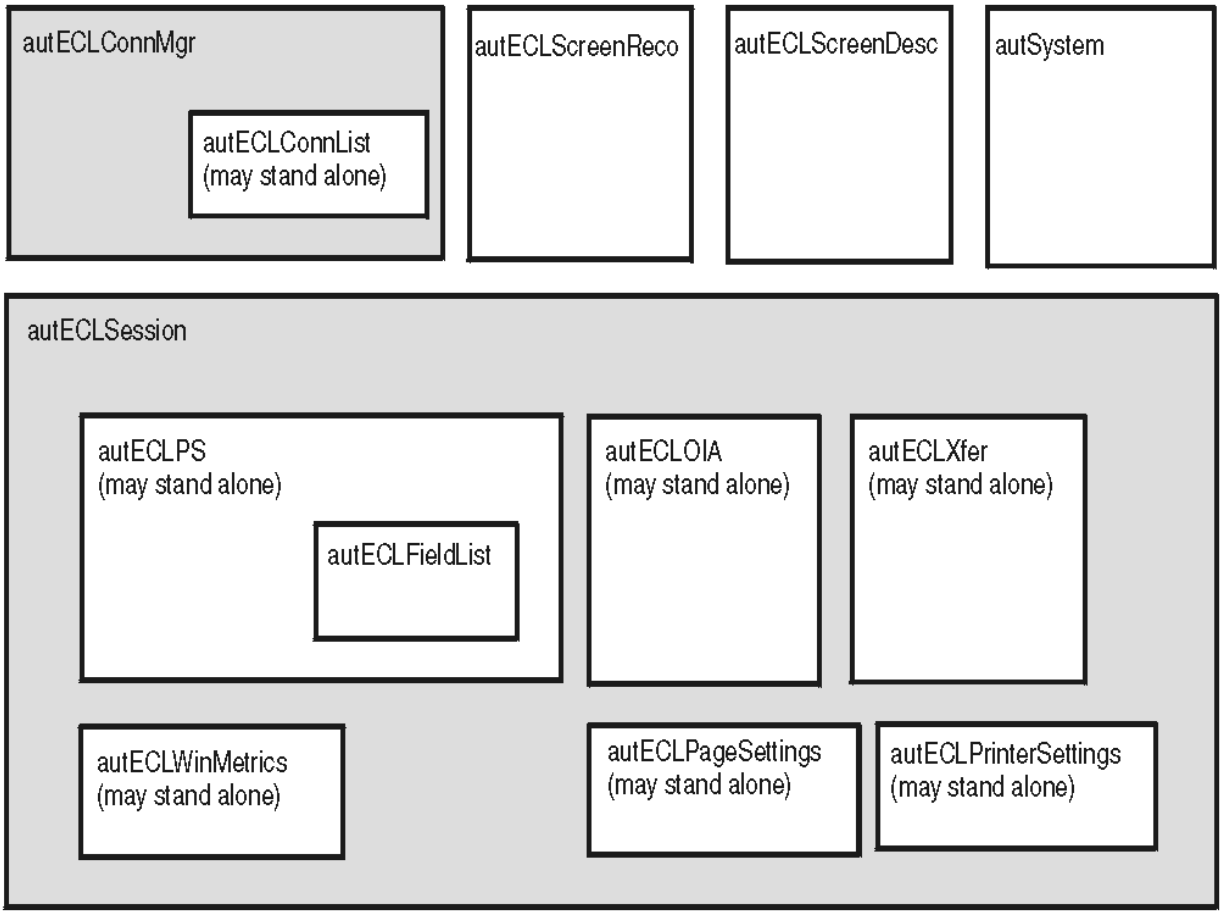
Z and I Emulator for Windows 提供多个自动化服务器来实现这一目标。这些服务器是作为现实世界的直观对象实施的，具有控制 Z and I Emulator for Windows 可操作性的方法和属性。每个对象都以 autECL 开头，用于自动化主机访问类库。对象如下所示：

- 第 autECLConnList 类 (on page 244) 页上的 autECLConnList（即连接列表）包含给定系统的 Z and I Emulator for Windows 连接列表。它包含在 autECLConnMgr 中，但可以独立于 autECLConnMgr 创建。
- 第 autECLConnMgr 类 (on page 251) 页上的 autECLConnMgr（即连接管理器）提供了管理给定系统的 Z and I Emulator for Windows 连接的方法和属性。此上下文中的连接是 Z and I Emulator for Windows 窗口。
- 第 autECLFieldList 类 (on page 257) 页上的 autECLFieldList（即字段列表）会对仿真器表示空间中的字段执行操作。
- 第 autECLOIA 类 (on page 268) 页上的 autECLOIA（即操作员信息区域）提供了查询和操控操作员信息区域的方法和属性。这包含在 autECLSession 中，但可以独立于 autECLSession 创建。
- 第 autECLPS 类 (on page 285) 页上的 autECLPS（即表示空间）提供了查询和操控相关 Z and I Emulator for Windows 连接的表示空间的方法和属性。这包含表示空间中所有字段的列表。它包含在 autECLSession 中，但可以独立于 autECLSession 创建。
- 第 autECLScreenDesc 类 (on page 324) 页上的 autECLScreenDesc（即屏幕描述）提供了描述屏幕的方法和属性。这可用于等待 autECLPS 对象或 autECLScreenReco 对象上的屏幕。
- 第 autECLScreenReco 类 (on page 332) 页上的 autECLScreenReco（即屏幕识别）提供了 HACL 屏幕识别系统的引擎。
- 第 autECLSession 类 (on page 338) 页上的 autECLSession（即会话）提供了与会话相关的常规功能和信息。为方便起见，它包含 autECLPS、autECLOIA、autECLXfer、autECLWinMetrics、autECLPageSettings 和 autECLPrinterSettings 对象。
- 第 autECLWinMetrics 类 (on page 351) 页上的 autECLWinMetrics（即窗口度量）提供了查询与此对象关联的 Z and I Emulator for Windows 会话的窗口度量的方法。例如，使用此对象可最小化或最大化 Z and I Emulator for Windows 窗口。这包含在 autECLSession 中，但可以独立于 autECLSession 创建。
- 第 autECLXfer 类 (on page 366) 页上的 autECLXfer（即文件传输）提供了通过与此文件传输对象关联的 Z and I Emulator for Windows 连接在主机和工作站之间传输文件的方法和属性。这包含在 autECLSession 中，但可以独立于 autECLsession 创建。

- 第 autECLPageSettings 类 (on page 380) 页上的 autECLPageSettings（即页面设置）提供了查询和操控常用设置的方法和属性，如会话“页面设置”对话框的 CPI、LPI 和字体名称。这包含在 autECLSession 中，但可以独立于 autECLSession 创建。
- 第 autECLPrinterSettings 类 (on page 391) 页上的 autECLPrinterSettings（即打印机设置）提供了查询和操控设置的方法和属性，如会话“打印机设置”对话框的打印机和 PDT 方式。这包含在 autECLSession 中，但可以独立于 autECLSession 创建。

图 3: Host Access Class Library Automation Objects (on page 243) 是 autECL 对象的图形表示:

图 3. Host Access Class Library Automation Objects



本章详细介绍了每个对象的方法和属性，旨在涵盖自动化对象的所有潜在用户。因为使用对象的最常见方法是通过脚本应用程序（如 Visual Basic®），所以所有示例均使用 Visual Basic® 格式显示。

autSystem 类

autSystem 类提供两个实用程序函数，可能对某些编程语言很有用。请参阅 autSystem 类 (on page 379) 以获取更多信息。

autECLConnList 类

autECLConnList 包含有关所有已启动连接的信息。它在注册表中的名称是 ZIEWin.autECLConnList。

autECLConnList 对象包含有关与某一主机连接的一组信息集合。集合中的每个元素表示一个单个的连接（仿真器窗口）。此列表中的连接可能处于任何状态（例如，已停止或已断开连接）。所有已启动的连接都将显示在此列表中。列表元素包含连接的状态。

autECLConnList 对象提供当前连接的静态快照。列表并不随着连接的启动与停止而动态地进行更新。在构造 autECLConnList 对象时，将自动调用 Refresh 方法。如果在构造 autECLConnList 对象后立即使用该对象，则连接列表是最新的。但是，如果自构造起已过了一段时间，则应在访问其他方法之前在 autECLConnList 对象中调用 Refresh 方法，以确保具有最新数据。调用 Refresh 后，就可以开始浏览集合

属性

本节介绍了 autECLConnList 对象的属性。

类型	名称	属性
长	计数	只读

下表显示了对列表中的每项都有效的集合元素属性。

类型	名称	属性
字符串	姓名	只读
长	句柄	只读
字符串	ConnType	只读
长	CodePage	只读
布尔值	已启动	只读
布尔值	CommStarted	只读
布尔值	APIEnabled	只读
布尔值	就绪	只读

计数

这是最近一次调用 Refresh 方法时 autECLConnList 集合中存在的连接数。Count 属性是长整型数据类型且为只读。以下示例使用了 Count 属性。

```
Dim autECLConnList as Object
Dim Num as Long

Set autECLConnList = CreateObject("ZIEWin.autECLConnList")
```

```
autECLConnList.Refresh
Num = autECLConnList.Count
```

姓名

此集合元素属性是连接的连接名称字符串。Z and I Emulator for Windows 仅返回字符串中的短字符标识（A-Z 或 a-z）。给定名称只能有一个打开的 Z and I Emulator for Windows 连接。例如，一次只能有一个打开的连接“A”。Name 是字符串数据类型且为只读。以下示例使用 Name 集合元素属性。

```
Dim  Str as String
Dim autECLConnList as Object
Dim Num as Long

Set autECLConnList = CreateObject("ZIEWin.autECLConnList")

autECLConnList.Refresh
Str = autECLConnList(1).Name
```

句柄

此集合元素属性是连接的句柄。给定句柄只能有一个打开的 Z and I Emulator for Windows 连接。Handle 是长整型数据类型且为只读。以下示例使用了 Handle 属性。

```
Dim autECLConnList as Object
Dim Hand as Long

Set autECLConnList = CreateObject("ZIEWin.autECLConnList")

autECLConnList.Refresh
Hand = autECLConnList(1).Handle
```

ConnType

此集合元素属性是连接类型。这种类型可能会随着时间的推移而更改。ConnType 是字符串数据类型且为只读。以下示例展示了 ConnType 属性。

```
Dim  Type as String
Dim autECLConnList as Object

Set autECLConnList = CreateObject("ZIEWin.autECLConnList")

autECLConnList.Refresh
Type = autECLConnList(1).ConnType
```

ConnType 属性的连接类型包括：

返回的字符串	意义
DISP3270	3270 显示
DISP5250	5250 显示
PRNT3270	3270 打印机

PRNT5250	5250 打印机
ASCII	VT 仿真

CodePage

此集合元素属性是连接的代码页。此代码页可能会随着时间的推移而更改。CodePage 是长整型数据类型且为只读。以下示例展示了 CodePage 属性。

```
Dim  CodePage as Long
Dim autECLConnList as Object

Set autECLConnList = CreateObject("ZIEWin.autECLConnList")

autECLConnList.Refresh
CodePage = autECLConnList(1).CodePage
```

已开始

此集合元素属性会指示是否已启动仿真器窗口。如果窗口已打开，则值为 True；否则为 False。Started 是布尔数据类型且为只读。以下示例展示了 Started 属性。

```
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

此集合元素属性会指示与主机的连接的状态。如果主机已连接，则值为 True；否则为 False。CommStarted 是布尔数据类型且为只读。以下示例展示了 CommStarted 属性。

```
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

此集合元素属性指示仿真器是否已启用 API。连接可能已启用或已禁用 API，具体取决于其 API 设置的状态（在 Z and I Emulator for Windows 窗口中，选择**文件 → API 设置**）。如果仿真器已启用，则值为 True；否则为 False。APIEnabled 是布尔数据类型且为只读。以下示例展示了 APIEnabled 属性。

```
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
```

Ready

此集合元素属性指示仿真器窗口是否已启动、已启用 API 和已连接。此属性会检查是否有所有三个属性。如果仿真器就绪，则值为 True；否则为 False。Ready 是布尔数据类型且为只读。以下示例展示了 Ready 属性。

```
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 方法

下节介绍了对 autECLConnList 对象有效的方法。

void Refresh()

Object FindConnectionByHandle(Long Hand)

Object FindConnectionByName(String Name)

集合元素方法

以下集合元素方法对列表中的每项都有效。

void StartCommunication()
void StopCommunication()

Refresh

Refresh 方法会获取所有已启动连接的快照。

注： 应在访问 autECLConnList 集合之前调用此方法，以确保具有当前数据。

原型

void Refresh()

Parameters

无人

返回值

无人

示例

以下示例展示了如何使用 Refresh 方法获取所有已启动连接的快照。

```
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

此方法会在 autECLConnList 对象中查找在 **Handle** 参数中传递的句柄的元素。此方法通常用于查看系统中某一给定会话是否有效。

原型

Object FindConnectionByHandle(Long Hand)

Parameters

Long Hand

要在列表中搜索的句柄。

返回值

对象

集合元素分派对象。

示例

以下示例展示了如何通过连接句柄查找元素。

```
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

此方法会在 autECLConnList 对象中查找在 **Name** 参数中传递的名称的元素。此方法通常用于查看系统中某一给定会话是否有效。

原型

Object FindConnectionByName(String Name)

Parameters

String Name

要在列表中搜索的名称。

返回值

对象

集合元素分派对象。

示例

以下示例展示了如何通过连接名称在 autECLConnList 对象中查找元素。

```
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 集合元素方法会将 ZIEWin 仿真器连接到主机数据流。这与进入 ZIEWin 仿真器**通信**菜单并选择**连接**具有相同的效果。

原型

void StartCommunication()

Parameters

无人

返回值

无人

示例

以下示例展示了如何将 ZIEWin 仿真器会话连接到主机。

```
Dim autECLConnList as Object

Set autECLConnList = CreateObject("ZIEWin.autECLConnList")

'Start the first session
autECLConnList.Refresh
autECLConnList(1).StartCommunication()
```

StopCommunication

StopCommunication 集合元素方法会断开 ZIEWin 仿真器与主机数据流的连接。这与进入 ZIEWin 仿真器**通信**菜单并选择**断开连接**具有相同的效果。

原型

void StopCommunication()

Parameters

无人

返回值

无人

示例

以下示例展示了如何断开 ZIEWin 仿真器会话与主机的连接。

```
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 类

autECLConnMgr 会管理给定机器上的所有 Z and I Emulator for Windows 连接。它包含与连接管理相关的方法，如启动和停止连接。它还会创建 autECLConnList 对象，以枚举系统上所有已知连接的列表（请参阅 autECLConnList 类 (on page 244)）。它在注册表中的名称是 ZIEWin.autECLConnMgr。

属性

本节介绍了 autECLConnMgr 对象的属性。

Type	名称	属性
autECLConnList 对象	autECLConnList	只读

autECLConnList

autECLConnMgr 对象包含 autECLConnList 对象。请参阅 autECLConnList 类 (on page 244) 以获取有关其方法和属性的详细信息。此属性的值为 autECLConnList，这是 autECLConnList 分派对象。下列示例展示了此属性。

```
Dim Mgr as Object
Dim Num as Long

Set Mgr = CreateObject("ZIEWin.autECLConnMgr ")

Mgr.autECLConnList.Refresh
Num = Mgr.autECLConnList.Count
```

autECLConnMgr 方法

下节介绍了对 autECLConnMgr 有效的方法。

void RegisterStartEvent()
void UnregisterStartEvent()
void StartConnection(String ConfigParms)
void StopConnection(Variant Connection, [optional] String StopParms)

RegisterStartEvent

此方法会注册 autECLConnMgr 对象，以接收会话中启动事件的通知。

原型

void RegisterStartEvent()

Parameters

无人

返回值

无人

示例

有关示例，请参阅事件处理示例 (on page 256)。

UnregisterStartEvent

结束开始事件处理

原型

void UnregisterStartEvent()

Parameters

无人

返回值

无人

示例

有关示例，请参阅事件处理示例 (on page 256)。

StartConnection

此成员函数会启动新的 Z and I Emulator for Windows 仿真器窗口。ConfigParms 字符串包含 使用注意事项 (on page 253) 下所述的连接配置信息。

原型

void StartConnection(String ConfigParms)

Parameters

String ConfigParms

配置字符串。

返回值

无人

使用注意事项

配置字符串特定于实现。autECL 对象的不同实现可能要求在配置字符串中具有不同的格式或信息。在从此调用返回时将启动新仿真器，但它可能连接到主机，也可能不连接到主机。

对于 Z and I Emulator for Windows，配置字符串有以下格式：

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

可选参数用方括号 [] 括起。参数间至少用一个空格分隔。参数可以是大写、小写或大小写混合，并且可以按任意顺序显示。每个参数的含义如下所示：

- PROFILE=<filename>：命令 Z and I Emulator for Windows 工作站概要文件（.WS 文件），其中包含配置信息。此参数不是可选的；必须提供概要文件名称。如果文件名包含空格，则必须用单引号将名称引起来。<filename> 值可以是不带扩展名的概要文件名，带 .WS 扩展名的概要文件名，或全限定概要文件名路径。
- CONNNAME=<c> 指定新连接的短标识。此值必须是单个字母字符（A-Z 或 a-z）。如果未指定此值，则将自动分配下一个可用的连接标识。
- WINSTATE=<MAX|MIN|RESTORE|HIDE> 指定仿真器窗口的初始状态。如果未指定此参数，则缺省值为 RESTORE。

示例

以下示例展示了如何启动新 Z and I Emulator for Windows 仿真器窗口。

