Host Access Class Library
Host Access Class Library
Before using this information and the product it supports, be sure to read the general information under Appendix C, "Notices," on page 407.
## Contents

### Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Getter . . .</td>
<td>ix</td>
</tr>
<tr>
<td>GetHandle . .</td>
<td>xi</td>
</tr>
<tr>
<td>ECLConnList Class . .</td>
<td>31</td>
</tr>
<tr>
<td>ECLConnList Methods . .</td>
<td>32</td>
</tr>
<tr>
<td>ECLConnList Constructor . .</td>
<td>33</td>
</tr>
<tr>
<td>ECLConnList Destructor . .</td>
<td>33</td>
</tr>
<tr>
<td>GetFirstConnection . .</td>
<td>34</td>
</tr>
<tr>
<td>GetNextConnection . .</td>
<td>35</td>
</tr>
<tr>
<td>FindConnection . .</td>
<td>35</td>
</tr>
<tr>
<td>GetCount . .</td>
<td>36</td>
</tr>
<tr>
<td>Refresh . .</td>
<td>37</td>
</tr>
<tr>
<td>ECLConnMgr Class . .</td>
<td>38</td>
</tr>
<tr>
<td>ECLConnMgr Methods . .</td>
<td>38</td>
</tr>
<tr>
<td>ECLConnMgr Constructor . .</td>
<td>38</td>
</tr>
<tr>
<td>ECLConnMgr Destructor . .</td>
<td>38</td>
</tr>
<tr>
<td>GetConnList . .</td>
<td>39</td>
</tr>
<tr>
<td>StartConnection . .</td>
<td>40</td>
</tr>
<tr>
<td>StopConnection . .</td>
<td>41</td>
</tr>
<tr>
<td>RegisterStartEvent . .</td>
<td>42</td>
</tr>
<tr>
<td>UnregisterStartEvent . .</td>
<td>43</td>
</tr>
<tr>
<td>ECLCommNotifyClass . .</td>
<td>44</td>
</tr>
<tr>
<td>ECLCommNotifyMethods . .</td>
<td>46</td>
</tr>
<tr>
<td>NotifyEvent . .</td>
<td>47</td>
</tr>
<tr>
<td>NotifyError . .</td>
<td>48</td>
</tr>
<tr>
<td>NotifyStop . .</td>
<td>48</td>
</tr>
<tr>
<td>ECLErrClass . .</td>
<td>48</td>
</tr>
<tr>
<td>ECLErrMethods . .</td>
<td>48</td>
</tr>
<tr>
<td>GetMsgNumber . .</td>
<td>48</td>
</tr>
<tr>
<td>GetReasonCode . .</td>
<td>49</td>
</tr>
<tr>
<td>GetMsgText . .</td>
<td>50</td>
</tr>
<tr>
<td>ECLFieldClass . .</td>
<td>50</td>
</tr>
<tr>
<td>ECLFieldMethods . .</td>
<td>51</td>
</tr>
<tr>
<td>ECLFieldMethods . .</td>
<td>53</td>
</tr>
<tr>
<td>GetStart . .</td>
<td>53</td>
</tr>
<tr>
<td>GetStartRow . .</td>
<td>54</td>
</tr>
<tr>
<td>GetStartCol . .</td>
<td>55</td>
</tr>
<tr>
<td>GetEnd . .</td>
<td>56</td>
</tr>
<tr>
<td>GetEndRow . .</td>
<td>57</td>
</tr>
<tr>
<td>GetEndCol . .</td>
<td>58</td>
</tr>
<tr>
<td>GetLength . .</td>
<td>59</td>
</tr>
<tr>
<td>GetScreen . .</td>
<td>59</td>
</tr>
</tbody>
</table>

### Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tables . . .</td>
<td>xi</td>
</tr>
<tr>
<td>About This Book . . .</td>
<td>xiii</td>
</tr>
<tr>
<td>Who Should Read This Book . . .</td>
<td>xiii</td>
</tr>
<tr>
<td>How to Use This Book . . .</td>
<td>xiii</td>
</tr>
<tr>
<td>Where to Find More Information . . .</td>
<td>xiv</td>
</tr>
<tr>
<td>What’s New in This Release . . .</td>
<td>xiv</td>
</tr>
</tbody>
</table>