```
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 方法会停止（终止）连接句柄标识的仿真器窗口。请参阅 使用注意事项 (on page 254) 以获取 StopParms 字符串的内容。

原型

void StopConnection(Variant Connection, [optional] String StopParms)

Parameters

Variant Connection

连接名称或句柄。此变体的合法类型包括 short、long、BSTR、short by reference、long by reference 和 BSTR by reference。

String StopParms

停止参数字符串。有关字符串的格式，请参阅用法说明。此参数是可选参数。

返回值

无人

使用注意事项

停止参数字符串特定于实现。autECL 对象的不同实现可能需要参数字符串的不同格式和内容。对于 Z and I Emulator for Windows，字符串采用下列格式：

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

可选参数用方括号 [] 括起。参数间至少用一个空格分隔。参数可以是大写、小写或大小写混合，并且可以按任意顺序显示。每个参数的含义如下所示：

- SAVEPROFILE=<YES|NO|DEFAULT> 控制将当前配置保存回工作站概要文件（.WS 文件）。这会导致使用您可能所做的任何配置更改来更新概要文件。如果指定 NO，将停止连接且不更新概要文件。如果指定 YES，将停止连接并使用当前（可能已更改的）配置更新概要文件。如果指定 DEFAULT，则更新选项由**文件 -> 退出时保存**仿真器菜单选项控制。如果未指定该参数，则使用 DEFAULT。

示例

以下示例展示了如何停止连接句柄标识的仿真器窗口。

```
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 事件

以下事件对 autECLConnMgr 有效：

void NotifyStartEvent(By Val Handle As Variant, By Val Started As Boolean)
NotifyStartError(By Val ConnHandle As Variant)
void NotifyStartStop(Long Reason)

NotifyStartEvent

会话已启动或已停止。

原型

void NotifyStartEvent(By Val Handle As Variant, By Val Started As Boolean)

注： Visual Basic 将正确创建此子例程。

Parameters

By Val Handle As Variant

已启动或已停止的会话的句柄。

By Val Started As Boolean

如果会话已启动，则为 True；否则为 False。

示例

有关示例，请参阅事件处理示例 (on page 256)。

NotifyStartError

事件处理出错时会发生此事件。

原型

NotifyStartError(By Val ConnHandle As Variant)

注： Visual Basic 将正确创建此子例程。

Parameters

无人

示例

有关示例，请参阅事件处理示例 (on page 256)。

NotifyStartStop

事件处理停止时会发生此事件。

原型

void NotifyStartStop(Long Reason)

Parameters

Long Reason

停止的原因码。目前，这将始终为 0。

事件处理示例

以下是有关如何实现启动事件的简短示例：

```
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 类

autECLFieldList 会对仿真器表示空间中的字段执行操作。此对象不能独立存在。它包含在 autECLPS 中，并且只能通过 autECLPS 对象访问。autECLPS 可以独立存在，也可以包含在 autECLSession 中。

autECLFieldList 包含在某一给定表示空间上的所有字段的集合。该集合的每个元素都包含集合元素属性 (on page 258)中显示的元素。

autECLFieldList 对象提供调用 Refresh 方法时包含的表示空间的静态快照。

注： 在访问元素之前，应先调用 autECLFieldList 对象中的 Refresh 方法，以确保具有当前字段数据。调用 Refresh 后，可以开始浏览集合。

属性

本节介绍 autECLFieldList 对象的属性和集合元素属性。

类型	名称	属性
长	计数	只读

以下属性是集合元素属性，对列表中的每项都有效。

类型	名称	属性
长	StartRow	只读
长	StartCol	只读
长	EndRow	只读
长	EndCol	只读
长	Length	只读
布尔值	Modified	只读
布尔值	受保护	只读
布尔值	Numeric	只读
布尔值	HighIntensity	只读
布尔值	PenDetectable	只读
布尔值	Display	只读

计数

此属性是最近一次调用 Refresh 方法时 autECLFieldList 集中存在的字段数。Count 是长整型数据类型且为只读。下列示例展示了此属性。

```
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

此集合元素属性是 autECLFieldList 集合的给定字段中第一个字符的行位置。StartRow 是长整型数据类型且为只读。下列示例展示了此属性。

```
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

此集合元素属性是 autECLFieldList 集合的给定字段中第一个字符的列位置。StartCol 是长整型数据类型且为只读。下列示例展示了此属性。

```
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

此集合元素属性是 autECLFieldList 集合的给定字段中最后一个字符的行位置。EndRow 是长整型数据类型且为只读。下列示例展示了此属性。

```
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

此集合元素属性是 autECLFieldList 集合的给定字段中最后一个字符的列位置。EndCol 是长整型数据类型且为只读。下列示例展示了此属性。

```
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
```

Length

集合元素属性是 autECLFieldList 集中某一给定字段的长度。Length 是长整型数据类型且为只读。下列示例展示了此属性。

```
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
```

Modified

此集合元素属性指示 autECLFieldList 集合中的给定字段是否具有已修改属性。Modified 是布尔数据类型且为只读。下列示例展示了此属性。

```
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
```

受保护

此集合元素属性指示 autECLFieldList 集合中的给定字段是否具有受保护属性。Protected 是布尔数据类型且为只读。下列示例展示了此属性。

```
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
```

Numeric

此集合元素属性指示 autECLFieldList 集合中的给定字段是否具有仅输入数字属性。Numeric 是布尔数据类型且为只读。下列示例展示了此属性。

```
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

此集合元素属性指示 autECLFieldList 集合中的给定字段是否具有高亮度属性。HighIntensity 是布尔数据类型且为只读。下列示例展示了此属性。

```
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

此集合元素属性指示 autECLFieldList 集合中的给定字段是否具有光笔可检测属性。PenDetectable 是布尔数据类型且为只读。下列示例展示了此属性。

```
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
```

Display

此集合元素属性指示 autECLFieldList 集合中的给定字段是否具有显示属性。Display 是布尔数据类型且为只读。下列示例展示了此属性。

```
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 方法

下节介绍了对 autECLFieldList 对象有效的方法。

void Refresh()
Object FindFieldByRowCol(Long Row, Long Col)
Object FindFieldByText(String text, [optional] Long Direction, [optional] Long StartRow, [optional] Long StartCol)

集合元素方法

以下集合元素方法对列表中的每项都有效。

String GetText()
void SetText(String Text)

Refresh

Refresh 方法会获取所有字段的快照。

注： 在访问字段集合之前，应先调用 Refresh 方法，以确保具有当前字段数据。

原型

void Refresh()

Parameters

无人

返回值

无人

示例

以下示例展示了如何获取给定表示空间的所有字段的快照。

```
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

此方法会在 autECLFieldList 对象中搜索包含给定行和列坐标的字段。返回的值是 autECLFieldList 集合中的集合元素对象。

原型

Object FindFieldByRowCol(Long Row, Long Col)

Parameters

Long Row

要搜索的字段行。

Long Col

要搜索的字段列。

返回值

对象

autECLFieldList 集合项的分派对象。

示例

以下示例展示了如何在 autECLFieldList 对象中搜索包含给定行和列坐标的字段。

```
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

此方法会在 autECLFieldList 对象中搜索包含以**文本**形式进行传递的字符串的字段。返回的值是 autECLFieldList 集合中的集合元素对象。

原型

Object FindFieldByText(String Text, [optional] Long Direction, [optional] Long StartRow, [optional] Long StartCol)

Parameters

字符串文本

要搜索的文本字符串。

Long StartRow

表示空间中要开始搜索的行位置。

Long StartCol

表示空间中要开始搜索的列位置。

Long Direction

搜索方向。值包括表示向前搜索的 **1** 和表示向后搜索的 **2**

返回值

对象

autECLFieldList 集合项的分派对象。

示例

以下示例展示了如何在 autECLFieldList 对象中搜索包含作为文本传递的字符串的字段。

```
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

集合元素方法 GetText 会检索 autECLFieldList 项中给定字段的字符。

原型

String GetText()

Parameters

无人

返回值

字符串

字段文本。

示例

以下示例展示了如何使用 GetText 方法。

```
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

此方法会使用作为文本传递的字符串填充 autECLFieldList 项中的给定字段。如果文本超过字段的长度，文本将被截断。

原型

void SetText(String Text)

Parameters

String text

要在字段中设置的字符串

返回值

无人

示例

以下示例展示了如何使用作为文本传递的字符串填充 autECLFieldList 项中的字段。

```
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 类

autECLOIA 对象会从操作员信息区域检索状态。它在注册表中的名称是 ZIEWin.autECLOIA。

必须在初始时为创建的对象设置连接。使用 SetConnectionByName 或 SetConnectionByHandle 来初始化对象。只能对连接进行一次设置。在设置连接之后，任何再调用设置连接方法都将引发异常。如果未设置连接，并尝试访问属性或方法，也将引发异常。

注： autECLSession 对象中的 autECLOIA 对象由 autECLSession 对象设置。

以下示例展示了如何在 Visual Basic 中创建和设置 autECLOIA 对象。

```
DIM  autECLOIA as Object

Set autECLOIA = CreateObject("ZIEWin.autECLOIA")
autECLOIA.SetConnectionByName("A")
```

属性

本节介绍了 autECLOIA 对象的属性。

类型	名称	属性
布尔值	字母数字	只读
布尔值	APL	只读
布尔值	UpperShift	只读
布尔值	Numeric	只读
布尔值	CapsLock	只读
布尔值	InsertMode	只读
布尔值	CommErrorReminder	只读
布尔值	MessageWaiting	只读
长	InputInhibited	只读
字符串	姓名	只读
长	句柄	只读
字符串	ConnType	只读
长	CodePage	只读
布尔值	已启动	只读
布尔值	CommStarted	只读
布尔值	APIEnabled	只读
布尔值	就绪	只读
布尔值	NumLock	只读

字母数字

此属性会查询操作员信息区域，以确定光标位置处的字段是否为字母数字。Alphanumeric 是布尔数据类型且为只读。下列示例展示了此属性。

```
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

此属性会查询操作员信息区域，以确定键盘是否为 APL 方式。APL 是布尔数据类型且为只读。下列示例展示了此属性。

```
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...
```

Katakana

此属性会查询操作员信息区域，以确定是否启用片假名字符。Katakana 是布尔数据类型且为只读。下列示例展示了此属性。

```
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

此属性会查询操作员信息区域，以确定是否启用平假名字符。Hiragana 是布尔数据类型且为只读。下列示例展示了此属性。

```
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

此属性会查询操作员信息区域，以确定键盘是否为 UpperShift 方式。UpperShift 是布尔数据类型且为只读。下列示例展示了此属性。

```
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...
```

Numeric

此属性会查询操作员信息区域，以确定光标位置处的字段是否为数字。Numeric 是布尔数据类型且为只读。下列示例展示了此属性。

```
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

此属性会查询操作员信息区域，以确定键盘 CapsLock 键是否已打开。CapsLock 是布尔数据类型且为只读。下列示例展示了此属性。

```
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

此属性会查询操作员信息区域，以确定键盘是否为插入方式。InsertMode 是布尔数据类型且为只读。下列示例展示了此属性。

```
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

此属性会查询操作员信息区域，以确定是否存在通信错误提醒条件。CommErrorReminder 是布尔数据类型且为只读。下列示例展示了此属性。

```
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

此属性会查询操作员信息区域，以确定消息等待指示符是否已打开。这仅发生在 5250 会话中。MessageWaiting 是布尔数据类型且为只读。下列示例展示了此属性。

```
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

此属性会查询操作员信息区域，以确定键盘输入是否被禁止。InputInhibited 是长整型数据类型且为只读。下表显示了 InputInhibited 的有效值。

名称	价值
未禁止	0
系统等待	1
通信检查	2
程序检查	3
机器检查	4
其他禁止	5

下列示例展示了此属性。

```
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...
```

姓名

此属性是设置了 autECLOIA 的连接的连接名称字符串。Z and I Emulator for Windows 仅返回字符串中的短字符标识（A-Z 或 a-z）。给定名称只能有一个打开的 Z and I Emulator for Windows 连接。例如，一次只能有一个打开的连接“A”。Name 是字符串数据类型且为只读。下列示例展示了此属性。

```
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
```

句柄

这是设置了 autECLOIA 对象的连接的句柄。给定句柄只能有一个打开的 Z and I Emulator for Windows 连接。例如，一次只能有一个打开的连接“A”。Handle 是长整型数据类型且为只读。下列示例展示了此属性。

```
DIM  Obj as Object
Set Obj = CreateObject("ZIEWin.autECLOIA")

' Initialize the connection
Obj.SetConnectionByName("A")

' Save the handle
Hand = Obj.Handle
```

ConnType

这是设置了 autECLOIA 的连接类型。这种类型可能会随着时间的推移而更改。ConnType 是字符串数据类型且为只读。下列示例展示了此属性。

```
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 属性的连接类型包括：

返回的字符串	意义
DISP3270	3270 显示
DISP5250	5250 显示
PRNT3270	3270 打印机

返回的字符串	意义
PRNT5250	5250 打印机
ASCII	VT 仿真

CodePage

这是设置了 autECLOIA 的连接的代码页。此代码页可能会随着时间的推移而更改。CodePage 是长整型数据类型且为只读。下列示例展示了此属性。

```
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
```

已开始

此项会指示是否已启动仿真器窗口。如果窗口已打开，则值为 True；否则为 False。Started 是布尔数据类型且为只读。下列示例展示了此属性。

```
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

此项会指示与主机的连接的状态。如果主机已连接，则值为 True；否则为 False。CommStarted 是布尔数据类型且为只读。下列示例展示了此属性。

```
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

此项会指示仿真器是否已启用 API。连接可能已启用或已禁用 API，具体取决于其 API 设置的状态（在 Z and I Emulator for Windows 窗口中，选择**文件 → API 设置**）。如果仿真器已启用，则值为 True；否则为 False。APIEnabled 是布尔数据类型且为只读。下列示例展示了此属性。

```
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
```

Ready

此项会指示仿真器窗口是否已启动、已启用 API 和已连接。此属性会检查是否有所有三个属性。如果仿真器就绪，则值为 True；否则为 False。Ready 是布尔数据类型且为只读。下列示例展示了此属性。

```
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

此属性会查询操作员信息区域，以确定键盘 NumLock 键是否已打开。NumLock 是布尔数据类型且为只读。下列示例展示了此属性。

```
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 方法

下节介绍了对 autECLOIA 有效的方法。

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

此方法会注册一个对象，以接收所有 OIA 事件的通知。

原型

void RegisterOIAEvent()

Parameters

无人

返回值

无人

示例

有关示例，请参阅事件处理示例 (on page 284)。

UnregisterOIAEvent

结束 OIA 事件处理。

原型

void UnregisterOIAEvent()

Parameters

无人

返回值

无人

示例

有关示例，请参阅事件处理示例 (on page 284)。

SetConnectionByName

SetConnectionByName 方法使用连接名称为新创建的 autECLOIA 对象设置连接。在 Z and I Emulator for Windows 中，此连接名称是短连接标识（字符 A-Z 或 a-z）。给定名称只能有一个打开的 Z and I Emulator for Windows 连接。例如，一次只能有一个打开的连接“A”。

注： 如果在 autECLSession 中使用 autECLOIA 对象，请不要调用此项。

原型

void SetConnectionByName(String Name)

Parameters

String Name

连接的单字符字符串短名称（A-Z 或 a-z）。

返回值

无人

示例

以下示例展示了如何使用连接名称为新创建的 autECLOIA 对象设置连接。

```
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

SetConnectionByHandle 方法使用连接句柄为新创建的 autECLOIA 对象设置连接。在 Z and I Emulator for Windows 中，此连接句柄是长整型值。给定句柄只能有一个打开的 Z and I Emulator for Windows 连接。例如，一次只能有一个打开的连接“A”。

注： 如果在 autECLSession 中使用 autECLOIA 对象，请不要调用此项。

```
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
```

原型

void SetConnectionByHandle(Long Handle)

Parameters

Long Handle

要为对象设置的连接的长整型值。

返回值

无人

示例

以下示例展示了如何使用连接句柄为新创建的 autELCOIA 对象设置连接。

StartCommunication

StartCommunication 集合元素方法会将 ZIEWin 仿真器连接到主机数据流。这与进入 ZIEWin 仿真器**通信**菜单并选择**连接**具有相同的效果。

原型

void StartCommunication()

Parameters

无人

返回值

无人

示例

无人

```
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 集合元素方法会断开 ZIEWin 仿真器与主机数据流的连接。这与进入 ZIEWin 仿真器**通信**菜单并选择**断开连接**具有相同的效果。

原型

void StopCommunication()

Parameters

无人

返回值

无人

示例

以下示例展示了如何将 ZIEWin 仿真器会话连接到主机。

```
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 方法会等至与 autECLOIA 对象关联的连接的 OIA 指示连接可以接受键盘输入。

原型

Boolean WaitForInputReady([optional] Variant TimeOut)

Parameters

Variant TimeOut

以毫秒为单位的最长时间长度，此参数为可选。缺省值是无限长。

返回值

如果符合条件，方法返回 True；如果超过了 Timeout 值，则返回 False。

示例

```
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 方法会等至与 autECLOIA 对象相关联的连接的 OIA 指示该连接已连接到主机系统。

原型

Boolean WaitForSystemAvailable([optional] Variant TimeOut)

Parameters

Variant TimeOut

以毫秒为单位的最长时间长度，此参数为可选。缺省值是无限长。

返回值

如果符合条件，方法返回 True；如果超过了 Timeout 值，则返回 False。

示例

```
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

当与 autECLOIA 对象关联的连接的 OIA 表示正在使用应用程序时，WaitForAppAvailable 方法等待。

原型

Boolean WaitForAppAvailable([optional] Variant TimeOut)

Parameters

Variant TimeOut

以毫秒为单位的最长时间长度，此参数为可选。缺省值是无限长。

返回值

如果符合条件，方法返回 True；如果超过了 Timeout 值，则返回 False。

示例

```
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 方法等待与 autECLOIA 对象相关联的连接所指定的 OIA 位置发生变化。

原型

Boolean WaitForTransition([optional] Variant Index, [optional] Variant timeout)

Parameters

Variant Index

要监控的 OIA 的 1 字节十六进制位置。此参数是可选参数。缺省值为 3。

Variant TimeOut

以毫秒为单位的最长时间长度，此参数为可选。缺省值是无限长。

返回值

如果符合条件，方法返回 True；如果超过了 Timeout 值，则返回 False。

示例

```
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

取消任何当前活动的 wait 方法。

原型

void CancelWaits()

Parameters

无人

返回值

无人

autECLOIA 事件

以下事件对 autECLOIA 有效：

void NotifyOIAEvent()
void NotifyOIAError()
void NotifyOIAStop(Long Reason)

NotifyOIAEvent

给定的 OIA 已发生。

原型

void NotifyOIAEvent()

Parameters

无人

示例

有关示例，请参阅事件处理示例 (on page 284)。

NotifyOIAError

OIA 出错时会发生此事件。

原型

void NotifyOIAError()

Parameters

无人

示例

有关示例，请参阅事件处理示例 (on page 284)。

NotifyOIAStop

事件处理停止时会发生此事件。

原型

void NotifyOIAStop(Long Reason)

Parameters

Long Reason

停止的长原因码。目前，这将始终为 0。

事件处理示例

以下是有关如何实现 OIA 事件的简短示例

```
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 类

autECLPS 会对表示空间执行操作。它在注册表中的名称是 ZIEWin.autECLPS。

必须在初始时为创建的对象设置连接。使用 SetConnectionByName 或 SetConnectionByHandle 来初始化对象。只能对连接进行一次设置。在设置连接之后，任何再调用 SetConnection 方法都将引发异常。如果未设置连接，并尝试访问属性或方法，也将引发异常。

注:

1. 在表示空间中，第一行坐标为行 1，第一列坐标为列 1。因此，左上位置的坐标为行 1、列1。
2. autECLSession 对象中的 autECLPS 对象由 autECLSession 对象设置。

以下示例展示了如何在 Visual Basic 中创建和设置 autECLPS 对象。

```
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
```

属性

本节介绍了 autECLPS 对象的属性。

类型	名称	属性
对象	autECLFieldList	只读
长	NumRows	只读
长	NumCols	只读
长	CursorPosRow	只读
长	CursorPosCol	只读
字符串	姓名	只读
长	句柄	只读
字符串	ConnType	只读
长	CodePage	只读
布尔值	已启动	只读
布尔值	CommStarted	只读
布尔值	APIEnabled	只读
布尔值	就绪	只读

autECLFieldList

这是与 autECLPS 对象关联的连接的字段集合对象。请参阅 autECLFieldList 类 (on page 257) 以获取详细信息。下列示例展示了此对象。

```
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

这是与 autECLPS 对象关联的连接的表示空间中的行数。NumRows 是长整型数据类型且为只读。下列示例展示了此属性。

```
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

这是与 autECLPS 对象关联的连接的表示空间中的列数。NumCols 是长整型数据类型且为只读。下列示例展示了此属性。

```
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

这是游标在与 autECLPS 对象关联的连接的表示空间中的当前行位置。CursorPosRow 是长整型数据类型且为只读。下列示例展示了此属性。

```
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

这是游标在与 autECLPS 对象关联的连接的表示空间中的当前列位置。CursorPosCol 是长整型数据类型且为只读。下列示例展示了此属性。

```
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
```

姓名

这是设置了 autECLPS 的连接的连接名称字符串。Z and I Emulator for Windows 仅返回字符串中的短字符标识（A-Z 或 a-z）。给定名称只能有一个打开的 Z and I Emulator for Windows 连接。例如，一次只能有一个打开的连接“A”。Name 是字符串数据类型且为只读。下列示例展示了此属性。

```
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
```

句柄

这是设置了 autECLPS 对象的连接的句柄。给定句柄只能有一个打开的 Z and I Emulator for Windows 连接。例如，一次只能有一个打开的连接“A”。Handle 是长整型数据类型且为只读。下列示例展示了此属性。

```
DIM  Obj as Object
Set Obj = CreateObject("ZIEWin.autECLPS")

' Initialize the connection
Obj.SetConnectionByName("A")

' Save the connection handle
Hand = Obj.Handle
```

ConnType

这是设置了 autECLPS 的连接类型。此连接类型可能会随着时间的推移而更改。ConnType 是字符串数据类型且为只读。下列示例展示了此属性。

```
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 属性的连接类型包括：

返回的字符串	意义
DISP3270	3270 显示
DISP5250	5250 显示
PRNT3270	3270 打印机

返回的字符串	意义
PRNT5250	5250 打印机
ASCII	VT 仿真

CodePage

这是设置了 autECLPS 的连接的代码页。此代码页可能会随着时间的推移而更改。CodePage 是长整型数据类型且为只读。下列示例展示了此属性。

```
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
```

已开始

此项会指示是否已启动连接仿真器窗口。如果窗口已打开，则值为 True；否则为 False。Started 是布尔数据类型且为只读。下列示例展示了此属性。

```
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

此项会指示与主机的连接的状态。如果主机已连接，则值为 True；否则为 False。CommStarted 是布尔数据类型且为只读。下列示例展示了此属性。

```
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

此项会指示仿真器是否已启用 API。连接可能已启用或已禁用 API，具体取决于其 API 设置的状态（在 Z and I Emulator for Windows 窗口中，选择**文件 → API 设置**）。如果 API 已启用，则值为 True；否则为 False。APIEnabled 是布尔数据类型且为只读。下列示例展示了此属性。

```
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
```

Ready

此项会指示仿真器窗口是否已启动、已启用 API 和已连接。此项会检查是否有所有三个属性。如果仿真器就绪，则值为 True；否则为 False。Ready 是布尔数据类型且为只读。下列示例展示了此属性。

```
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 方法

下节介绍了对 autECLPS 对象有效的方法。

```
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

此方法会注册 autECLPS 对象，以接收对所连接会话的 PS 所做的所有更改的通知。

原型

void RegisterPSEvent()

Parameters

无人

返回值

无人

示例

有关示例，请参阅事件处理示例 (on page 322)。

RegisterKeyEvent

开始击键事件处理。

原型

void RegisterKeyEvent()

Parameters

无人

返回值

无人

示例

有关示例，请参阅事件处理示例 (on page 322)。

RegisterCommEvent

此方法会注册一个对象，以接收所有通信链路连接/断开连接事件的通知。

原型

void RegisterCommEvent()

Parameters

无人

返回值

无人

示例

有关示例，请参阅事件处理示例 (on page 322)。

UnregisterPSEvent

结束 PS 事件处理。

原型

void UnregisterPSEvent()

Parameters

无人

返回值

无人

示例

有关示例，请参阅事件处理示例 (on page 322)。

UnregisterKeyEvent

结束击键事件处理。

原型

void UnregisterKeyEvent()

Parameters

无人

返回值

无人

示例

有关示例，请参阅事件处理示例 (on page 322)。

UnregisterCommEvent

结束通信链路事件处理。

原型

void UnregisterCommEvent()

Parameters

无人

返回值

无人

SetConnectionByName

此方法使用连接名称为新创建的 autECLPS 对象设置连接。在 Z and I Emulator for Windows 中，此连接名称是短标识（字符 A-Z 或 a-z）。给定名称只能有一个打开的 Z and I Emulator for Windows 连接。例如，一次只能有一个打开的连接“A”。

注： 如果在 autECLSession 中使用 autECLPS 对象，请不要调用此项。

原型

void SetConnectionByName(String Name)

Parameters

String Name

连接的单字符字符串短名称（A-Z 或 a-z）。

返回值

无人

示例

以下示例展示了如何使用连接名称为新创建的 autECLPS 对象设置连接。

```
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

此方法使用连接句柄为新创建的 autECLPS 对象设置连接。在 Z and I Emulator for Windows 中，此连接句柄是长整型值。给定句柄只能有一个打开的 Z and I Emulator for Windows 连接。例如，一次只能有一个打开的连接“A”。

注： 如果在 autECLSession 中使用 autECLPS 对象，请不要调用此项。

原型

void SetConnectionByHandle(Long Handle)

Parameters

Long Handle

要为对象设置的连接的长整型值。

返回值

无人

示例

以下示例展示了如何使用连接句柄为新创建的 autECLPS 对象设置连接。

```
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 会设置游标在与 autECLPS 对象相关联的连接的表表示空间中的位置。位置集在行/列单元中。

原型

void SetCursorPos(Long Row, Long Col)

Parameters

Long Row

游标在表示空间中的行位置。

Long Col

游标在表示空间中的列位置。

返回值

无人

示例

以下示例展示了如何设置游标在表示空间中的位置，用于与 autECLPS 对象关联的连接。

```
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 方法会将键字符串发送到表示空间，用于与 autECLPS 对象关联的连接。此方法允许您将助记符击键发送到表示空间。请参阅 Sendkeys 助记符关键字 (on page 411) 以获取这些击键的列表。

原型

void SendKeys(字符串文本，[可选]长行，[可选]长列)

Parameters

String text

要发送到表示空间的键字符串。

Long Row

要将键发送到表示空间的行位置。此参数是可选参数。缺省值是当前光标行位置。如果指定了行，则还必须指定列。

Long Col

要将键发送到表示空间的列位置。此参数是可选参数。缺省值是当前光标列位置。如果指定了列，则还必须指定行。

返回值

无

示例

以下示例演示如何使用 SendKeys 方法将密钥字符串发送到与 autECLPS 对象关联的连接的表示空间。

```
Dim autECLPSObj 作为对象 Dim autECLPSObj 作为对象 Set autECLPSObj = CreateObject("ZIEWin .autECLPS")
 设置 autECCLConnList = CreateObject("ZIEWin .autECLConnList") ' 初始化连接 autECLPSObj.Refresh
 autECLPSObj.SetConnectionByHandle(autECLConnList(1).Handle) autECLPSObj.SendKeys "HCL是一家非常酷的公司”，
 3, 1
```

SearchText

SearchText 方法会在与 autECLPS 对象关联的会话的表示空间中搜索文本的第一个实例。搜索区分大小写。如果找到文本，则该方法返回 TRUE 值。如果未找到文本，则返回 FALSE 值。如果使用可选的行和列参数，则还将返回**行**和**列**，指示找到的文本的位置。

原型

boolean SearchText(String text, [optional] Long Dir, [optional] Long Row, [optional] Long Col)

Parameters

String text

要搜索的字符串。

Long Dir

搜索方向。必须为表示向前搜索的 **1** 或表示向后搜索的 **2**。此参数是可选参数。缺省值是表示向前搜索的 1。

Long Row

在表示空间中要开始搜索的行位置。如果搜索成功，则返回所找到文本的行。此参数是可选参数。如果指定了行，则还必须指定列。

Long Col

在表示空间中要开始搜索的列位置。如果搜索成功，则返回所找到文本的列。此参数是可选参数。如果指定了列，则还必须指定行。

返回值

如果找到文本，则返回 TRUE；如果未找到文本，则返回 FALSE

示例

以下示例展示了如何在表示空间中搜索文本，用于与 autECLPS 对象关联的连接。

```
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 方法会从与 autECLPS 对象相关联的连接的表示空间中检索字符。

原型

String GetText([optional] Long Row, [optional] Long Col, [optional] Long LenToGet)

Parameters

Long Row

在表示空间中要开始检索的行位置。此参数是可选参数。

Long Col

在表示空间中要开始检索的列位置。此参数是可选参数。

Long LenToGet

要从表示空间中检索的字符数。此参数是可选参数。缺省值是以 BuffLen 传递的数组长度。

返回值

字符串

来自 PS 的文本。

示例

以下示例展示了如何从与 autECLPS 对象关联的连接的表示空间中检索字符串。

```
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 方法会将字符串发送到表示空间，用于与 autECLPS 对象关联的连接。尽管此方法与 SendKeys 方法相似，但此方法不会发送助记符击键（例如 [enter] 或 [pf1]）。

原型

void SetText(String Text, [optional] Long Row, [optional] Long Col)

Parameters

字符串文本

要发送的字符数组。

Long Row

要从表示空间开始检索的行。此参数是可选参数。缺省值是当前光标行位置。

Long Col

要从表示空间开始检索的列位置。此参数是可选参数。缺省值是当前光标列位置。

返回值

无人

示例

以下示例展示了如何在表示空间中搜索文本，用于与 autECLPS 对象关联的连接。

```
Dim autECLPSObj as Object

'Initialize the connection
Set autECLPSObj = CreateObject("ZIEWin.autECLPS")
autECLPSObj.SetConnectionByName("A")
autECLPSObj.SetText"HCL is great", 2, 1
```

CopyText

此方法会将表示空间中从给定位置开始的指定长度文本复制到剪贴板。所复制文本的长度将是指定的长度，如果未指定长度，则复制直到表示空间末尾的文本。如果未指定位置，所复制文本将从表示空间中当前光标位置开始。如果未指定任何参数，会将整个表示空间复制到剪贴板。

原型

void CopyText([optional] Long Row, [optional] Long Col, [optional] Long LenToGet)

Parameters

Long LenToGet

要从表示空间中复制的字符数。此参数是可选参数。缺省值是以 BuffLen 传递的数组长度。

Long Row

要从表示空间开始复制的行。此参数是可选参数。缺省值是当前光标行位置。

Long Col

要从表示空间开始复制的列位置。此参数是可选参数。缺省值是当前光标列。

返回值

无人

示例

以下示例展示了如何将表示空间中的文本复制到剪贴板，用于与 autECLPS 对象关联的连接。

```
Dim autECLPSObj as Object
'Initialize the connection

Set autECLPSObj = CreateObject("ZIEWin.autECLPS")

autECLPSObj.SetConnectionByName("A")
autECLPSObj.CopyText 6, 53, 10
```

粘贴文本

此方法将指定长度的文本从剪贴板粘贴到演示空间中的给定位置。粘贴文本的长度是指定的长度，如果未指定长度，则粘贴剪贴板中的整个文本，直到到达演示空间的末尾。如果未指定位置，则文本将粘贴到演示空间中的当前光标位置。如果演示空间是字段格式的，并且在粘贴剪贴板内容时，当存在制表符“\t”时，剩余的粘贴内容将移动到下一个可写字段。

原型

void PasteText([optional] Long Row, [optional] Long Col, [optional] Long LenToGet)

Parameters

Long LenToGet

要从剪贴板粘贴到表示空间的字符数。此参数是可选参数。缺省值是剪贴板中的文本长度。

Long Row

要开始从剪贴板粘贴到表示空间的行。此参数是可选参数。缺省值是当前光标行位置。

Long Col

要开始从剪贴板粘贴到表示空间的列位置。此参数是可选参数。缺省值是当前光标列位置。

返回值

无人

示例

以下示例展示了如何将文本从剪贴板粘贴到与 autECLPS 对象关联的连接的表示空间。

```
Dim autECLPSObj as Object
'Initialize the connection

Set autECLPSObj = CreateObject("ZIEWin.autECLPS")

autECLPSObj.SetConnectionByName("A")
autECLPSObj.PasteText 8, 53, 10
```

GetTextRect

GetTextRect 方法会从与 autECLPS 对象关联的连接的表示空间的矩形区域中检索字符。文本检索中不涉及任何换行设置；仅检索矩形区域。

原型

StringGetTextRect(Long StartRow, Long StartCol, Long EndRow, Long EndCol)

Parameters

Long StartRow

在表示空间中要开始检索的行。

Long StartCol

在表示空间中要开始检索的列。

Long EndRow

在表示空间中要结束检索的行。

Long EndCol

在表示空间中要结束检索的列。

返回值

字符串

PS 文本。

示例

以下示例展示了如何从表示空间中的矩形区域检索字符，用于与 autECLPS 对象关联的连接。

```
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 方法将字符设置到 autECLPS 对象相关连接的表示空间内的矩形区域中。文本设置中不涉及任何换行设置；只设置矩形区域。

原型

SetTextRect(String Text, Long StartRow, Long StartCol, Long EndRow, Long EndCol)

Parameters

字符串文本

要在表示空间上设置的字符数组。

Long StartRow

在表示空间中要开始设置的行。

Long StartCol

在表示空间中要开始设置的列。

Long EndRow

在表示空间中要结束设置的行。

Long EndCol

在表示空间中要结束设置的列。

返回值

无人

示例

以下示例显示了如何将字符设置到 autECLPS 对象相关连接的表示空间内的矩形区域中。

```
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
```

如果要使用括号，则可以使用 SetTextRect 作为

```
call SetTextRect("HCL is great company to collaborate with", 1, 1, 4, 8)
```

StartCommunication

StartCommunication 集合元素方法会将 ZIEWin 仿真器连接到主机数据流。这与进入 ZIEWin 仿真器**通信**菜单并选择**连接**具有相同的效果。

原型

void StartCommunication()

Parameters

无人

返回值

无人

示例

以下示例展示了如何将 ZIEWin 仿真器会话连接到主机。

```
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 集合元素方法会断开 ZIEWin 仿真器与主机数据流的连接。这与进入 ZIEWin 仿真器**通信**菜单并选择**断开连接**具有相同的效果。

原型

void StopCommunication()

Parameters

无人

返回值

无人

示例

以下示例展示了如何将 ZIEWin 仿真器会话连接到主机。