### Chapter 1. Introduction

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++ Objects</td>
<td>1</td>
</tr>
<tr>
<td>HACL C++ 1390/1399 Code Page Support</td>
<td>2</td>
</tr>
<tr>
<td>Java Objects</td>
<td>2</td>
</tr>
<tr>
<td>Automation Objects</td>
<td>2</td>
</tr>
<tr>
<td>LotusScript Extension</td>
<td>2</td>
</tr>
<tr>
<td>ECL Concepts</td>
<td>2</td>
</tr>
<tr>
<td>Connections, Handles and Names</td>
<td>2</td>
</tr>
<tr>
<td>Sessions</td>
<td>3</td>
</tr>
<tr>
<td>ECL Container Objects</td>
<td>4</td>
</tr>
<tr>
<td>ECL List Objects</td>
<td>4</td>
</tr>
<tr>
<td>Events</td>
<td>4</td>
</tr>
<tr>
<td>Error Handling</td>
<td>5</td>
</tr>
<tr>
<td>Addressing (Rows, Columns, Positions)</td>
<td>5</td>
</tr>
<tr>
<td>Migrating from EHLAPI</td>
<td>6</td>
</tr>
<tr>
<td>Execution/Language Interface</td>
<td>6</td>
</tr>
<tr>
<td>Features</td>
<td>6</td>
</tr>
<tr>
<td>Session IDs</td>
<td>7</td>
</tr>
<tr>
<td>Presentation Space Models</td>
<td>8</td>
</tr>
<tr>
<td>SendKey Interface</td>
<td>8</td>
</tr>
<tr>
<td>Events</td>
<td>8</td>
</tr>
<tr>
<td>PS Connect/Disconnect and Multithreading</td>
<td>9</td>
</tr>
</tbody>
</table>

### Chapter 2. Host Access Class Library

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Building C++ ECL Programs</td>
<td>13</td>
</tr>
<tr>
<td>IBM Visual Age C++</td>
<td>14</td>
</tr>
<tr>
<td>Microsoft Visual C++</td>
<td>15</td>
</tr>
<tr>
<td>ECLBase Class</td>
<td>15</td>
</tr>
<tr>
<td>Derivation</td>
<td>15</td>
</tr>
<tr>
<td>ECLBase Methods</td>
<td>16</td>
</tr>
<tr>
<td>GetVersion</td>
<td>16</td>
</tr>
<tr>
<td>ConvertHandle2ShortName</td>
<td>16</td>
</tr>
<tr>
<td>ConvertShortName2Handle</td>
<td>17</td>
</tr>
<tr>
<td>ConvertTypeToString</td>
<td>17</td>
</tr>
<tr>
<td>ConvertPos</td>
<td>18</td>
</tr>
<tr>
<td>ECLConnection Class</td>
<td>19</td>
</tr>
<tr>
<td>Derivation</td>
<td>19</td>
</tr>
<tr>
<td>ECLConnection Methods</td>
<td>20</td>
</tr>
<tr>
<td>ECLConnection Constructor</td>
<td>20</td>
</tr>
<tr>
<td>ECLConnection Destructor</td>
<td>21</td>
</tr>
<tr>
<td>GetCodePage</td>
<td>21</td>
</tr>
<tr>
<td>GetHandle</td>
<td>22</td>
</tr>
<tr>
<td>GetConnType</td>
<td>23</td>
</tr>
<tr>
<td>GetName</td>
<td>24</td>
</tr>
<tr>
<td>GetEncryptionLevel</td>
<td>25</td>
</tr>
<tr>
<td>IsStarted</td>
<td>26</td>
</tr>
<tr>
<td>IsCommStarted</td>
<td>26</td>
</tr>
<tr>
<td>IsAPIEnabled</td>
<td>27</td>
</tr>
<tr>
<td>IsReady</td>
<td>28</td>
</tr>
<tr>
<td>IsDBCSHost</td>
<td>28</td>
</tr>
<tr>
<td>StartCommunication</td>
<td>29</td>
</tr>
<tr>
<td>StopCommunication</td>
<td>29</td>
</tr>
<tr>
<td>RegisterCommEvent</td>
<td>30</td>
</tr>
<tr>
<td>UnregisterCommEvent</td>
<td>31</td>
</tr>
</tbody>
</table>
ECLRecoNotify Class  133
Derivation  133
ECLRecoNotify Methods  133
ECLRecoNotify Constructor  133
ECLRecoNotify Destructor  133
NotifyEvent  134
NotifyStop  134
NotifyError  134
ECLScreenDesc Class  135
Derivation  135
ECLScreenDesc Methods  135
ECLScreenDesc Constructor  135
ECLScreenDesc Destructor  136
AddAttrib  136
AddCursorPos  137
AddNumFields  138
AddNumInputFields  138
AddOIAInhibitStatus  139
AddString  139
AddStringInRect  140
Clear  141
ECLScreenReco Class  141
Derivation  142
ECLScreenReco Methods  142
ECLScreenReco Constructor  143
ECLScreenReco Destructor  143
AddPS  143
IsMatch  144
RegisterScreen  144
RemovePS  144
UnregisterScreen  145
ECLSession Class  145
Derivation  145
Properties  145
Usage Notes  145
ECLSession Methods  146
ECLSession Constructor  146
ECLSession Destructor  147
GetPS  147
GetOIA  148
GetXfer  148
GetWinMetrics  149
GetPageSettings  150
GetPrinterSettings  150
RegisterUpdateEvent  151
UnregisterUpdateEvent  151
ECLStartNotify Class  151
Derivation  152
Example  152
ECLStartNotify Methods  154
NotifyEvent  154
NotifyError  154
NotifyStop  155
ECLUpdateNotify Class  155
ECLWinMetrics Class  155
Derivation  155
Properties  156
Usage Notes  156
ECLWinMetrics Methods  156
ECLWinMetrics Constructor  156
ECLWinMetrics Destructor  157
GetWindowTitle  158
SetWindowTitle  158
GetXpos  159
SetXpos  160
GetYpos  161
SetYpos  162
GetWidth  162
SetWidth  163
GetHeight  164
SetHeight  165
GetWindowRect  165
SetWindowRect  166
IsVisible  167
SetVisible  167
IsActive  168
SetActive  168
IsMinimized  169
SetMaximized  169
IsMaximized  170
SetMaximized  170
IsMaximized  170
IsRestored  171
SetRestored  172
ECLXfer Class  172
Derivation  172
Properties  172
Usage Notes  172
ECLXfer Methods  173
ECLXfer Constructor  173
ECLXfer Destructor  173
SendFile  174
ReceiveFile  175
ECLPageSettings Class  177
Derivation  177
Properties  177
Restrictions  177
Usage Notes  177
ECLPageSettings Methods  178
Connection types  178
ECLPageSettings Constructor  178
GetCPI  179
SetCPI  180
IsFontCPI  180
GetLPI  181
SetLPI  181
IsFontLPI  182
SetFontFaceName  182
GetFontFaceName  183
SetFontSize  183
GetFontSize  183
SetMaxLinesPerPage  184
GetMaxLinesPerPage  184
SetMaxCharsPerLine  185
GetMaxCharsPerLine  185
RestoreDefaults  186
ECLPrinterSettings Class  186
Derivation  186
Properties  186
Restrictions  186
Usage Notes  187
ECLPrinterSettings Methods  187
ECLPrinterSettings Constructor  187