```
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 方法会运行 MacroName 参数指示的 Z and I Emulator for Windows 宏文件。

原型

void StartMacro(String MacroName)

Parameters

String MacroName

位于 Z and I Emulator for Windows 用户类应用程序数据目录（在安装时指定）的宏文件的名称，不带文件扩展名。此方法不支持长文件名。

返回值

无人

使用注意事项

必须将短文件名用于宏名。此方法不支持长文件名。

示例

以下示例展示了如何启动宏。

```
Dim PS as Object

Set PS = CreateObject("ZIEWin.autECLPS")
PS.StartMacro "mymacro"
```

等待

Wait 方法会等待由 milliseconds 参数指定的毫秒数

原型

void Wait(milliseconds as Long)

Parameters

Long milliseconds

要等待的毫秒数。

返回值

无人

示例

```
Dim autECLPSObj as Object

Set autECLPSObj = CreateObject ("ZIEWin.autECLPS")
autECLPSObj.SetConnectionByName ("A")

' Wait for 10 seconds
autECLPSObj.Wait(10000)
```

WaitForCursor

WaitWhileCursor 方法等待与 autECLPS 对象相关联连接的“表示空间”中的光标位于一个指定位置。

原型

Boolean WaitForCursor(Variant Row, Variant Col, [optional]Variant TimeOut,
[optional] Boolean bWaitForIr)

Parameters

Variant Row

光标所在的行位置。

Variant Col

光标所在的列位置。

Variant TimeOut

以毫秒为单位的最长时间长度，此参数为可选。缺省值是无限长。

Boolean bWaitForIr

如果该值为 True，在符合等待条件之后，功能将等到 OIA 准备好接受输入为止。此参数是可选参数。缺省值为 False。

返回值

如果符合条件，方法返回 True；如果超过了 Timeout 值，则返回 False。

示例

```
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

当与 autECLPS 对象相关联连接的“表示空间”中的光标位于一个指定位置时，WaitWhileCursor 方法等待。

原型

Boolean WaitWhileCursor(Variant Row, Variant Col, [optional]Variant TimeOut,
[optional] Boolean bWaitForIr)

Parameters

Variant Row

光标所在的行位置。

Variant Col

光标所在的列位置。

Variant TimeOut

以毫秒为单位的最长时间长度，此参数为可选。缺省值是无限长。

Boolean bWaitForIr

如果该值为 True，在符合等待条件之后，功能将等到 OIA 准备好接受输入为止。此参数是可选参数。缺省值为 False。

返回值

如果符合条件，方法返回 True；如果超过了 Timeout 值，则返回 False。

示例

```
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 方法等待指定字符串出现在与 autECLPS 对象相关联连接的“表示空间”中。如果使用了可选的 Row 和 Column 参数，则字符串必须从指定位置开始。如果对 Row 和 Col 指定 0,0，则方法将搜索整个 PS。

原型

```
Boolean WaitForString(Variant WaitString, [optional] Variant Row,
    [optional] Variant Col, [optional] Variant TimeOut, [optional] Boolean bWaitForIr,
    [optional] Boolean bCaseSens)
```

Parameters

Variant WaitString

要等待的字符串。

Variant Row

字符串将开始的行位置。此参数是可选参数。缺省值为 0。

Variant Col

字符串将开始的列位置。此参数是可选参数。缺省值为 0。

Variant TimeOut

以毫秒为单位的最长时间长度，此参数为可选。缺省值是无限长。

Boolean bWaitForIr

如果该值为 True，在符合等待条件之后，功能将等到 OIA 准备好接受输入为止。此参数是可选参数。缺省值为 False。

Boolean bCaseSens

如果此值为 True，则等待条件被验证为区分大小写。此参数是可选参数。缺省值为 False。

返回值

如果符合条件，方法返回 True；如果超过了 Timeout 值，则返回 False。

示例

```
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

当指定字符串出现在与 autECLPS 对象相关联连接的“表示空间”中时，WaitWhileString 方法等待。如果使用了可选的 Row 和 Column 参数，则字符串必须从指定位置开始。如果对 Row 和 Col 指定 0,0，则方法将搜索整个 PS。

原型

```
Boolean WaitWhileString(Variant WaitString, [optional] Variant Row,
    [optional] Variant Col, [optional] Variant TimeOut, [optional] Boolean bWaitForIr,
    [optional] Boolean bCaseSens)
```

Parameters

Variant WaitString

该方法将在此字符串值存在时等待。

Variant Row

字符串将开始的行位置。此参数是可选参数。缺省值为 0。

Variant Col

字符串将开始的列位置。此参数是可选参数。缺省值为 0。

Variant TimeOut

以毫秒为单位的最长时间长度，此参数为可选。缺省值是无限长。

Boolean bWaitForIr

如果该值为 True，在符合等待条件之后，功能将等到 OIA 准备好接受输入为止。此参数是可选参数。缺省值为 False。

Boolean bCaseSens

如果此值为 True，则等待条件被验证为区分大小写。此参数是可选参数。缺省值为 False。

返回值

如果符合条件，方法返回 True；如果超过了 Timeout 值，则返回 False。

示例

```
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 方法会等待指定字符串出现在与指定矩形中的 autECLPS 对象相关联的连接的表示空间中。

原型

Boolean WaitForStringInRect(Variant WaitString, Variant sRow, Variant sCol,
Variant eRow, Variant eCol, [optional] Variant nTimeOut,
[optional] Boolean bWaitForIr, [optional] Boolean bCaseSens)

Parameters

Variant WaitString

要等待的字符串。

Variant sRow

搜索矩形的起始行位置。

Variant sCol

搜索矩形的起始列位置。

Variant eRow

搜索矩形的结束行位置。

Variant eCol

搜索矩形的结束列位置

Variant nTimeOut

以毫秒为单位的最长时间长度，此参数为可选。缺省值是无限长。

Boolean bWaitForIr

如果该值为 True，在符合等待条件之后，功能将等到 OIA 准备好接受输入为止。此参数是可选参数。缺省值为 False。

Boolean bCaseSens

如果此值为 True，则等待条件被验证为区分大小写。此参数是可选参数。缺省值为 False。

返回值

如果符合条件，方法返回 True；如果超过了 Timeout 值，则返回 False。

示例

```
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

当指定字符串出现在与指定矩形中的 autECLPS 对象相关联的连接的表示空间中时，WaitWhileStringInRect 方法会等待。

原型

Boolean WaitWhileStringInRect(Variant WaitString, Variant sRow, Variant sCol,
Variant eRow, Variant eCol, [optional] Variant nTimeOut,
[optional] Boolean bWaitForIr, [optional] Boolean bCaseSens)

Parameters

Variant WaitString

该方法将在此字符串值存在时等待。

Variant sRow

搜索矩形的起始行位置。

Variant sCol

搜索矩形的起始列位置。

Variant eRow

搜索矩形的结束行位置。

Variant eCol

搜索矩形的结束列位置。

Variant nTimeOut

以毫秒为单位的最长时间长度，此参数为可选。缺省值是无限长。

Boolean bWaitForIr

如果该值为 True，在符合等待条件之后，功能将等到 OIA 准备好接受输入为止。此参数是可选参数。缺省值为 False。

Boolean bCaseSens

如果此值为 True，则等待条件被验证为区分大小写。此参数是可选参数。缺省值为 False。

返回值

如果符合条件，方法返回 True；如果超过了 Timeout 值，则返回 False。

示例

```
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 方法将等待，直到指定的属性值出现在与指定行/列位置的 autECLPS 对象相关联连接的表示空间中。可选的 MaskData 参数可用于控制您正在查找哪个属性值。可选的平面参数允许您选择四个 PS 平面中的任一个。

原型

```
Boolean WaitForAttrib(Variant Row, Variant Col, Variant WaitData,
    [optional] Variant MaskData, [optional] Variant plane, [optional] Variant TimeOut,
    [optional] Boolean bWaitForIr)
```

Parameters

Variant Row

属性的行位置。

Variant Col

属性的列位置。

Variant WaitData

要等待的属性的 1 字节十六进制值。

Variant MaskData

作为属性掩码使用的 1 字节十六进制值。此参数是可选参数。缺省值为 0xFF。

Variant plane

要获取的属性的平面。该平面可有下列值：

1

文本平面

2

颜色平面

3

字段平面（缺省值）

4

扩展字段平面

Variant TimeOut

以毫秒为单位的最长时间长度，此参数为可选。缺省值是无限长。

Boolean bWaitForIr

如果该值为 True，在符合等待条件之后，功能将等到 OIA 准备好接受输入为止。此参数是可选参数。缺省值为 False。

返回值

如果符合条件，方法返回 True；如果超过了 Timeout 值，则返回 False。

示例

```
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

当指定的属性值出现在与指定行/列位置的 autECLPS 对象相关联连接的表示空间中时，WaitWhileAttrib 方法将等待。可选的 MaskData 参数可用于控制您正在查找哪个属性值。可选的平面参数允许您选择四个 PS 平面中的任一个。

原型

```
Boolean WaitWhileAttrib(Variant Row, Variant Col, Variant WaitData,
    [optional] Variant MaskData, [optional] Variant plane, [optional] Variant TimeOut,
    [optional] Boolean bWaitForIr)
```

Parameters

Variant Row

属性的行位置。

Variant Col

属性的列位置。

Variant WaitData

要等待的属性的 1 字节十六进制值。

Variant MaskData

作为属性掩码使用的 1 字节十六进制值。此参数是可选参数。缺省值为 0xFF。

Variant plane

要获取的属性的平面。该平面可有下列值：

1

文本平面

2

颜色平面

3

字段平面（缺省值）

4

扩展字段平面

Variant TimeOut

以毫秒为单位的最长时间长度，此参数为可选。缺省值是无限长。

Boolean bWaitForIr

如果该值为 True，在符合等待条件之后，功能将等到 OIA 准备好接受输入为止。此参数是可选参数。缺省值为 False。

返回值

如果符合条件，方法返回 True；如果超过了 Timeout 值，则返回 False。

示例

```
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

同步等待 autECLScreenDesc 参数描述的屏幕出现在表示空间中。

注： 在 autECLScreenDesc 对象上设置了等待 OIA 输入标志，它不会作为参数传递到 wait 方法。

原型

Boolean WaitForScreen(Object screenDesc, [optional] Variant TimeOut)

Parameters

Object screenDesc

描述屏幕的 autECLScreenDesc 对象（请参阅 autECLScreenDesc 类 (on page 324)）。

Variant TimeOut

以毫秒为单位的最长时间长度，此参数为可选。缺省值是无限长。

返回值

如果符合条件，方法返回 True；如果超过了 Timeout 值，则返回 False。

示例

```
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

同步等待至 autECLScreenDesc 参数描述的屏幕不再位于表示空间中。

注： 在 autECLScreenDesc 对象上设置了等待 OIA 输入标志，它不会作为参数传递到 wait 方法。

原型

Boolean WaitWhileScreen(Object screenDesc, [optional] Variant TimeOut)

Parameters

Object ScreenDesc

描述屏幕的 autECLScreenDesc 对象（请参阅 autECLScreenDesc 类 (on page 324)）。

Variant TimeOut

以毫秒为单位的最长时间长度，此参数为可选。缺省值是无限长。

返回值

如果符合条件，方法返回 True；如果超过了 Timeout 值，则返回 False。

示例

```
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

取消任何当前活动的 wait 方法。

原型

void CancelWaits()

Parameters

无人

返回值

无人

autECLPS 事件

以下事件对 autECLPS 有效：

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

给定的 PS 已更新。

原型

void NotifyPSEvent()

Parameters

无人

示例

有关示例，请参阅事件处理示例 (on page 322)。

NotifyKeyEvent

发生了击键事件，并且提供了键信息。此函数可用于拦截对给定 PS 的击键。键信息将被传递给事件处理程序，可以继续传递，也可以执行其他操作。

注： 一次只能有一个对象具有向给定 PS 注册的击键事件处理。

原型

void NotifyKeyEvent(string KeyType, string KeyString, PassItOn as Boolean)

Parameters

String KeyType

拦截的键类型。

周一

助记符击键

A

ASCII

String KeyString

拦截的击键

Boolean PassItOn

用于指示击键是否应回传到 PS 的标志。

真

允许将击键传递到 PS。

FALSE

防止将击键传递到 PS。

示例

有关示例，请参阅事件处理示例 (on page 322)。

NotifyCommEvent

已连接或已断开连接的给定通信链路。

原型

void NotifyCommEvent(boolean bConnected)

Parameters

boolean bConnected

如果通信链路当前已连接，则返回 True；否则返回 False。

示例

有关示例，请参阅事件处理示例 (on page 322)。

NotifyPSError

事件处理出错时会发生此事件。

原型

void NotifyPSError()

Parameters

无人

示例

有关示例，请参阅事件处理示例 (on page 322)。

NotifyKeyError

事件处理出错时会发生此事件。

原型

void NotifyKeyError()

Parameters

无人

示例

有关示例，请参阅事件处理示例 (on page 322)。

NotifyCommError

事件处理出错时会发生此事件。

原型

void NotifyCommError()

Parameters

无人

示例

有关示例，请参阅事件处理示例 (on page 322)。

NotifyPSStop

事件处理停止时会发生此事件。

原型

void NotifyPSStop(Long Reason)

Parameters

Long Reason

停止的原因码。目前，这将始终为 0。

示例

有关示例，请参阅事件处理示例 (on page 322)。

NotifyKeyStop

事件处理停止时会发生此事件。

原型

void NotifyKeyStop(Long Reason)

Parameters

Long Reason

停止的原因码。目前，这将始终为 0。

示例

有关示例，请参阅事件处理示例 (on page 322)。

NotifyCommStop

事件处理停止时会发生此事件。

原型

void NotifyCommStop(Long Reason)

Parameters

Long Reason

停止的原因码。目前，这将始终为 0。

事件处理示例

以下是有关如何实现 PS 事件的简短示例

```
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 类

autECLScreenDesc 类用于描述 HCL 主机访问类库屏幕识别技术的屏幕。它使用表示空间的所有四个主要平面来描述它（文本、字段、扩展字段和颜色平面），以及游标位置。

使用此对象上提供的方法，程序员可以设置主机端应用程序中给定屏幕外观的详细描述。创建并设置 autECLScreenDesc 对象后，可以将其传递到同步 autECLPS 上提供的 WaitFor... 方法，也可以将其传递到 autECLScreenReco，后者会在 PS 中显示与 autECLScreenDesc 对象匹配的屏幕时触发异步事件。

autECLScreenDesc 方法

下节介绍了对 autECLScreenDesc 有效的方法。

```
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

将给定位置的属性值添加到屏幕描述。

原型

void AddAttrib(Variant attrib, Variant row, Variant col, Variant plane)

Parameters

Variant attrib

属性的 1 字节十六进制值。

Variant row

行位置。

Variant col

列位置。

Variant plane

要获取的属性的平面。该平面可有下列值：

0.所有平面
1.文本平面
2.颜色平面
3.字段平面
4.扩展字段平面

返回值

无人

示例

```
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

将屏幕描述的游标位置设置为给定位置。

原型

void AddCursorPos(Variant row, Variant col)

Parameters

Variant row

行位置。

Variant col

列位置。

返回值

无人

示例

```
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

将字段数添加到屏幕描述。

原型

void AddNumFields(Variant num)

Parameters

Variant num

字段数。

返回值

无人

示例

```
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

将字段数添加到屏幕描述。

原型

void AddNumInputFields(Variant num)

Parameters

Variant num

输入字段数。

返回值

无人

示例

```
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

设置屏幕描述的 OIA 监控类型。

原型

void AddOIAInhibitStatus(Variant type)

Parameters

Variant type

OIA 状态的类型。有效值如下所示：

0.无所谓
1.未禁止

返回值

无人

示例

```
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

将给定位置的字符串添加到屏幕描述。

原型

```
void AddString(String str, Variant row, Variant col, [optional] Boolean caseSense)
```

Parameters

String str

要添加的字符串。

Variant row

行位置。

Variant col

列位置。

Boolean caseSense

如果此值为 True，则将字符串添加为区分大小写。此参数是可选参数。缺省值为 True。

返回值

无人

示例

```
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

将给定矩形中的字符串添加到屏幕描述。

原型

void AddStringInRect(String str, Variant sRow, Variant sCol,
Variant eRow, Variant eCol, [optional] Variant caseSense)

Parameters

String str

要添加的字符串

Variant sRow

左上行位置。

Variant sCol

左上列位置。

Variant eRow

右下行位置。

Variant eCol

右下列位置。

Variant caseSense

如果此值为 True，则将字符串添加为区分大小写。此参数是可选参数。缺省值为 True。

返回值

无人

示例

```
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
```

Clear

从屏幕描述中删除所有描述元素。

原型

void Clear()