Contents  V
Chapter 3. Host Access Class Library

Host Access Class Library Objects .......................... 199

autoSystem Class ............................................ 200
autoECLConnList Class ....................................... 201
  Properties .................................................. 201
autoECLConnList Methods .................................... 204
  Collection Element Methods ............................... 204
  Refresh .................................................... 204
  FindConnectionElementByHandle ......................... 205
  FindConnectionByName ..................................... 205
  StartCommunication ....................................... 206
  StopCommunication ........................................ 206
autoECLConnMgr Class ......................................... 207
  Properties .................................................. 207
autoECLConnMgr Methods ..................................... 207
  RegisterStartEvent ....................................... 207
  UnregisterStartEvent ..................................... 207
  StartConnection .......................................... 208
  StopConnection .......................................... 209
autoECLConnMgr Events ...................................... 210
  NotifyStartEvent ......................................... 210
  NotifyStartError ......................................... 210
  NotifyStartStop .......................................... 210
  Event Processing Example ............................... 210
autoECLFieldList Class ...................................... 211
  Properties .................................................. 211
autoECLFieldList Methods .................................. 216
  Collection Element Methods ............................... 216
  Refresh .................................................... 216
  FindFieldByRowCol ....................................... 217
  FindFieldByText ......................................... 217
  GetText .................................................... 218
  SetText .................................................... 219
autoECLOIA Class ............................................ 219
  Properties .................................................. 220
autoECLOIA Methods ......................................... 227
  RegisterOIAEvent ......................................... 227
  UnregisterOIAEvent ....................................... 227
  SetConnectionByName ...................................... 227
  SetConnectionByHandle ................................... 228
  StartCommunication ....................................... 229
  StopCommunication ....................................... 229
  WaitForInputReady ....................................... 230
  WaitForSystemAvailable ................................. 230
  WaitForAppAvailable ..................................... 231
  WaitForTransition ....................................... 231
  CancelWait ............................................... 232
autoECLOIA Events .......................................... 232
  NotifyOIAEvent ........................................... 232
  NotifyOIAError .......................................... 232
  NotifyOIAStop ............................................ 233
  Event Processing Example ............................... 233
autoECLPS Class ............................................. 233
  Properties .................................................. 234
autoECLPS Methods .......................................... 239
  RegisterPSEvent ......................................... 239
  RegisterKeyEvent ......................................... 240
  RegisterCommEvent ........................................ 240
  UnregisterPSEvent ....................................... 240
  UnregisterKeyEvent ....................................... 240
  UnregisterCommEvent ..................................... 241
  SetConnectionByName ...................................... 241
  SetConnectionByHandle ................................... 242
  SetCursorPos .............................................. 242
  SendKeys .................................................. 243
  SearchText ................................................ 243
  GetText .................................................... 244
  SetText .................................................... 245
  GetTextRect ............................................... 246
  StartCommunication ....................................... 246
  StopCommunication ....................................... 247
  StartMacro ............................................... 247
  Wait ....................................................... 248
  WaitForCursor ............................................ 248
  WaitWhileCursor .......................................... 249
  WaitForString ............................................ 250
  WaitWhileString .......................................... 250
  WaitForStringInRect ..................................... 251
  WaitWhileStringInRect ................................... 252
  WaitForAttrib ............................................. 253
  WaitWhileAttrib .......................................... 255
  WaitForScreen ............................................ 256
  WaitWhileScreen .......................................... 256
  CancelWait ............................................... 257
autoECLPS Events ............................................ 257
  NotifyPSEvent ............................................. 257
  NotifyKeyEvent ........................................... 258
  NotifyCommEvent .......................................... 258
  NotifyPSError ............................................. 258
  NotifyKeyError ............................................ 259
  NotifyCommError .......................................... 259
  NotifyPSStop .............................................. 259
  NotifyKeyStop ............................................. 259
  NotifyCommStop ............................................ 260
  Event Processing Example ............................... 260
autoECLScreenDesc Class .................................... 261
autoECLScreenDesc Methods ................................ 262
  AddAttrib .................................................. 262
  AddCursorPos .............................................. 263
  AddNumFields ............................................. 263
  AddNumInputFields ....................................... 264
  AddOIAInhibitStatus ...................................... 265
  AddString ................................................ 265
  AddStringInRect .......................................... 266
  Clear ...................................................... 267
autoECLScreenReco Class ................................... 267
autoECLScreenReco Methods ................................ 268
  AddFS ...................................................... 268
IsMatch ........................................................................................................ 268
RegisterScreen ................................................................................................ 269
RemovePS ........................................................................................................ 269
UnregisterScreen ............................................................................................. 269
autECLScreenReco Events ............................................................................. 270
NotifyRecoEvent ............................................................................................. 270
NotifyRecoError .............................................................................................. 270
NotifyRecoStop .............................................................................................. 270
Event Processing Example ............................................................................. 271
autECLSession Class ....................................................................................... 271
Properties ........................................................................................................ 272
autECLSession Methods ............................................................................... 277
RegisterSessionEvent ...................................................................................... 277
RegisterCommEvent ......................................................................................... 277
UnregisterSessionEvent .................................................................................... 277
UnregisterCommEvent ....................................................................................... 278
SetConnectionByName ..................................................................................... 278
SetConnectionByHandle ................................................................................... 278
StartCommunication ....................................................................................... 279
StopCommunication ....................................................................................... 280
autECLSession Events ..................................................................................... 280
NotifyCommEvent ........................................................................................... 280
NotifyCommError ........................................................................................... 280
NotifyCommStop ............................................................................................. 281
Event Processing Example ............................................................................. 281
autECLWinMetrics Class ............................................................................... 281
Properties ........................................................................................................ 282
autECLWinMetrics Methods ........................................................................... 288
RegisterCommEvent ......................................................................................... 289
UnregisterCommEvent ....................................................................................... 289
SetConnectionByName ...................................................................................... 289
SetConnectionByHandle ................................................................................... 290
GetWindowRect ............................................................................................... 290
SetWindowRect ................................................................................................ 291
StartCommunication ....................................................................................... 291
StopCommunication ....................................................................................... 292
autECLWinMetrics Events .............................................................................. 292
NotifyCommEvent ........................................................................................... 292
NotifyCommError ........................................................................................... 293
NotifyCommStop ............................................................................................. 293
Event Processing Example ............................................................................. 293
autECLXfer Class ........................................................................................... 294
Properties ........................................................................................................ 294
autECLXfer Methods ....................................................................................... 297
RegisterCommEvent ......................................................................................... 297
UnregisterCommEvent ....................................................................................... 297
SetConnectionByName ...................................................................................... 297
SetConnectionByHandle ................................................................................... 298
SendFile .......................................................................................................... 298
ReceiveFile ...................................................................................................... 299
StartCommunication ....................................................................................... 300
StopCommunication ....................................................................................... 301
autECLXfer Events ......................................................................................... 301
NotifyCommEvent ........................................................................................... 301
NotifyCommError ........................................................................................... 302
NotifyCommStop ............................................................................................. 302
Event Processing Example ............................................................................. 302
autSystem Class .............................................................................................. 303
autSystem Methods ........................................................................................ 303
Shell ............................................................................................................... 303
Inputnd ............................................................................................................ 303
autECLPageSettings Class ............................................................................. 304
Usage Notes ..................................................................................................... 304
Restrictions ...................................................................................................... 305
Connection types ............................................................................................ 305
Properties ........................................................................................................ 305
autECLPageSettings Methods ...................................................................... 311
RestrToTextDefaults ....................................................................................... 311
SetConnectionByName ..................................................................................... 312
SetConnectionByHandle ................................................................................... 312
autECLPageSettings Class ............................................................................. 313
Usage Notes ..................................................................................................... 313
Restrictions ...................................................................................................... 314
Properties ........................................................................................................ 314
autECLPageSettings Methods ...................................................................... 321
SetPDTMode .................................................................................................... 321
SetPrToDskAppend .......................................................................................... 322
SetPrToDskSeparate ......................................................................................... 322
SetSpecificPrinter .......................................................................................... 323
SetWinDefaultPrinter ...................................................................................... 324
SetConnectionByName ..................................................................................... 324
SetConnectionByHandle ................................................................................... 325
Support For Primary Interop Assemblies for Automation Objects .................... 326
Chapter 4. Host Access Class Library
LotusScript Extension ...................................................................................... 329
lsxECLConnection Class ............................................................................... 330
Properties ........................................................................................................ 331
lsxECLConnection Methods .......................................................................... 333
StartCommunication ....................................................................................... 333
StopCommunication ....................................................................................... 334
lsxECLConnList Class ...................................................................................... 334
Properties ........................................................................................................ 335
lsxECLConnList Methods ............................................................................... 335
Refresh ............................................................................................................ 335
FindConnectionByHandle ............................................................................... 336
FindConnectionByName ................................................................................... 336
lsxECLConnMgr Class ...................................................................................... 337
Properties ........................................................................................................ 337
lsxECLConnMgr Methods ............................................................................... 337
StartConnection .............................................................................................. 337
StopConnection .............................................................................................. 338
lsxECLField Class .......................................................................................... 339
Properties ........................................................................................................ 339
lsxECLField Methods ...................................................................................... 342
GetText ............................................................................................................ 343
SetText ............................................................................................................ 343
lsxECLFieldList Class ..................................................................................... 344
Properties ........................................................................................................ 344
lsxECLFieldList Methods ............................................................................... 344
Refresh ............................................................................................................ 344
FindFieldByRowCol ....................................................................................... 345
FindFieldByText ............................................................................................ 345
lsxECLLOIA Class .......................................................................................... 346
Properties ........................................................................................................ 347
lsxECLLOIA Methods ...................................................................................... 353
WaitForInputReady ......................................................................................... 353
WaitForSystemAvailable ............................................................................... 353
WaitForAppAvailable ...................................................................................... 354
WaitForTransition ........................................................................................... 354
lsxECLPS Class .............................................................................................. 355