Parameters

无人

返回值

无人

示例

```
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 类

autECLScreenReco 类是主机访问类库屏幕识别系统的引擎。它包含添加和删除屏幕描述的方法。它还包含用于识别这些屏幕以及异步回调这些屏幕的事件处理程序代码的逻辑。

将 autECLScreenReco 类的对象视为唯一识别集。该对象可以有多个 autECLPS 对象，它可以监视屏幕和多个要查找的屏幕，当它在任何添加的 autECLPS 对象中看到注册的屏幕时，它将触发应用程序中定义的事件处理代码。

只需在应用程序启动时设置 autECLScreenReco 对象，当任何要监控的 autECLPS 中出现任何屏幕时，autECLScreenReco 就将调用事件代码。您完全无需监控屏幕。

有关示例，请参阅事件处理示例 (on page 337)。

autECLScreenReco 方法

下节介绍了对 autECLScreenReco 有效的方法。

void AddPS(autECLPS ps)
Boolean IsMatch(autECLPS ps, AutECLScreenDesc sd)
void RegisterScreen(AutECLScreenDesc sd)
void RemovePS(autECLPS ps)
void UnregisterScreen(AutECLScreenDesc sd)

AddPS

将要监控的 autECLPS 对象添加到 autECLScreenReco 对象。

原型

void AddPS(autECLPS ps)

Parameters

autECLPS ps

要监控的 PS 对象。

返回值

无人

示例

有关示例，请参阅事件处理示例 (on page 337)。

IsMatch

允许传递 autECLPS 对象和 AutECLScreenDesc 对象，并确定屏幕描述是否与 PS 的当前状态匹配。屏幕识别引擎使用此逻辑，但提供此逻辑是为了使任何例程都可以调用它。

原型

Boolean IsMatch(autECLPS ps, AutECLScreenDesc sd)

Parameters

autECLPS ps

要比较的 autPS 对象。

AutECLScreenDesc sd

要比较的 autECLScreenDesc 对象。

返回值

如果 AutECLScreenDesc 对象与 PS 中的当前屏幕匹配，则返回 True，否则返回 False。

示例

```
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

开始监控添加到给定屏幕描述的屏幕识别对象的所有 autECLPS 对象。如果屏幕出现在 PS 中，则将出现 NotifyRecoEvent。

原型

void RegisterScreen(AutECLScreenDesc sd)

Parameters

AutECLScreenDesc sd

要注册的屏幕描述对象。

返回值

无人

示例

有关示例，请参阅事件处理示例 (on page 337)。

RemovePS

从屏幕识别监控中删除 autECLPS 对象。

原型

void RemovePS(autECLPS ps)

Parameters

autECLPS ps

要删除的 autECLPS 对象。

返回值

无人

示例

有关示例，请参阅事件处理示例 (on page 337)。

UnregisterScreen

从屏幕识别监控中删除屏幕描述。

原型

void UnregisterScreen(AutECLScreenDesc sd)

Parameters

AutECLScreenDesc sd

要删除的屏幕描述对象。

返回值

无人

示例

有关示例，请参阅事件处理示例 (on page 337)。

autECLScreenReco 事件

以下事件对 autECLScreenReco 有效：

void NotifyRecoEvent(AutECLScreenDesc sd, autECLPS ps)
void NotifyRecoError()
void NotifyRecoStop(Long Reason)

NotifyRecoEvent

当已注册屏幕描述出现在添加到 autECLScreenReco 对象的 PS 中时，会发生此事件。

原型

void NotifyRecoEvent(AutECLScreenDesc sd, autECLPS ps)

Parameters

AutECLScreenDesc sd

满足其条件的 Screen Description 对象。

autECLPS ps

存在匹配项的 PS 对象。

示例

有关示例，请参阅事件处理示例 (on page 337)。

NotifyRecoError

事件处理出错时会发生此事件。

原型

void NotifyRecoError()

Parameters

无人

示例

有关示例，请参阅事件处理示例 (on page 337)。

NotifyRecoStop

事件处理停止时会发生此事件。

原型

void NotifyRecoStop(Long Reason)

Parameters

Long Reason

停止的原因码。目前，这将始终为 0。

事件处理示例

以下是有关如何实现屏幕识别事件的简短示例：

```
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 类

autECLSession 对象提供与常规仿真器相关的服务并包含指向主机访问类库中其他密钥对象的指针。它在注册表中的名称是 ZIEWin.autECLSession。

尽管 autECLSession 包含的对象能够独立运行，但指向它们的指针存在于 autECLSession 类中。创建 autECLSession 对象时，还将创建 autECLPS、autECLOIA、autECLXfer、autECLWindowMetrics、autECLPageSettings 和 autECLPrinterSettings 对象。就像引用任何其他属性一样引用它们。

注:

1. 此对象的当前版本为 1.2。此对象有两个版本；它们在注册表中的 ProgID 是 ZIEWin.autECLSession.1 和 ZIEWin.autECLSession.2。与版本无关的 ProgID 是 ZIEWin.autECLSession。ZIEWin.autECLSession.1 对象不支持 autECLPageSettings 和 autECLPrinterSettings 属性。
2. 必须在初始时为创建的对象设置连接。使用 SetConnectionByName 或 SetConnectionByHandle 来初始化对象。只能对连接进行一次设置。在设置连接之后，任何再调用 SetConnection 方法都将引发异常。如果未设置连接并尝试访问 autECLSession 属性或方法，则还会引发异常。

以下示例展示了如何在 Visual Basic 中创建和设置 autECLSession 对象。

```
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
```

属性

本节介绍了 autECLSession 对象的属性。

类型	名称	属性
字符串	姓名	只读
长	句柄	只读

类型	名称	属性
字符串	ConnType	只读
长	CodePage	只读
布尔值	已启动	只读
布尔值	CommStarted	只读
布尔值	APIEnabled	只读
布尔值	就绪	只读
对象	autECLPS	只读
对象	autECLOIA	只读
对象	autECLXfer	只读
对象	autECLWinMetrics	只读
对象	autECLPageSettings	只读
对象	autECLPrinterSettings	只读

姓名

此属性是设置了 autECLSession 的连接的连接名称字符串。Z and I Emulator for Windows 仅返回字符串中的短字符标识（A-Z 或 a-z）。给定名称只能有一个打开的 Z and I Emulator for Windows 连接。例如，一次只能有一个打开的连接“A”。Name 是字符串数据类型且为只读。下列示例展示了此属性。

```
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
```

句柄

这是设置了 autECLSession 对象的连接的句柄。给定句柄只能有一个打开的 Z and I Emulator for Windows 连接。例如，一次只能有一个打开的连接“A”。Handle 是长整型数据类型且为只读。下列示例展示了此属性。

```
DIM  SessObj as Object
Set SessObj = CreateObject("ZIEWin.autECLSession")

' Initialize the session
SessObj.SetConnectionByName("A")

' Save the session handle
Hand = SessObj.Handle
```

ConnType

这是设置了 autECLXfer 的连接类型。这种类型可能会随着时间的推移而更改。ConnType 是字符串数据类型且为只读。下列示例展示了此属性。

```
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 属性的连接类型包括：

返回的字符串	意义
DISP3270	3270 显示
DISP5250	5250 显示
PRNT3270	3270 打印机
PRNT5250	5250 打印机
ASCII	VT 仿真

CodePage

这是设置了 autECLXfer 的连接的代码页。此代码页可能会随着时间的推移而更改。CodePage 是长整型数据类型且为只读。下列示例展示了此属性。

```
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
```

已开始

此项会指示是否已启动仿真器窗口。如果窗口已打开，则值为 True；否则为 False。Started 是布尔数据类型且为只读。下列示例展示了此属性。

```
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

此项会指示与主机的连接的状态。如果主机已连接，则值为 True；否则为 False。CommStarted 是布尔数据类型且为只读。下列示例展示了此属性。

```
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

此项会指示仿真器是否已启用 API。连接可能已启用或已禁用 API，具体取决于其 API 设置的状态（在 Z and I Emulator for Windows 窗口中，选择**文件 → API 设置**）。如果仿真器已启用，则值为 True；否则为 False。APIEnabled 是布尔数据类型且为只读。下列示例展示了此属性。

```
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
```

Ready

此项会指示仿真器窗口是否已启动、已启用 API 和已连接。此属性会检查是否有所有三个属性。如果仿真器就绪，则值为 True；否则为 False。Ready 是布尔数据类型且为只读。下列示例展示了此属性。

```
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 对象

autECLPS 对象允许您访问 ZIEWin.autECLPS 类中包含的方法。请参阅 autECLPS 类 (on page 285) 以获取更多信息。下列示例展示了此对象。

```
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 对象

autECLOIA 对象允许您访问.ZIEWin.autECLOIA 类中包含的方法。请参阅 autECLOIA 类 (on page 268) 以获取更多信息。下列示例展示了此对象。

```
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 对象

autECLXfer 对象允许您访问 ZIEWin.autECLXfer 类中包含的方法。请参阅 autECLXfer 类 (on page 366) 以获取更多信息。下列示例展示了此对象。

```
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 对象

autECLWinMetrics 对象允许您访问 ZIEWin.autECLWinMetrics 类中包含的方法。请参阅 autECLWinMetrics 类 (on page 351) 以获取更多信息。下列示例展示了此对象。

```
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 对象

autECLPageSettings 对象让您可访问 ZIEWin.autECLPageSettings 类中包含的方法。请参阅 autECLPageSettings 类 (on page 380) 以获取更多信息。

以下示例展示了 autECLPageSettings 对象。

```
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"
```

VBSCRIPT 中也支持 autECLPageSettings 对象。以下示例展示了如何使用 VBSCRIPT。

```
sub test_()
  autECLSession.SetConnectionByName(ThisSessionName)
  autECLSession.autECLPageSettings.FaceName="Courier"
 end sub
```

autECLPrinterSettings 对象

autECLPrinterSettings 对象让您可访问 ZIEWin.autECLPrinterSettings 类中包含的方法。请参阅 autECLPageSettings 类 (on page 380) 以获取更多信息。

以下示例展示了 autECLPageSettings 对象。

```
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
```

VBSCRIPT 中也支持 autECLPrinterSettings 对象。以下示例展示了如何使用 VBSCRIPT。

```
sub test_()
  autECLSession.SetConnectionByName(ThisSessionName)
  autECLSession.autECLPrinterSettings.SetWinDefaultPrinter
end sub
```

autECLSession 方法

下节介绍了对 autECLSession 对象有效的方法。

void RegisterSessionEvent(Long updateType)
void RegisterCommEvent()
void UnregisterSessionEvent()
void UnregisterCommEvent()
void SetConnectionByName (String Name)
void SetConnectionByHandle (Long Handle)
void StartCommunication()
void StopCommunication()

RegisterSessionEvent

此方法会注册 autECLSession 对象，以接收指定会话事件的通知。

注： 此方法不受支持，不建议使用。

原型

void RegisterSessionEvent(Long updateType)

Parameters

Long updateType

要监控的更新类型：

1. PS 更新
2. OIA 更新
3. PS 或 OIA 更新

返回值

无人

示例

有关示例，请参阅事件处理示例 (on page 350)。

RegisterCommEvent

此方法会注册一个对象，以接收所有通信链路连接/断开连接事件的通知。

原型

void RegisterCommEvent()

Parameters

无人

返回值

无人

示例

有关示例，请参阅事件处理示例 (on page 350)。

UnregisterSessionEvent

结束会话事件处理。

注： 此方法不受支持，不建议使用。

原型

void UnregisterSessionEvent()

Parameters

无人

返回值

无人

示例

有关示例，请参阅事件处理示例 (on page 350)。

UnregisterCommEvent

结束通信链路事件处理。

原型

void UnregisterCommEvent()

Parameters

无人

返回值

无人

示例

有关示例，请参阅事件处理示例 (on page 350)。

SetConnectionByName

此方法使用连接名称为新创建的 autECLSession 对象设置连接。在 Z and I Emulator for Windows 中，此连接名称是短标识（字符 A-Z 或 a-z）。给定名称只能有一个打开的 Z and I Emulator for Windows 连接。例如，一次只能有一个打开的连接“A”。

原型

void SetConnectionByName(String Name)

Parameters

String Name

连接的单字符字符串短名称（A-Z 或 a-z）。

返回值

无人

示例

以下示例展示了如何使用连接名称为新创建的 autECLSession 对象设置连接。

```
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

此方法使用连接句柄为新创建的 autECLSession 对象设置连接。在 Z and I Emulator for Windows 中，此连接句柄是长整型值。给定句柄只能有一个打开的 Z and I Emulator for Windows 连接。例如，一次只能有一个打开的连接“A”。

原型

void SetConnectionByHandle(Long Handle)

Parameters

Long Handle

要为对象设置的连接的长整型值。

返回值

无人

示例

以下示例展示了如何使用连接句柄为新创建的 autECLSession 对象设置连接。

```
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 集合元素方法会将 ZIEWin 仿真器连接到主机数据流。这与进入 ZIEWin 仿真器**通信**菜单并选择**连接**具有相同的效果。

原型

void StartCommunication()

Parameters

无人

返回值

无人

示例

以下示例展示了如何将 ZIEWin 仿真器会话连接到主机。

```
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 集合元素方法会断开 ZIEWin 仿真器与主机数据流的连接。这与进入 ZIEWin 仿真器**通信**菜单并选择**断开连接**具有相同的效果。

原型

void StopCommunication()

Parameters

无人

返回值

无人

示例

以下示例展示了如何将 ZIEWin 仿真器会话连接到主机。

```
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 事件

以下事件对 autECLSession 有效：

void NotifyCommEvent(boolean bConnected)

void NotifyCommError()

void NotifyCommStop(Long Reason)

NotifyCommEvent

给定的通信链路已连接或已断开连接。

原型

void NotifyCommEvent(boolean bConnected)

Parameters

boolean bConnected

如果通信链路当前已连接，则返回 TRUE。否则，其标号标识为 FALSE。

示例

有关示例，请参阅事件处理示例 (on page 350)。

NotifyCommError

事件处理出错时会发生此事件。

原型

void NotifyCommError()

Parameters

无人

示例

有关示例，请参阅事件处理示例 (on page 350)。

NotifyCommStop

事件处理停止时会发生此事件。

原型

void NotifyCommStop(Long Reason)

Parameters

Long Reason

停止的原因码。目前，这将始终为 0。

事件处理示例

以下是有关如何实现会话事件的简短示例

```
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 类

autECLWinMetrics 对象会对仿真器窗口执行操作。它允许您执行窗口矩形和位置操控（例如 SetWindowRect、Ypos 和 Width），以及窗口状态操控（例如 Visible 或 Restored）。它在注册表中的名称是 ZIEWin.autECLWinMetrics。

必须在初始时为创建的对象设置连接。使用 SetConnectionByName 或 SetConnectionByHandle 来初始化对象。只能对连接进行一次设置。在设置连接之后，任何再调用设置连接方法都将引发异常。如果未设置连接，并尝试访问属性或方法，也将引发异常。

注： autECL 对象中的 autECLSession 对象由 autECL 对象设置。

以下示例展示了如何在 Visual Basic 中创建和设置 autECLWinMetrics 对象。

```
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
```

属性

本节介绍了 autECLWinMetrics 对象的属性。

类型	名称	属性
字符串	WindowTitle	读/写
长	Xpos	读/写
长	Ypos	读/写
长	宽度	读/写
长	高度	读/写
布尔值	可见	读/写
布尔值	活动	读/写
布尔值	Minimized	读/写
布尔值	Maximized	读/写
布尔值	Restored	读/写
字符串	姓名	只读
长	句柄	只读
字符串	ConnType	只读
长	CodePage	只读
布尔值	已启动	只读
布尔值	CommStarted	只读
布尔值	APIEnabled	只读
布尔值	就绪	只读

WindowTitle

这是当前在标题栏中、与 autECLWinMetrics 对象相关联连接的标题。该属性既能更改又能检索。WindowTitle 是字符串数据类型，并且已启用读/写。下列示例展示了此流程。下列示例展示了此属性。

```
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
```

使用注意事项

如果将 WindowTitle 设置为空白，则连接的窗口标题将恢复为其原始设置。

Xpos

这是仿真器窗口矩形左上点的 *x* 位置。该属性既能更改又能检索。Xpos 是长整型数据类型，并且已启用读/写。但是，如果连接到的连接是就地嵌入式对象，则此属性为只读。下列示例展示了此属性。

```
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

这是仿真器窗口矩形左上点的 *y* 位置。该属性既能更改又能检索。Ypos 是长整型数据类型，并且已启用读/写。但是，如果连接到的连接是就地嵌入式对象，则此属性为只读。下列示例展示了此属性。

```
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
```

宽度

这是仿真器窗口矩形的宽度。该属性既能更改又能检索。Width 是长整型数据类型，并且已启用读/写。但是，如果连接到的连接是就地嵌入式对象，则此属性为只读。下列示例展示了此属性。

```
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
```

高度

这是仿真器窗口矩形的高度。该属性既能更改又能检索。Height 是长整型数据类型，并且已启用读/写。但是，如果连接到的连接是就地嵌入式对象，则此属性为只读。下列示例展示了此属性。

```
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
```

可见

这是仿真器窗口的可见性状态。该属性既能更改又能检索。Visible 是布尔数据类型，并且已启用读/写。但是，如果连接到的连接是就地嵌入式对象，则此属性为只读。下列示例展示了此属性。

```
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
```

活动

这是仿真器窗口的焦点状态。该属性既能更改又能检索。Active 是布尔数据类型，并且已启用读/写。但是，如果连接到的连接是就地嵌入式对象，则此属性为只读。下列示例展示了此属性。

```
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
```

Minimized

这是仿真器窗口的最小化状态。该属性既能更改又能检索。Minimized 是布尔数据类型，并且已启用读/写。但是，如果连接到的连接是就地嵌入式对象，则此属性为只读。下列示例展示了此属性。

```
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
```

Maximized

这是仿真器窗口的最大化状态。该属性既能更改又能检索。Maximized 是布尔数据类型，并且已启用读/写。但是，如果连接到的连接是就地嵌入式对象，则此属性为只读。下列示例展示了此属性。

```
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
```

Restored

这是仿真器窗口的恢复状态。Restored 是布尔数据类型，并且已启用读/写。但是，如果连接到的连接是就地嵌入式对象，则此属性为只读。下列示例展示了此属性。

```
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
```

姓名

此属性是设置了 autECLWinMetrics 的连接的连接名称字符串。当前，Z and I Emulator for Windows 仅返回字符串中的短字符标识（A-Z 或 a-z）。给定名称只能有一个打开的 Z and I Emulator for Windows 连接。例如，一次只能有一个打开的连接“A”。Name 是字符串数据类型且为只读。下列示例展示了此属性。