Contents vii
## Chapter 5. Host Access Class Library for Java

### Appendix A. Sendkeys Mnemonic Keywords

### Appendix B. ECL Planes — Format and Content

### Appendix C. Notices

### Index
Figures

1. HACL Layers .................................... 1
2. Host Access Class Objects ...................... 11
3. Host Access Class Library Automation
   Objects ........................................... 200
## Tables

1. Copy-Construction and Assignment Examples 51
2. Primary Interop Assemblies for Personal Communications Automation Objects 327
3. Mnemonic Keywords for the Sendkeys Method 397
4. 3270 Field Attributes 401
5. 5250 Field Attributes 402
6. Mask Values 402
7. Color Plane Information 403
8. 3270 Extended Character Attributes 404
9. 5250 Extended Character Attributes 405
About This Book

This book provides necessary programming information for you to use the IBM® Personal Communications for Windows®, Version 5.8 Host Access Class Library (HACL). In this book, Windows refers to Windows 98, Windows NT®, Windows Me, Windows 2000, and Windows XP. Throughout this book, workstation refers to all supported personal computers. When only one model or architecture of the personal computer is referred to, only that type is specified.

Who Should Read This Book

This book is intended for programmers and developers who write application programs that use the Host Access Class Library (HACL) functions.

A working knowledge of Windows is assumed. For information about Windows, see the list of publications under “Where to Find More Information.”

This book assumes you are familiar with the language and compiler that you are using. For information on how to write, compile, or link-edit programs, refer to “Where to Find More Information” for the appropriate references for the specific language you are using.

How to Use This Book

This book is organized as follows:

- Chapter 1, “Introduction,” on page 1, gives an overview of the Host Access Class Library.
- Chapter 2, “Host Access Class Library C++,” on page 11, describes the Host Access Class Library C++ methods and properties.
- Chapter 5, “Host Access Class Library for Java,” on page 395, explains where you can find detailed information about the Host Access Class Library (HACL) Java™ classes.
- Appendix A, “Sendkeys Mnemonic Keywords,” on page 397, contains the mnemonic keywords for the Sendkeys method.
- Appendix B, “ECL Planes — Format and Content,” on page 401, describes the format and contents of the different data planes in the HACL presentation space model.

Where to Find More Information

The Personal Communications library includes the following publications:

- CD-ROM Guide to Installation
- Quick Beginnings
- Emulator User’s Reference
- Administrator’s Guide and Reference
- Emulator Programming
- Client/Server Communications Programming
- System Management Programming
- CM Mouse Support User’s Guide and Reference
- Host Access Class Library
- Configuration File Reference

See also:
- IBM 3270 Information Display System Data Stream Programmer’s Reference

In addition to the printed books, there are HTML documents provided with Personal Communications:

**Host Access Class Library for Java**
This HTML document describes how to write an ActiveX/OLE 2.0–compliant application to use Personal Communications as an embedded object.

**Host Access Beans for Java**
This HTML document describes Personal Communications emulator functions delivered as a set of JavaBeans™.

**Open Host Interface Objects for Java**
This HTML document describes how to write an OHIO–compliant application to use Personal Communications as an embedded object.

### What’s New in This Release

**Support for controlling Page and Printer settings added to the HACL interfaces**
The new C++ classes and methods for controlling page and printer settings are as follows:
- ECLPageSettings
- ECLPrinterSettings

The ECLSession class includes the following new methods to retrieve the Page and Printer settings sub-objects from an ECL session object:
- ECLPageSettings *GetPageSettings() (page)
- ECLPrinterSettings *GetPrinterSettings() (page)

The new Automation Objects for controlling Page and Printer settings are as follows:
- autECLPageSettings
- autECLPrinterSettings

These objects have also been added to the autECLSession object properties.

**Support For Primary Interop Assemblies for automation objects**
Automation objects exposed by Personal Communications can be used by applications written in any language that targets the .NET framework. Managed .NET applications can program Personal Communications by using the Primary Interop Assemblies (PIA) that wrap the automation objects.

See “Support For Primary Interop Assemblies for Automation Objects” on page 326 for details about this support.
Chapter 1. Introduction

The Host Access Class Library (HACL) is a set of objects that allows application programmers to access host applications easily and quickly. IBM Personal Communications provides support for a wide variety of programming languages and environments by supporting several different HACL layers: C++ objects, Java objects, Microsoft® COM-based automation technology (OLE), and LotusScript Extension (LSX). Each layer provides the same basic functionality, but each layer has some differences due to the different syntax and capabilities of each environment. The most functional and flexible layer is the C++ layer, which provides the basis for all others.

This layering concept allows the basic HACL functions to be used with a wide variety of programming environments including Java, Microsoft Visual Basic, Visual Basic for Applications, Lotus® Notes™, Lotus WordPro and Visual C++. The following figure shows the HACL layers.

Figure 1. HACL Layers

C++ Objects

This C++ class library presents a complete object-oriented abstraction of a host connection that includes: reading and writing the host presentation space (screen), enumerating the fields on the screen, reading the Operator Indicator Area (OIA) for status information, accessing and updating information about the visual emulator window, transferring files, and performing asynchronous notification of significant events.

HACL C++ 1390/1399 Code Page Support

Personal Communications Version 5.8 supports Japanese code page 1390/1399 on a Unicode session for the following HACL C++ methods:

- GetScreen (ECLField Class)
- SetText (ECLField Class)
- GetScreen (ECLPS Class)
- SearchText (ECLPS Class)
- SendKeys (ECLPS Class)

For more information about these methods, see their individual sections in Chapter 2, “Host Access Class Library C++,” on page 11.

Note: HACL C++ support for Personal Communications Unicode sessions is only available on Windows NT and Windows 2000 operating systems.

Java Objects

Java objects provides Java wrapping for all HACL functions similar to Host-on-Demand Version 3. See Chapter 5, “Host Access Class Library for Java,” on page 395 for details on HACL Java classes.

Automation Objects

The Host Access Class Library Automation Objects allow Personal Communications to support the Microsoft COM-based automation technology (formerly known as OLE automation). The HACL Automation Objects are a series of automation servers that allow automation controllers, for example, Microsoft Visual Basic, to programatically access Personal Communications’ data and functionality. In other words, applications that are enabled for controlling the automation protocol (automation controller) can control some of Personal Communications’ operations (automation server).


LotusScript Extension

The Host Access Class Library LotusScript Extension (LSX) is a language extension module for LotusScript (the scripting and macro language of Lotus Notes and all the Lotus SmartSuite® products). This LSX gives users of Lotus products access to the HACL functions through easy-to-use scripting functions.


ECL Concepts

The following sections describe several essential concepts of the Emulator Class Library (ECL). Understanding these concepts will aid you in making effective use of the library.

Connections, Handles and Names

In the context of the ECL, a connection is a single, unique Personal Communications emulator window. The emulator window may or may not be
actually connected to a host and may or may not be visible on the screen. For instance, a Personal Communications window can be in a disconnected state. Connections are distinguished by their connection handle or by their connection name. Most HACL objects are associated with a specific connection. Typically, the object takes a connection handle or connection name as a parameter on the constructor of the object. For languages like Visual Basic that do not support parameters on constructors, a member function is supplied for making the association. Once constructed, the object cannot be associated with any other connection. For example, to create an ECLPS (Presentation Space) object associated with connection 'B', the following code would be used:

**C++**

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

**Visual Basic**

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

**LotusScript Extension**

```lotus
dim myPSObj as new 1sxECLPS("B")
```

An HACL connection name is a single character from A–Z using uppercase characters. There are a maximum of 26 connection names, and Personal Communications is currently limited to 26 concurrent connections. A connection's name is the same as its EHLLAPI short session ID, and the session ID shown on the Personal Communications window title and OIA.

An HACL handle is a unique 32-bit number that represents a single connection. Unlike a connection name, a connection handle is not limited to 26 values, and the value itself has no significance to the application. You can use a connection handle across threads and processes to refer to the same connection.

For future expansion, applications should use the connection handle whenever possible. Most HACL objects accept a handle or a name when a connection needs to be identified. There are functions available in the base HACL class to convert a handle to a name, and a name to a handle. These functions are available from any HACL object.

**Note:** Connection properties are dynamic. For example, the connection type returned by GetConnType may change if you reconfigure the connection to a different host. In general, the application should not assume that connection properties remain fixed.

**Sessions**

In the context of the ECL, a session object (ECLSession) is only a container for all the other connection-specific objects. It provides a shortcut for an application to create a complete set of HACL objects for a particular connection. The term *session* should not be confused with the Personal Communications session concept. A Personal Communications session refers to a physical emulation window on the screen.
Creating or destroying ECLSession objects does not affect Personal Communications sessions (windows). An application can create any number of ECLSession objects that refer to the same or different connections.