```
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
```

句柄

这是设置了 autECLWinMetrics 对象的连接的句柄。给定句柄只能有一个打开的 Z and I Emulator for Windows 连接。例如，一次只能有一个打开的连接“A”。Handle 是长整型数据类型且为只读。下列示例展示了此属性。

```
DIM  Obj as Object
Set Obj = CreateObject("ZIEWin.autECLWinMetrics")

' Initialize the connection
Obj.SetConnectionByName("A")

' Save the handle
Hand = Obj.Handle
```

ConnType

这是设置了 autECLWinMetrics 的连接类型。这种类型可能会随着时间的推移而更改。ConnType 是字符串数据类型且为只读。下列示例展示了此属性。

```
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 属性的连接类型包括：

返回的字符串	意义
DISP3270	3270 显示
DISP5250	5250 显示
PRNT3270	3270 打印机
PRNT5250	5250 打印机
ASCII	VT 仿真

CodePage

这是设置了 autECLWinMetrics 的连接的代码页。此代码页可能会随着时间的推移而更改。CodePage 是长整型数据类型且为只读。下列示例展示了此属性。

```
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
```

已开始

此项会指示是否已启动仿真器窗口。如果窗口已打开，则值为 True；否则为 False。Started 是布尔数据类型且为只读。下列示例展示了此属性。

```
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

此项会指示与主机的连接的状态。如果主机已连接，则值为 True；否则为 False。CommStarted 是布尔数据类型且为只读。下列示例展示了此属性。

```
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

此项会指示仿真器是否已启用 API。连接可能已启用或已禁用 API，具体取决于其 API 设置的状态（在 Z and I Emulator for Windows 窗口中，选择**文件 → API 设置**）。如果仿真器已启用，则值为 True；否则为 False。APIEnabled 是布尔数据类型且为只读。下列示例展示了此属性。

```
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
```

Ready

此项会指示仿真器窗口是否已启动、已启用 API 和已连接。此属性会检查是否有所有三个属性。如果仿真器就绪，则值为 True；否则为 False。Ready 是布尔数据类型且为只读。下列示例展示了此属性。

```
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 方法

下节介绍了对 autECLWinMetrics 对象有效的方法。

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

此方法会注册一个对象，以接收所有通信链路连接/断开连接事件的通知。

原型

void RegisterCommEvent()

Parameters

无人

返回值

无人

示例

有关示例，请参阅事件处理示例 (on page 366)。

UnregisterCommEvent

结束通信链路事件处理。

原型

void UnregisterCommEvent()

Parameters

无人

返回值

无人

SetConnectionByName

此方法使用连接名称为新创建的 autECLWinMetrics 对象设置连接。在 Z and I Emulator for Windows 中，此连接名称是短标识（字符 A-Z 或 a-z）。给定名称只能有一个打开的 Z and I Emulator for Windows 连接。例如，一次只能有一个打开的连接“A”。

注： 如果在 autECLSession 中使用 autECLWinMetrics 对象，请不要调用此项。

原型

void SetConnectionByName(String Name)

Parameters

String Name

连接的单字符字符串短名称（A-Z 或 a-z）。

返回值

无人

示例

以下示例展示了如何使用连接名称为新创建的 autECLWinMetrics 对象设置连接。

```
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

此方法使用连接句柄为新创建的 autECLWinMetrics 对象设置连接。在 Z and I Emulator for Windows 中，此连接句柄是长整型值。给定句柄只能有一个打开的 Z and I Emulator for Windows 连接。例如，一次只能有一个打开的连接“A”。

注： 如果在 autECLSession 中使用 autECLWinMetrics 对象，请不要调用此项。

原型

void SetConnectionByHandle(Long Handle)

Parameters

Long Handle

要为对象设置的连接的长整型值。

返回值

无人

示例

以下示例展示了如何使用连接句柄为新创建的 autECLWinMetrics 对象设置连接。

```
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 方法会返回仿真器窗口矩形的边界点。

原型

void GetWindowRect(Variant Left, Variant Top, Variant Right, Variant Bottom)

Parameters

Variant Left, Top, Right, Bottom

仿真器窗口的边界点。

返回值

无人

示例

以下示例展示了如何返回仿真器窗口矩形的边界点。

```
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 方法会设置仿真器窗口矩形的边界点。

原型

void SetWindowRect(Long Left, Long Top, Long Right, Long Bottom)

Parameters

Long Left, Top, Right, Bottom

仿真器窗口的边界点。

返回值

无人

示例

以下示例展示了如何设置仿真器窗口矩形的边界点。

```
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 集合元素方法会将 ZIEWin 仿真器连接到主机数据流。这与进入 ZIEWin 仿真器**通信**菜单并选择**连接**具有相同的效果。

原型

void StartCommunication()

Parameters

无人

返回值

无人

示例

以下示例展示了如何将 ZIEWin 仿真器会话连接到主机。

```
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 集合元素方法会断开 ZIEWin 仿真器与主机数据流的连接。这与进入 ZIEWin 仿真器**通信**菜单并选择**断开连接**具有相同的效果。

原型

void StopCommunication()

Parameters

无人

返回值

无人

示例

以下示例展示了如何将 ZIEWin 仿真器会话连接到主机。

```
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 事件

以下事件对 autECL WinMetrics 有效：

void NotifyCommEvent(boolean bConnected)
NotifyCommError()
void NotifyCommStop(Long Reason)

NotifyCommEvent

已连接或已断开连接的给定通信链路。

原型

void NotifyCommEvent(boolean bConnected)

Parameters

boolean bConnected

如果通信链路当前已连接，则返回 True；否则返回 False。

示例

有关示例，请参阅事件处理示例 (on page 366)。

NotifyCommError

事件处理出错时会发生此事件。

原型

NotifyCommError()

Parameters

无人

示例

有关示例，请参阅事件处理示例 (on page 366)。

NotifyCommStop

事件处理停止时会发生此事件。

原型

void NotifyCommStop(Long Reason)

Parameters

Long Reason

停止的原因码。目前，这将始终为 0。

事件处理示例

以下是有关如何实现 WinMetrics 事件的简短示例。

```
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 类

autECLXfer 对象提供文件传送服务。它在注册表中的名称是 ZIEWin.autECLXfer。

必须在初始时为创建的对象设置连接。使用 SetConnectionByName 或 SetConnectionByHandle 来初始化对象。只能对连接进行一次设置。在设置连接之后，任何再调用 SetConnection 方法都将引发异常。如果未设置连接，并尝试访问 autECLXfer 属性或方法，则也会引发异常。下面展示了如何在 Visual Basic 中创建和设置 autECLXfer 对象。

```
DIM  XferObj as Object

Set XferObj = CreateObject("ZIEWin.autECLXfer")

' Initialize the connection
XferObj.SetConnectionByName("A")
```

属性

本节介绍了 autECLXfer 对象的属性。

Type	名称	特性
字符串	姓名	只读
长	句柄	只读
字符串	ConnType	只读
长	CodePage	只读
布尔值	已启动	只读
布尔值	CommStarted	只读
布尔值	APIEnabled	只读
布尔值	就绪	只读

姓名

此属性是设置了 autECLXfer 的连接的连接名称字符串。Z and I Emulator for Windows 仅返回字符串中的短字符标识（A-Z 或 a-z）。给定名称只能有一个打开的 Z and I Emulator for Windows 连接。例如，一次只能有一个打开的连接“A”。Name 是字符串数据类型且为只读。下列示例展示了此属性。

```
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
```

句柄

这是设置了 autECLXfer 对象的连接的句柄。给定句柄只能有一个打开的 Z and I Emulator for Windows 连接。例如，一次只能有一个打开的连接“A”。Handle 是长整型数据类型且为只读。下列示例展示了此属性。

```
DIM  Obj as Object
Set Obj = CreateObject("ZIEWin.autECLXfer")

' Initialize the connection
Obj.SetConnectionByName("A")

' Save the handle
Hand = Obj.Handle
```

ConnType

这是设置了 autECLXfer 的连接类型。这种类型可能会随着时间的推移而更改。Conntype 是字符串数据类型且为只读。下列示例展示了此属性。

```
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 属性的连接类型包括：

返回的字符串	**意义**
DISP3270	3270 显示
DISP5250	5250 显示
PRNT3270	3270 打印机
PRNT5250	5250 打印机
ASCII	VT 仿真

CodePage

这是设置了 autECLXfer 的连接的代码页。此代码页可能会随着时间的推移而更改。CodePage 是长整型数据类型且为只读。下列示例展示了此属性。

```
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
```

已开始

此项会指示是否已启动仿真器窗口。如果窗口已打开，则值为 True；否则为 False。Started 是布尔数据类型且为只读。下列示例展示了此属性。

```
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

此项会指示与主机的连接的状态。如果主机已连接，则值为 True；否则为 False。CommStarted 是布尔数据类型且为只读。下列示例展示了此属性。

```
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

此项会指示仿真器是否已启用 API。连接可能已启用或已禁用 API，具体取决于其 API 设置的状态（在 Z and I Emulator for Windows 窗口中，选择**文件 → API 设置**）。如果仿真器已启用，则值为 True；否则为 False。APIEnabled 是布尔数据类型且为只读。下列示例展示了此属性。

```
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
```

Ready

此项会指示仿真器窗口是否已启动、已启用 API 和已连接。此属性会检查是否有所有三个属性。如果仿真器就绪，则值为 True；否则为 False。Ready 是布尔数据类型且为只读。下列示例展示了此属性。

```
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 方法

下节介绍了对 autECLXfer 对象有效的方法。

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

此方法会注册一个对象，以接收所有通信链路连接/断开连接事件的通知。

原型

void RegisterCommEvent()

Parameters

无人

返回值

无人

示例

有关示例，请参阅事件处理示例 (on page 378)。

UnregisterCommEvent

结束通信链路事件处理。

原型

void UnregisterCommEvent()

Parameters

无人

返回值

无人

SetConnectionByName

SetConnectionByName 方法使用连接名称为新创建的 autECLXfer 对象设置连接。在 Z and I Emulator for Windows 中，此连接名称是短标识（字符 A-Z 或 a-z）。给定名称只能有一个打开的 Z and I Emulator for Windows 连接。例如，一次只能有一个打开的连接“A”。

注： 如果在 autECLSession 中使用 autECLXfer 对象，请不要调用此项。

原型

void SetConnectionByName(String Name)

Parameters

String Name

连接的单字符字符串短名称（A-Z 或 a-z）。

返回值

无人

示例

以下示例展示了如何使用连接名称为新创建的 autECLXfer 对象设置连接。

```
DIM  XferObj as Object

Set XferObj = CreateObject("ZIEWin.autECLXfer")

' Initialize the connection
XferObj.SetConnectionByName("A")
```

SetConnectionByHandle

SetConnectionByHandle 方法使用连接名称为新创建的 autECLXfer 对象设置连接。在 Z and I Emulator for Windows 中，此连接句柄是长整型值。给定句柄只能有一个打开的 Z and I Emulator for Windows 连接。例如，一次只能有一个打开的连接 “A” 。

注： 如果在 autECLSession 中使用 autECLXfer 对象，请不要调用此项。

原型

void SetConnectionByHandle(Long Handle)

Parameters

Long Handle

要为对象设置的连接的长整型值。

返回值

无人

示例

以下示例展示了如何使用连接句柄为新创建的 autECLXfer 对象设置连接。

```
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 方法会将文件从工作站发送到主机，用于与 autECLXfer 对象关联的连接。

原型

void SendFile(String PCFile, String HostFile, String Options)

Parameters

String PCFile

工作站上文件的名称。

String HostFile

主机上文件的名称。

字符串选项

与主机相关的传输选项。请参阅 使用注意事项 (on page 373) 以获取更多信息。

返回值

无人

使用注意事项

文件传输选项取决于主机。以下是 VM/CMS 主机的一些有效主机选项的列表。

ASCII
CRLF
APPEND

LRECL
RECFM
CLEAR/NOCLEAR
PROGRESS
QUIET

请参阅仿真器编程以获取受支持的主机和关联文件传输选项的列表。

示例

以下示例展示了如何将文件从工作站发送到主机，用于与 autECLXfer 对象关联的连接。

```
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 方法会接收从主机到工作站的文件，用于与 autECLXfer 对象关联的连接。

原型

void ReceiveFile(String PCFile, String HostFile, String Options)

Parameters

String PCFile

工作站上文件的名称。

String HostFile

主机上文件的名称。

字符串选项

与主机相关的传输选项。请参阅 使用注意事项 (on page 375) 以获取更多信息。

返回值

无人

使用注意事项

文件传输选项取决于主机。以下是 VM/CMS 主机的一些有效主机选项的列表：

ASCII
CRLF
APPEND
LRECL
RECFM
CLEAR/NOCLEAR
PROGRESS
QUIET

请参阅仿真器编程手册以获取受支持的主机和关联文件传输选项的列表。

示例

以下示例展示了如何从主机接收文件并将其发送到工作站，用于与 autECLXfer 对象关联的连接。

```
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 集合元素方法会将 ZIEWin 仿真器连接到主机数据流。这与进入 ZIEWin 仿真器**通信**菜单并选择**连接**具有相同的效果。

原型

void StartCommunication()

Parameters

无人

返回值

无人

示例

以下示例展示了如何将 ZIEWin 仿真器会话连接到主机。

```
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 集合元素方法会断开 ZIEWin 仿真器与主机数据流的连接。这与进入 ZIEWin 仿真器**通信**菜单并选择**断开连接**具有相同的效果。

原型

void StopCommunication()

Parameters

无人

返回值

无人

示例

以下示例展示了如何将 ZIEWin 仿真器会话连接到主机。

```
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 事件

以下事件对 autECLXfer 有效：

void NotifyCommEvent(boolean bConnected)

NotifyCommError()

void NotifyCommStop(Long Reason)

NotifyCommEvent

已连接或已断开连接的给定通信链路。

原型

void NotifyCommEvent(boolean bConnected)

Parameters

boolean bConnected

如果通信链路当前已连接，则返回 True；否则返回 False。

示例

有关示例，请参阅事件处理示例 (on page 378)。

NotifyCommError

事件处理出错时会发生此事件。

原型

NotifyCommError()

Parameters

无人

示例

有关示例，请参阅事件处理示例 (on page 378)。

NotifyCommStop

事件处理停止时会发生此事件。

原型

void NotifyCommStop(Long Reason)

Parameters

Long Reason

停止的原因码。目前，这将始终为 0。

事件处理示例

以下是有关如何实现 Xfer 事件的简短示例

```
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 类

autSystem 类用于执行一些编程语言中不存在的实用程序运算。

autSystem 方法

下节介绍了对 autSystem 对象有效的方法。

Long Shell(VARIANT ExeName, VARIANT Parameters, VARIANT WindowStyle)
String Inputnd()

shell

shell 函数会运行任何可执行文件。

原型

LongShell(VARIANT ExeName, VARIANT Parameters, VARIANT WindowStyle)

Parameters

VARIANT ExeName

可执行文件的完整路径和文件名。

VARIANT Parameters

要传递给可执行文件的任何参数。此参数是可选参数。

VARIANT WindowStyle

要显示为可执行文件的初始窗口样式。此参数为可选，可能具有以下值：

1. 正常大小，设置焦点（缺省值）
2. 最小化，设置焦点
3. Maximized
4. 正常大小，不设置焦点
5. 最小化，不设置焦点

返回值

若方法成功，将返回进程标识，或者若方法失败，将返回零。

示例

```
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 方法会向用户显示一个带不显示文本框的弹出式输入框，以便当用户输入数据时，仅显示星号 (*)。

原型

StringInputnd()

Parameters

无人

返回值

在输入框中输入的字符，若未输入任何值则是 ""。

示例

```
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 类

autECLPageSettings 对象控制 Z and I Emulator for Windows 连接的页面设置。它在注册表中的名称是 ZIEWin.autECLPageSettings。此自动化对象也可以在 VB 脚本中使用。

只读属性 autECLPageSettings 已添加到 autECLSession 对象中。请参阅 autECLSession 类 (on page 338) 以获取有关如何使用此属性的信息。

注： autECLSession 对象中的 autECLPageSettings 对象由 autECLSession 对象设置。

以下示例展示了如何在 Visual Basic 中创建和设置 autECLPageSettings 对象。

```
DIM PgSet as Object
Set PgSet = CreateObject("ZIEWin.autECLPageSettings")
PgSet.SetConnectionByName("A")
```

使用注意事项

必须在初始时为创建的对象设置连接。使用 SetConnectionByName 或 SetConnectionByHandle 来初始化对象。只能对连接进行一次设置。在设置连接之后，任何再调用设置连接方法都将引发异常。如果未设置连接，并尝试访问属性或方法，将引发异常。

属性 CPI、LPI 和 FontSize 取决于属性 FaceName。因此，如果在设置 FaceName 之前设置了 CPI、LPI 或 FontSize，并且它们对新的 FaceName 无效，则可能会在连接中重新配置不同的 CPI、LPI 或 FontSize 值。您应该在设置 CPI、LPI 或 FontSize 之前设置 FaceName。否则，每次设置 FaceName 时，查询 CPI、LPI 和 FontSize，并确保它们具有所需的值。

限制

与每个方法关联的连接都必须处于特定状态，方法才能成功。如果未满足这些限制，则会引发相应的例外。

调用 autECLPageSettings 对象的任何属性或方法时，必须满足以下限制。

- 调用此 API 时，主机会话不应在打印。
- **文件 → 页面设置**对话框和**文件 → 打印机设置**对话框不能在使用中。
- 关联的连接不能处于 PDT 方式。