**ECL Container Objects**

Several of the HACL classes act as containers of other objects. For example, the ECLSession object contains an instance of the ECLPS, ECLOIA, ECLWinMetrics, and ECLXfer objects. Containers provide methods to return a pointer to the contained object. For example, the ECLSession object has a GetOIA method, which returns a pointer to an OIA object. Contained objects are not implemented as public members of the container’s class, but rather are accessed only through methods.

For performance or other reasons, the contained objects may or may not be created when the container object is created. The class implementation may choose to defer construction of the contained objects until the first time the application requests a pointer to them. The application should not assume that contained objects are created at the same time as the container. For example, an instance of the ECLPS object may not be constructed when an ECLSession object is constructed. Instead, the ECLSession class may delay the construction of the ECLPS object until the first time the GetPS method is called.

When a container class is destroyed, all the contained instances are also destroyed. Any pointers that have been returned to the application become invalid and must not be used.

**Note:** Some HACL layers (such as the Automation Objects) may hide the containment scheme or recast it into a naming scheme that does not use explicit pointers.

**ECL List Objects**

Several HACL classes provide list iteration capabilities. For example, the ECLConnList class manages the list of connections. ECL list classes are not asynchronously updated to reflect changes in the list content. The application must explicitly call the Refresh method to update the contents of a list. This allows an application to iterate a list without concern that the list may change during the iteration.

**Events**

The HACL provides the capability of asynchronous notification of certain events. An application can choose to be notified when specific events occur. For example, the application can be notified when a new Personal Communications connection starts. Currently the HACL supports notification for the following events:

- Connection start/stop
- Communications connect/disconnect
- Operator keystrokes
- Presentation space or OIA updates

Notification of events is implemented by the ECLNotify abstract base classes. A separate class exists for each event type. To be notified of an event, the application must define and create an object derived from one of the ECLNotify abstract base classes. That object must then be registered by calling the appropriate HACL registration function. Once an application object is registered, its NotifyEvent method is called whenever the event of interest occurs.
Notes:
1. The application’s NotifyEvent method is called asynchronously on a separate thread of execution. Therefore, the NotifyEvent method should be reentrant, and if it accesses application resources, appropriate locking or synchronization should be used.
2. Some HACL layers (such as the Automation Objects) may not fully support or implement HACL events.

Error Handling
At the C++ layer, HACL uses C++ structured exception handling. In general, errors are indicated to the application by the throwing of a C++ exception with an ECLErr object. To catch errors, the application should enclose calls to the HACL objects in a try/catch block such as:

```c++
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");
}
```

When a HACL error is caught, the application can call methods of the ECLErr object to determine the exact cause of the error. The ECLErr object can also be called to construct a complete language-sensitive error message.

In both the Automation Objects layer and the LotusScript Extension layer, runtime errors cause an appropriate scripting error to be created. An application can use an On Error handler to capture the error, query additional information about the error and take appropriate action.

Addressing (Rows, Columns, Positions)
The HACL provides two ways of addressing points (character positions) in the host presentation space. The application can address characters by row/column numbers, or by a single linear position value. Presentation space addressing is always 1-based (not zero-based) no matter what addressing scheme is used.

The row/column addressing scheme is useful for applications that relate directly to the physical screen presentation of the host data. The rectangular coordinate system (with row 1 column 1 in the upper left corner) is a natural way to address points on the screen. The linear positional addressing method (with position 1 in the upper left corner, progressing from left to right, top to bottom) is useful for applications that view the entire presentation space as a single array of data elements, or for applications ported from the EHLLAPI interface which uses this addressing scheme.

At the C++ layer, the different addressing schemes are chosen by calling different signatures for the same methods. For example, to move the host cursor to a given screen coordinate, the application can call the ECLPS::SetCursorPos method in one of two signatures:

```
PSObj->SetCursorPos(81);
PSObj->SetCursorPos(2, 1);
```
These statements have the same effect if the host screen is configured for 80 columns per row. This example also points out a subtle difference in the addressing schemes — the linear position method can yield unexpected results if the application makes assumptions about the number of characters per row of the presentation space. For example, the first line of code in the example would put the cursor at column 81 of row 1 in a presentation space configured for 132 columns. The second line of code would put the cursor at row 2 column 1 no matter what the configuration of the presentation space.

Note: Some HACL layers may expose only a single addressing scheme.

Migrating from EHLLAPI

Applications currently written to the Emulator High Level Language API (EHLLAPI) can be modified to use the Host Access Class Library. In general it requires significant source code changes or application restructuring to migrate from EHLLAPI to HACL. HACL presents a different programming model than EHLLAPI and in general requires a different application structure to be effective.

The following sections will help a programmer familiar with EHLLAPI understand how HACL is similar and how HACL is different than EHLLAPI. Using this information you can understand how a particular application can be modified to use the HACL.

Note: EHLLAPI uses the term session to mean the same thing as an HACL connection. The terms are used interchangeably in this section.

Execution/Language Interface

At the most fundamental level, EHLLAPI and HACL differ in the mechanics of how the API is called by an application program.