可能对每种特定属性或方法都有附加限制。

连接类型

以下连接类型对 autECLPageSettings 类中的方法有效：

- 3270 显示
- 3270 打印机
- 5250 显示
- VT (ASCII)

如果利用不受支持的连接访问或调用属性或方法，则会引发异常。使用 ConnType 属性确定连接类型。

属性

本节介绍了 autECLPageSettings 对象的属性。

类型	名称	属性
长	CPI	读/写
布尔值	FontCPI	只读
长	LPI	读/写
布尔值	FontLPI	只读
字符串	FaceName	读/写
长	FontSize	读/写
长	MaxLinesPerPage	读/写
长	MaxCharsPerLine	读/写
字符串	姓名	只读
长	句柄	只读
字符串	ConnType	只读
长	CodePage	只读
布尔值	已启动	只读
布尔值	CommStarted	只读
布尔值	APIEnabled	只读
布尔值	就绪	只读

CPI

此属性确定每英寸打印的字符数。这是长整型数据类型，并且已启用读/写。

将此属性设置为预定义的常量 pcFontCPI，以便在“页面设置”中选择“字体 CPI”，或将其设置为某个特定 CPI 值。如果在连接中配置 FontCPI 时查询此属性，则返回实际 CPI 值，而不返回常量 pcFontCPI。

要确定是否在连接中设置了 FontCPI，请使用 FontCPI 属性。

示例

```
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

这将确定是否在连接中设置了“字体 CPI”。FontCPI 是布尔数据类型且为只读。

示例

```
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

此属性确定每英寸打印的行数。这是长整型数据类型，并且已启用读/写。将其设置为预定义的常量 pcFontLPI，以便在“页面设置”中选择“字体 LPI”，或将其设置为某个特定 LPI 值。如果在连接中配置 FontLPI 时查询此属性，则返回实际 LPI 值，而不返回常量 pcFontLPI。要确定是否在连接中设置了 FontLPI，请使用 FontLPI 属性。

示例

```
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

此属性确定是否在连接中设置了“字体 LPI”。FontLPI 是布尔数据类型且为只读。

示例

```
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

这是连接的页面设置的字体名称。FaceName 是字符串数据类型，并且已启用读/写。

示例

```
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

此属性是每页可打印的最大行数。这也称为最大打印行数或 MPL。有效值范围为 1-255。这是长整型数据类型，并且已启用读/写。

示例

```
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

此属性是每行可打印的最大字符数。这也称为最大打印位置或 MPP。有效值范围为 1-255。这是长整型数据类型，并且已启用读/写。

示例

```
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
```

姓名

此属性是设置了 autECLPageSettings 的连接的连接名称字符串。Z and I Emulator for Windows 仅在字符串中返回短字符标识（**A** 到 **Z** 的单字母字符）。给定名称只能有一个打开的 Z and I Emulator for Windows 连接。例如，一次只能有一个打开的连接 **A**。Name 是字符串数据类型且为只读。

示例

```
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
```

句柄

此属性是设置了 autECLPageSettings 对象的连接的句柄。给定句柄只能有一个打开的 Z and I Emulator for Windows 连接。例如，一次只能有一个打开的连接 **A**。Handle 是长整型数据类型且为只读。

示例

```
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

此属性是设置了 autECLPageSettings 的连接类型。这种类型可能会随着时间的推移而更改。ConnType 是字符串数据类型且为只读。

字符串值	**连接类型**
DISP3270	3270 显示
DISP5250	5250 显示
PRNT3270	3270 打印机
PRNT5250	5250 打印机
ASCII	VT 仿真

示例

```
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

此属性是设置了 autECLPageSettings 的连接类型。这种类型可能会随着时间的推移而更改。ConnType 是字符串数据类型且为只读。

示例

```
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
```

已开始

此属性会指示是否已启动仿真器窗口。如果窗口已打开，则值为 TRUE；否则为 FALSE。Started 是布尔数据类型且为只读。

示例

```
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

此属性会指示与主机的连接的状态。如果主机已连接，则值为 TRUE；否则为 FALSE。CommStarted 是布尔数据类型且为只读。

示例

```
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

此属性会指示仿真器是否已启用 API。连接可能已启用或已禁用 API，具体取决于其 API 设置的状态（在 Z and I Emulator for Windows 窗口中，单击**设置 → API**）。如果仿真器已启用 API，则值为 TRUE；否则为 FALSE。APIEnabled 是布尔数据类型且为只读。

示例

```
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
```

Ready

此属性会指示仿真器窗口是否已启动、已启用 API 和已连接。此属性会检查是否有所有三个属性。如果仿真器就绪，则值为 TRUE；否则为 FALSE。Ready 是布尔数据类型且为只读。

示例

```
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 方法

下节介绍了对 autECLPageSettings 对象有效的方法。

void RestoreTextDefaults()
void SetConnectionByName (String Name)
void SetConnectionByHandle (Long Handle)

RestoreTextDefaults

RestoreTextDefaults 方法会恢复连接的“页面设置”对话框中**文本**属性页面的系统缺省值。这相当于在连接的“页面设置”对话框的**文本**属性页面上按**缺省值**按钮。

原型

void RestoreTextDefaults()

Parameters

无人

返回值

无人

示例

以下示例展示了 RestoreTextDefaults 方法。

```
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

SetConnectionByName 方法使用连接名称为新创建的 autECLPageSettings 对象设置连接。此连接名称是短连接标识（从 **A** 到 **Z** 的单个字母字符）。给定名称只能有一个打开的 Z and I Emulator for Windows 连接。例如，一次只能有一个打开的连接 **A**。

注： 如果使用 autECLSession 对象中包含的 autECLPageSettings 对象，请不要调用此方法。

原型

void SetConnectionByName(String Name)

Parameters

String Name

连接的单字符字符串短名称。有效值为 A–Z。

返回值

无人

示例

以下示例展示了如何使用连接名称为新创建的 autECLPageSettings 对象设置连接。

```
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

SetConnectionByHandle 方法使用连接名称为新创建的 autECLPageSettings 对象设置连接。在 Z and I Emulator for Windows 中，此连接句柄是长整型值。给定句柄只能有一个打开的 Z and I Emulator for Windows 连接。例如，一次只能有一个打开的连接 **A**。

注： 如果使用 autECLSession 对象中包含的 autECLPageSettings 对象，请不要调用此方法。

原型

void SetConnectionByHandle(Long Handle)

Parameters

Long Handle

要为对象设置的连接的长整型值。

返回值

无人

示例

以下示例展示了如何使用连接句柄为新创建的 autECLPageSettings 对象设置连接。

```
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 类

autECLPrinterSettings 对象控制 Z and I Emulator for Windows 连接的打印机设置。它在注册表中的名称是 `ZIEWin.autECLPrinterSetting`s。此自动化对象也可以在 VB 脚本中使用。

只读属性 **autECLPrinterSettings** 已添加到 autECLSession 对象中。请参阅 autECLSession 类 (on page 338) 以获取有关如何使用此属性的信息。

注： autECLSession 对象中的 autECLPrinterSettings 对象由 autECLSession 对象设置。

以下示例展示了如何在 Visual Basic 中创建和设置 autECLPrinterSettings 对象。

```
DIM PrSet as Object
Set PrSet = CreateObject("ZIEWin.autECLPrinterSettings")
PrSet.SetConnectionByName("A")
```

使用注意事项

必须在初始时为创建的对象设置连接。使用 SetConnectionByName 或 SetConnectionByHandle 来初始化对象。只能对连接进行一次设置。在设置连接之后，任何再调用设置连接方法都将引发异常。如果未设置连接，并尝试访问属性或方法，将引发异常。

属性 CPI、LPI 和 FontSize 取决于属性 FaceName。因此，如果在设置 FaceName 之前设置了 CPI、LPI 或 FontSize，并且它们对新的 FaceName 无效，则可能会在连接中重新配置不同的 CPI、LPI 或 FontSize 值。您应该在设置 CPI、LPI 或 FontSize 之前设置 FaceName。否则，每次设置 FaceName 时，查询 CPI、LPI 和 FontSize，并确保它们具有所需的值。

限制

与每个方法关联的连接都必须处于特定状态，方法才能成功。如果未满足这些限制，则会引发相应的例外。

调用 autECLPageSettings 对象的任何属性或方法时，必须满足以下限制。

- 调用此 API 时，主机会话不应在打印。
- **文件 → 页面设置**对话框和**文件 → 打印机设置**对话框不能在使用中。

可能对每种特定属性或方法都有附加限制。

属性

本节介绍了 autECLPrinterSettings 对象的属性。

类型	名称	属性
布尔值	PDTMode	只读
字符串	PDTFile	只读
长	PrintMode	只读
字符串	打印机	只读
字符串	PrtToDskAppendFile	只读
字符串	PrtToDskSeparateFile	只读
布尔值	PrompDialogOption	读/写
字符串	姓名	只读
长	句柄	只读
字符串	ConnType	只读

类型	名称	属性
布尔值	CodePage	只读
布尔值	已启动	只读
布尔值	CommStarted	只读
布尔值	APIEnabled	只读
布尔值	就绪	只读

PDTMode

此属性确定连接是否处于 PDT 方式。PDTMode 是布尔数据类型，并且已启用读/写。

示例

```
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

此属性是在连接中配置的 PDT 文件。如果未在连接中配置 PDT 文件，则此属性会提供 null 字符串。否则，此属性提供 PDT 文件的全限定路径名。PDTFile 是字符串数据类型且为只读。

示例

```
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

此属性指示连接的打印方式。PrintMode 是长整型数据类型且为只读。此属性会返回以下四个枚举值之一：

价值	枚举常量的名称	描述
1	pcPrtToDskAppend	**打印到磁盘 - 附加**方式。这意味着在连接的“打印机设置”对话框的**打印机**列表框中选择了**打印到磁盘 → 附加**选项。
2	pcPrtToDskSeparate	**打印到磁盘 - 单独**方式。这意味着在连接的“打印机设置”对话框的**打印机**列表框中选择了**打印到磁盘 → 单独**选项。
3	pcSpecificPrinter	**特定打印机**方式。这意味着在连接的“打印机设置”对话框的**打印机**列表框中选择了其中一台打印机，并且清除了**使用 Windows 缺省打印机**复选框。
4	pcWinDefaultPrinter	**Windows® 缺省打印机**方式。这意味着选中了**使用 Windows 缺省打印机**复选框。

示例

```
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
  ...
```

打印机

此属性是打印机的名称。它包含以下一项：

- 如果连接的 PrintMode 为 pcSpecificPrinter，则为特定打印机的名称。
- 如果连接的 PrintMode 为 pcWinDefaultPrinter，则为 Windows 缺省打印机的名称。
- 如果未在连接中配置打印机，或者如果连接的 PrintMode 是 pcPrtToDskAppend 或 pcPrtToDskSeparate，则为 null 字符串。

Printer 是字符串数据类型且为只读。

值必须采用以下格式：

```
<Printer name> on <Port Name>
```

例如：

- `HP LaserJet 4050 Series PCL 6 on LPT1`

示例

```
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

此属性是**打印到磁盘 - 附加**方式的文件集的名称。此文件称为打印到磁盘 - 附加文件。此属性包含以下一项：

- 连接的**打印到磁盘 - 附加**文件的全限定路径名。
- 如果未在连接中配置**打印到磁盘 - 附加**文件，则为 null 字符串。

PrtToDskAppendFile 是字符串数据类型且为只读。

示例

```
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

此属性是**打印到磁盘 - 单独**方式的文件集的名称。此文件称为**打印到磁盘 - 单独**文件。此属性包含以下一项：

- 连接的**打印到磁盘 - 单独**文件的全限定路径名。
- 如果未在连接中配置**打印到磁盘 - 单独**文件，则为 null 字符串。

PrtToDskSeparateFile 是字符串数据类型且为只读。

示例

```
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

此属性会指示是否设置了在打印前显示“打印机设置”对话框的选项。PromptDialogOption 是布尔数据类型且为只读。

示例

```
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
```

姓名

此属性是设置了 autECLPrinterSettings 的连接的连接名称字符串。Z and I Emulator for Windows 仅在字符串中返回短字符标识（**A** 到 **Z** 的单字母字符）。给定名称只能有一个打开的 Z and I Emulator for Windows 连接。例如，一次只能有一个打开的连接 **A**。Name 是字符串数据类型且为只读。

示例

```
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
```

句柄

此属性是设置了 autECLPrinterSettings 的连接的句柄。给定句柄只能有一个打开的 Z and I Emulator for Windows 连接。例如，一次只能有一个打开的连接 **A**。Handle 是长整型数据类型且为只读。

示例

```
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

此属性是设置了 autECLPrinterSettings 的连接类型。这种类型可能会随着时间的推移而更改。ConnType 是字符串数据类型且为只读。

字符串值	**连接类型**
DISP3270	3270 显示
DISP5250	5250 显示
PRNT3270	3270 打印机
PRNT5250	5250 打印机
ASCII	VT 仿真

示例

```
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

此属性是设置了 autECLPrinterSettings 的连接的代码页。此代码页可能会随着时间的推移而更改。CodePage 是长整型数据类型且为只读。

示例

```
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
```

已开始

此属性会指示是否已启动仿真器窗口。如果窗口已打开，则值为 TRUE；否则为 FALSE。Started 是布尔数据类型且为只读。

示例

```
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

此属性会指示与主机的连接的状态。如果主机已连接，则值为 TRUE；否则为 FALSE。CommStarted 是布尔数据类型且为只读。

示例

```
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

此属性会指示仿真器是否已启用 API。连接已启用或已禁用 API，具体取决于其 API 设置的状态（在 Z and I Emulator for Windows 窗口中，单击**设置 → API**）。

如果仿真器已启用 API，则值为 TRUE；否则为 FALSE。APIEnabled 是布尔数据类型且为只读。

示例

```
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
```

Ready

此属性会指示仿真器窗口是否已启动、已启用 API 和已连接。此属性会检查是否有所有三个属性。如果仿真器就绪，则值为 TRUE；否则为 FALSE。Ready 是布尔数据类型且为只读。

示例

```
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 方法

下节介绍了对 autECLPrinterSettings 对象有效的方法。

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 方法会在 PDT 方式下设置与给定 PDT 文件的连接，或者在非 PDT 方式（也称为 *GDI* 方式）下设置连接。

Restriction

如果在 bPDTMode 设置为 FALSE 的情况下调用此方法，则必须已将关联连接的 PrintMode 设置为 SpecificPrinter 或 WinDefaultPrinter。

原型

void SetPDTMode(Boolean bPDTMode, [optional] String PDTFile)

Parameters

Boolean bPDTMode

有效值如下：

- **TRUE**，将连接设置为 PDT 方式。
- **FALSE**，将连接设置为非 PDT 方式（GDI 方式）。

String PDTFile

此可选参数包含 PDT 文件名。

仅当 bPDTMode 为 TRUE 时才使用此参数。如果未指定此参数且将 bPDTMode 设置为 TRUE，则使用连接中配置的 PDT 文件。如果尚未在连接中配置任何 PDT 文件，则此方法将失败并出现异常。

如果 bPDTMode 为 FALSE，则忽略此参数。

有效值如下：

- 不带路径的文件名

  使用 Z and I Emulator for Windows 安装路径中 PDFPDT 子文件夹中的 PDTFile。
- 文件的全限定路径名

  如果 PDTFile 不存在，则此方法将失败并出现异常。

返回值

无人

示例

```
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

此方法会将连接的 PrintMode 设置为**打印到磁盘 - 附加**方式。它还会设置适合此方式的文件。

注:

1. 要设置此文件的文件夹必须具有写入访问权。否则，此方法将失败并出现异常。
2. 关联的连接必须处于 PDT 方式。

原型

void SetPrtToDskAppend([optional] String FileName)

Parameters

String FileName

此可选参数包含**打印到磁盘 - 附加**文件的名称。

如果此文件存在，则使用此文件。否则，将在打印完成时创建此文件。

有效值如下：

- 文件名，不带路径

  用户类应用程序数据目录路径将用于查找文件。
- 文件的全限定路径名

  该目录必须存在于路径中，否则该方法将失败并出现异常。文件不需要存在于路径中。

如果未指定此参数，则使用连接中为此 PrintMode 配置的文件。如果尚未在连接中配置任何文件，则此方法将失败并出现异常。

返回值

无人

示例

```
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

此方法会将连接的 PrintMode 设置为**打印到磁盘 - 单独**方式。它还会设置适合此方式的文件。

注:

1. 要设置此文件的文件夹必须具有写入访问权。否则，此方法将失败并出现异常。
2. 关联的连接必须处于 PDT 方式。

原型

void SetPrtToDskSeparate([optional] String FileName)

Parameters

String FileName

此可选参数包含**打印到磁盘 - 单独**文件的名称。

如果未指定此参数，则使用连接中为此 PrintMode 配置的文件。

可能的值包括:

- **NULL**（缺省值）

  使用连接中当前为此 PrintMode 配置的文件。如果尚未在连接中配置文件，则此方法将失败并出现异常。
- 文件名，不带路径

  用户类应用程序数据目录路径将用于查找文件。
- 文件的全限定路径名

  该目录必须存在于路径中，否则该方法将失败并出现异常。文件不需要存在于路径中。

注: 文件名不能包含扩展名。如果包含扩展名，则该方法将失败并出现异常。

返回值

无人

示例

```
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

此方法会使用由 Printer 参数指定的打印机将连接的 PrintMode 设置为特定打印机方式。

原型

void SetSpecificPrinter(String Printer)

Parameters

String Printer

包含打印机的名称。如果打印机不存在，则此方法将失败并出现异常。
值必须采用以下格式：

```
<Printer name> on <Port Name>
```

例如：

- `HP LaserJet 4050 Series PCL 6 on LPT1`

返回值

无人

示例

```
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

此方法会将连接的 PrintMode 设置为 Windows 缺省打印机方式（连接将使用 Windows 缺省打印机）。如果未配置 Windows 缺省打印机，则此方法将失败并出现异常。

原型

void SetWinDefaultPrinter()

Parameters

无人

返回值

无人

示例

```
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

SetConnectionByName 方法使用连接名称为新创建的 autECLPrinterSettings 对象设置连接。在 Z and I Emulator for Windows 中，此连接名称是短连接标识（从 **A** 到 **Z** 的单个字母字符）。给定名称只能有一个打开的 Z and I Emulator for Windows 连接。例如，一次只能有一个打开的连接 **A**。

注： 如果使用 autECLSession 对象中包含的 autECLPrinterSettings 对象，请不要调用此项。

原型

void SetConnectionByName(String Name)

Parameters

String Name

连接的单字符字符串短名称。有效值为 A–Z。

返回值

无人

示例

以下示例展示了如何使用连接名称为新创建的 autECLPrinterSettings 对象设置连接。

```
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

SetConnectionByHandle 方法使用连接名称为新创建的 autECLPrinterSettings 对象设置连接。在 Z and I Emulator for Windows 中，此连接句柄是长整型值。

给定句柄只能有一个打开的 Z and I Emulator for Windows 连接。例如，一次只能有一个打开的连接 **A**。

注： 如果使用 autECLSession 对象中包含的 autECLPrinterSettings 对象，请不要调用此方法。

原型

void SetConnectionByHandle(Long Handle)

Parameters

Long Handle

要为对象设置的连接的长整型值。

返回值

无人

示例

以下示例展示了如何使用连接句柄为新创建的 autECLPrinterSettings 对象设置连接。

```
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
```

支持自动化对象的主互操作组合件

HCL Z and I Emulator for Windows 公开的自动对象可以由采用针对 .NET 框架的任何语言编写的应用程序使用。托管 .NET 应用程序可以使用包装自动对象的主互操作组合件 (PIA) 来对 Z and I Emulator for Windows 进行编程。互操作组合件是托管 (.NET) 应用程序使用 COM 兼容对象的机制。互操作组合件包含绑定和元数据信息，这使 .NET 框架 (CLR) 能够装入或编组 COM 对象，并将其包装成 .NET 应用程序。PIA 包含由这些 COM 类型的发布程序定义的 COM 类型的官方描述。PIA 始终由原始 COM 类型的发布程序进行数字签名。

.NET 应用程序可以通过两种方式引用组合件。

- 如果它是一个简单的应用程序或唯一使用组合件的应用程序，Microsoft 建议将组合件复制到与应用程序相同的目录中。
- 如果有多个应用程序引用组合件，则可以在全局组合件高速缓存 (GAC) 中安装它们，并让所有解决方案引用 GAC 中的组合件。

对互操作组合件公开的类型进行编程的模型与 COM 非常相似。任何 .NET 语言都可以使用该语言的语法，来访问 COM 对象公开的方法、属性和事件。在 Z and I Emulator for Windows 安装映像的 \samples 目录中提供了使用 C# 编写的样本应用程序 (ECLSamps.net)。该样本演示了各种互操作组合件类型的简单用法。

对于使用 Z and I Emulator for Windows 自动化对象并已使用转换助手向导迁移到 Visual Basic .NET 的 Visual Basic 6.0 项目，只需使用相应的 Z and I Emulator for Windows 互操作引用（来自 \Interops 目录）替换转换助手向导隐式生成的引用，然后重新编译。替换引用的方法是删除转换助手生成的所有引用，并使用 Visual Studio .NET 添加 .NET 互操作引用。如果已在 GAC 中注册组合件并希望使用它们，请添加引用，并将 Z and I Emulator for Windows 互操作引用 的Copy Local 属性设置为 **False**。

Z and I Emulator for Windows 仿真器自动化对象的 PIA 安装在 Z and I Emulator for Windows 安装映像的 \Interops 目录中。如果 Z and I Emulator for Windows 产品安装程序检测到存在 .NET 框架，它将为您提供在 GAC 中注册这些类型的附加选项。在 GAC 中安装组合件时，也会将 PIA 放在注册表中相应类型库的注册表项下。

表 2: Z and I Emulator for Windows 自动化对象的主互操作组合件 (on page 408) 列出为 Z and I Emulator for Windows 自动化对象提供的 PIA

表 2. Z and I Emulator for Windows 自动化对象的主互操作组合件

自动化对象	互操作组合件依赖关系
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
autECLXfer	Interop.AutXferTypeLibrary.dll
autSystem	Interop.AutSystemTypeLibrary.dll

第 4 章. Host Access Class Library for Java

主机访问类库 (HACL) Java 类将 Z and I Emulator for Windows HACL 函数公开给 Java 编程环境。这允许创建使用 HACL 类中提供的函数的 Java Applet 和应用程序。

HACL Java HTML 文件可以在与 Z and I Emulator for Windows 产品文档一起提供的 Docs_Admin_Aids zip 文件夹中找到，路径如下：*ZIEWin_3.0_Docs_Admin_Aids.zip\publications\zh_CN\doc\hacl* 目录。

第 5 章. 故障诊断

可以使用以下自助信息资源和工具来帮助进行问题故障诊断。

- 请参阅产品的发行信息，以获取已知问题、变通方法和故障诊断信息。
- 检查是否存在用于解决问题的下载或修订。
- 搜索提供的知识库，以查看是否已经说明问题的解决方案。
- 如果仍需要帮助，请联系 HCL 软件支持机构，并报告您的问题。

HCL Z and I Emulator for Windows .NET 互操作组合件无法触发会话 OIA 通知

问题

注册 OIA 事件通知的 .NET 应用程序不会收到有关这些事件的通知。Visual Studio IntelliSense 功能也没有显示对应 COM 类型库的几种方法。

原因

.NET 互操作组合件是使用 Microsoft SDK 附带的工具 TlbImp.exe 从相应的 COM 类型库派生的。Type Library Importer 会将 COM 类型库中的类型定义转换为公共语言运行时组合件中的等效定义。但是，运行时编组程序无法编组所有数据类型。因此，在生成的公共语言运行时组合件中找不到某些 COM 类型库定义。

分辨率

这是 TlbImp.exe 的限制。

附录 A. Sendkeys 助记符关键字

表 3: Sendkeys 方法的助记符关键字 (on page 411) 包含 Sendkeys 方法的助记符关键字。

表 3. Sendkeys 方法的助记符关键字

关键字	描述
[backtab]	退格制表
[clear]	清除屏幕
[delete]	删除
[enter]	Enter
[eraseeof]	擦除字段结束
[help]	帮助
[insert]	插入
[jump]	跳转
[left]	向左箭头
[newline]	新行
[space]	空格键
[print]	打印
[reset]	重置
[tab]	Tab 键
[up]	向上箭头
[Down]	向下箭头
[capslock]	CapsLock
[right]	向右箭头
[home]	主页
[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
[pf14]	PF14

表 3. Sendkeys 方法的助记符关键字 (续)

关键字	描述
[pf15]	PF15
[pf16]	PF16
[pf17]	PF17
[pf18]	PF18
[pf19]	PF19
[pf20]	PF20
[pf21]	PF21
[pf22]	PF22
[pf23]	PF23
[pf24]	PF24
[eof]	文件结束
[scrlock]	滚动锁定
[numlock]	Num Lock
[pageup]	Page Up 键
[pagedn]	Page Down 键
[pa1]	PA 1
[pa2]	PA 2
[pa3]	PA 3
[test]	测试
[worddel]	字词删除
[fldext]	字段退出
[erinp]	擦除输入
[sysreq]	系统请求
[instog]	插入切换
[crsel]	光标选择
[fastleft]	快速光标左移
[attn]	注意
[devcance]	设备取消
[printps]	打印表示空间
[fastup]	快速光标上移
[fastdown]	快速光标下移
[hex]	十六进制
[fastright]	快速光标右移
[revvideo]	反相显示
[underscr]	下划线
[rstvideo]	重置反相显示
[red]	红色

表 3. Sendkeys 方法的助记符关键字 (续)

关键字	描述
[pink]	粉红色
[green]	绿色
[yellow]	黄色
[blue]	蓝色
[turq]	Turquoise
[white]	白色
[rstcolor]	重置主机颜色
[printpc]	打印 (PC)
[wordright]	正向字词制表符
[wordleft]	反向字词制表符
[field-]	字段 -
[field+]	字段 +
[rcdbacksp]	记录退格键
[printhost]	主机上的打印表示空间
[dup]	Dup
[fieldmark]	字段标记
[dispsosi]	显示 SO/SI
[gensosi]	生成 SO/SI
[dispattr]	显示属性
[fwdchar]	正向字符
[splitbar]	拆分竖线
[altcsr]	备用光标
[backspace]	Backspace
[空值]	Null

附录 B. ECL 平面 - 格式和内容

本附录介绍 ECL 表示空间模型中不同数据平面的格式和内容。每个平面都表示主机表示空间的一个不同方面，例如其字符内容、颜色规范、字段属性等。ECL::GetScreen 方法和其他方法从不同的表示空间平面返回数据。

每个平面都包含每个主机表示空间字符位置的一个字节。以下各节将根据逻辑内容和数据格式介绍每个平面。平面类型在 ECLPS.HPP 头文件中枚举。

TextPlane

文本平面表示表示空间的可视字符。在文本平面中显示非显示字段。文本平面的每个元素的字节值都对应于显示的字符的 ASCII 值。文本平面不包含任何二进制零 (null) 字符值。表示空间中的任何 null 字符（例如，null 填充的输入字段）表示ASCII 空白 (0x20) 字符。

FieldPlane

字段平面描述表示空间里字段的位置和它们的属性。此平面仅对字段格式的表示空间有意义。（例如，VT 连接未格式化）。

此平面是字段属性值的稀疏数组。此平面中的所有值均为二进制零，但表示空间中存在字段属性字符的情况除外。在这些位置，值是从该位置开始的字段的属性。字段的长度是字段属性位置与表示空间中下一个字段属性之间的线性距离，不包括属性位置本身。

字段属性位置的值如下表所示。

注： 连接类型不同，属性值也不同。

表 4. 3270 字段属性

位位置（0 是最低有效位）	意义
7	始终为“1”
6	始终为“1”
5	**0** 不受保护 **1** 受保护
4	**0** 字母数字数据 **1** 仅数字数据

表 4. 3270 字段属性 (续)

位位置（0 是最低有效位）	意义
3、2	**0、0** 正常亮度，非光笔可检测 **0、1** 正常亮度，光笔可检测 **1、0** 高亮度，光笔可检测 **1、1** 不显示，非光笔可检测
1	保留
0	**0** 字段未经过修改 **1** 不受保护的字段已经过修改

表 5. 5250 字段属性

位位置（0 是最低有效位）	意义
7	始终为“1”
6	**0** 不显示 **1** Display
5	**0** 不受保护 **1** 受保护
4	**0** 正常亮度 **1** 高密度
3、2、1	**0、0、0** 字母数字数据

表 5. 5250 字段属性 (续)

位位置（0 是最低有效位）	意义
	0、0、1 仅限字母 **0、1、0** 数字换档 **0、1、1** 数字数据以及数字特殊格式 **1、0、1** 仅限数字 **1、1、0** 仅限磁条读取设备数据 **1、1、1** 仅限于有符号数字
0	**0** 字段未经过修改 **1** 不受保护的字段已经过修改

表 6: 掩码值 (on page 416) 定义各种掩码值：

表 6. 掩码值

助记符	掩码	描述
FATTR_MDT	0x01	已修改字段
FATTR_PEN_MASK	0x0C	光笔可检测字段
FATTR_BRIGHT	0x08	强化字段
FATTR_DISPLAY	0x0C	可见字段
FATTR_ALPHA	0x10	字母数字字段
FATTR_NUMERIC	0x10	仅数字字段
FATTR_PROTECTED	0x20	受保护字段
FATTR_PRESENT	0x80	字段属性存在
FATTR_52_BRIGHT	0x10	5250 强化字段
FATTR_52_DISP	0x40	5250 可见字段

ColorPlane

色彩平面包含表示空间里每个字符的色彩信息。每个字符的前景和背景颜色都表示为在主机数据流中指定的颜色。仿真器窗口的任何颜色显示映射都不会修改颜色平面中的颜色。颜色平面的每个字节都包含以下颜色信息。

表 7. 颜色平面信息

位位置（0 是最低有效位）	意义
7 - 4	**背景字符颜色** **0x0** 空格 **0x1** 蓝色 **0x2** 绿色 **0x3** 青色 **0x4** 红色 **0x5** 品红色 **0x6** 棕色 (3270)，黄色 (5250) **0x7** 白色
3-0	**前景字符颜色** **0x0** 空格 **0x1** 蓝色 **0x2** 绿色 **0x3** 青色

表 7. 颜色平面信息 (续)

位位置（0 是最低有效位）	意义
	0x4 红色 **0x5** 品红色 **0x6** 棕色 (3270)，黄色 (5250) **0x7** 白色（正常亮度） **0x8** 灰色 **0x9** 淡蓝色 **0xA** 淡绿色 **0xB** 浅青色 **0xC** 浅红色 **0xD** 浅品红色 **0xE** 黄色 **0xF** 白色（高亮度）

ExfieldPlane

此平面包含扩展字符属性数据。

此平面是扩展字符属性值的稀疏数组。数组中的所有值均为二进制零，但主机为其指定了扩展字符属性的表示空间中的字符除外。扩展字符属性值的含义如下所示。

表 8. 3270 扩展字符属性

位位置（0 是最低有效位）	意义
7、6	**字符突出显示** **0、0** 正常 **0、1** 闪烁 **1、0** 反向显示 **1、1** 下划线
5、4、3	**字符颜色** **0、0、0** 缺省值 **0、0、1** 蓝色 **0、1、0** 红色 **0、1、1** 粉红色 **1、0、0** 绿色 **1、0、1** Turquoise **1、1、0** 黄色 **1、1、1** 白色
2、1	**字符属性** **00** 缺省值 **11** 双字节字符

表 8. 3270 扩展字符属性 (续)

位位置（0 是最低有效位）	意义
0	保留

表 9. 5250 扩展字符属性

位位置（0 是最低有效位）	意义
7	**0** 正常图像 **1** 反色图像
6	**0** 无下划线 **1** 下划线
5	**0** 不闪烁 **1** 闪烁
4	**0** 无列分隔符 **1** 列分隔符
3、2、1、0	保留

附录 C. 声明

本信息是为在美国提供的产品和服务编写的。HCL 可能在其他国家或地区不提供本资料中讨论的产品、服务或功能。请咨询您当地的 HCL代表，以获取有关您所在区域当前可获得的产品和服务的信息。任何对 HCL产品、程序或服务的提及并非意在明示或默示只能使用 HCL 产品、程序或服务。只要不侵犯 HCL 的知识产权，就可以改用任何具有同等功能的产品、程序或服务。但是，评估和验证任何非 HCL产品、程序或服务，则由用户自行负责。

HCL 可能已拥有或正在申请与本资料内容有关的各项专利。提供本文档并未授予用户使用这些专利的任何许可证。您可以用书面方式将许可查询寄往：

HCL
330 Potrero Ave.
Sunnyvale, CA 94085
USA
注意：Office of the General Counsel

HCL TECHNOLOGIES LTD.“按现状”提供本出版物，不附有任何种类的（无论是明示的还是默示的）保证，包括但不限于默示的有关不侵权、适销和适用于某特定用途的保证。某些管辖区域在某些交易中不允许免除明示或暗含的保证，因此本条款可能不适用于您。

本资料可能包含技术方面不够准确的地方或印刷错误。此处的信息将定期更改；这些更改将编入本资料的新版本中。HCL 可随时对本资料中描述的产品和/或程序进行改进和/或更改，恕不另行通知。

本资料中对非 HCL 文档或非 HCL Web 站点的任何提及只是为方便起见才提供，不以任何方式充当对那些文档或 Web 站点的保证。那些 Web 站点中的资料不是本 HCL 产品资料的一部分，使用那些 Web 站点带来的风险将由您自行承担。

HCL 可以按它认为适当的任何方式使用或分发您所提供的任何信息，而无须对您承担任何责任。

本程序的被许可方如果要了解有关程序的信息以达到如下目的：(i) 允许在独立创建的程序和其他程序（包括本程序）之间进行信息交换，以及 (ii) 允许对已经交换的信息进行相互使用，请与下列地址联系：

HCL
330 Potrero Ave.
Sunnyvale, CA 94085
USA
注意：Office of the General Counsel

只要遵守适当的条件和条款，包括某些情形下支付一定数额的费用，都可获得这方面的信息。

本资料中描述的许可程序及其所有可用的许可资料均由 HCL 依据 HCL 客户协议、HCL 国际软件许可协议或任何同等协议中的条款提供。

此处讨论的性能数据是在特定操作条件下得出的。实际结果可能会有差异。

涉及非 HCL 产品的信息可从这些产品的供应商、其出版说明或其他可公开获得的资料中获取。HCL 没有对这些产品进行测试，也无法确认其性能的精确性、兼容性或任何其他关于非 HCL 产品的声明。有关非 HCL 产品性能的问题应当向这些产品的供应商提出。

本资料包括日常业务运作中使用的数据和报告的示例。为了尽可能完整地说明这些示例，示例中可能会包括个人、公司、品牌和产品的名称。所有这些名称都是虚构的，若实际人员或业务企业与此相似，纯属巧合。

商标