EHLLAPI is implemented as a single call-point interface with multiple-use parameters. A single entry point (hlapi) in a DLL provides all the functions based on a fixed set of four parameters. Three of the parameters take on different meanings depending on the value of the forth command parameter. This simple interface makes is easier to call the API from a variety of programming environments and languages. The disadvantage is a lot of complexity packed into one function and four parameters.

HACL is an object-oriented interface that provides a set of programming objects instead of explicit entry points or functions. The objects have properties and methods that can be used to manipulate a host connection. You do not have to be concerned with details of structure packing and parameter command codes, but can focus on the application functions. HACL objects can only be used from one of the supported HACL layer environments (C++, Automation Objects, or LotusScript). These three layers are accessible to most modern programming environments such as Microsoft Visual C++, Visual Basic and Lotus SmartSuite applications.

Features

At a high level, HACL provides a number of features not available at the EHLLAPI level. There are also a few features of EHLLAPI not currently implemented in any HACL class.

HACL unique features include:
• Connection (session) start/stop functions
• Event notification for host communications link connect/disconnect
• Event notification for connection (session) start/stop
• Comprehensive error trapping
• Generation of language-specific error message text
• No architectural limit to the number of connections (sessions); currently, Personal Communications is limited to 26
• Support for multiple concurrent connections (sessions) and multithreaded applications
• Row/column addressing for host presentation space
• Simplified model for presentation space
• Automatic generation of list of fields and attributes
• Keyword-based function key strings

EHLLAPI features not currently implemented in the HACL include:
• Structured field support
• OIA character images
• Lock/unlock presentation space

Session IDs
The HACL architecture is not limited to 26 sessions. Therefore, a single character session ID such as that used in EHLLAPI is not appropriate. The HACL uses the concept of a connection handle, which is a simple 32-bit value that has no particular meaning to the application. A connection handle uniquely identifies a specific connection (session). You can use a connection handle across threads and processes to refer to the same connection.

All HACL objects and methods that need to reference a particular connection accept a connection handle. In addition, for backward compatibility and to allow a reference from the emulator user interface (which does not display the handle), some objects and methods also accept the traditional session ID. The application can obtain a connection handle by enumerating the connections with the ECLConnList object. Each connection is represented by an ECLConnection object. The ECLConnection::GetHandle method can be used to retrieve the handle associated with that specific connection.

It is highly recommended that applications use connection handles instead of connection names (EHLLAPI short session ID). Future implementations of the HACL may prevent applications that use connection names from accessing more than 26 sessions. In some cases it may be necessary to use the name, such as when the user is required to input the name of a specific session the application is to utilize. In the following C++ example, you supply the name of a session. The application then finds the connection in the connection list and creates PS and OIA objects for that session:

```c++
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) and it is put
//... into the UserRequestedID 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 ECOIA(ConnFound.GetHandle());

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

The second way of creating the PS and OIA objects shown in the example is not preferred because it uses the session name instead of the handle. This creates an implicit 26-session limit in this section of the code. Using the first example shown allows that section of code to work for any number of sessions.

**Presentation Space Models**

The HACL presentation space model is easier to use than that of EHLLAPI. The HACL presentation space consists of a number of planes, each of which contains one type of data. The planes are:

- Text
- Field attributes
- Color
- Extended attributes

The planes are all the same size and contain one byte for each character position in the host presentation space. An application can obtain any plane of interest using the ECLPS::GetScreen method.

This model is different from the EHLLAPI, in which text and non-text presentation space data is often interleaved in a buffer. An application must set the EHLLAPI session parameter to specify what type of data to retrieve, then make another call to copy the data to a buffer. The HACL model allows the application to get the data of interest in a single call and different data types are never mixed in a single buffer.

**SendKey Interface**

The HACL method for sending keystrokes to the host (ECLPS::Sendkeys) is similar to the EHLLAPI SendKey function. However, EHLLAPI uses cryptic escape codes to represent non-text keys such as Enter, PF1 and Backtab. The ECLPS object uses bracketed keywords to represent these keystrokes. For example, the following C++ sample would type the characters ABC at the current cursor position, followed by an Enter key:

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

**Events**

EHLLAPI provides some means for an application to receive asynchronous notification of certain events. However, the event models are not consistent (some
events use semaphores, others use window system messages), and the application is responsible for setting up and managing the event threads. The HACL simplifies all the event handling and makes it consistent for all event types. The application does not have to explicitly create multiple threads of execution, the HACL takes care of the threading internally.

However, you must be aware that the event procedures are called on a separate thread of execution. Access to dynamic application data must be synchronized when accessed from an event procedure. The event thread is spawned when the application registers for the event, and is terminated when the event is unregistered.

**PS Connect/Disconnect and Multithreading**

An EHLLAPI application must manage a connection to different sessions by calling ConnectPS and DisconnectPS EHLLAPI functions. The application must be carefully coded to avoid being connected to a session indefinitely because sessions have to be shared by all EHLLAPI applications. You must also ensure that an application is connected to a session before using certain other EHLLAPI functions.

The HACL does not require any explicit session connect or disconnect by the application. Each HACL object is associated with a particular connection (session) when it is constructed. To access different connections, the application only needs to create objects for each one. For example, the following example sends the keystrokes ABC to session A, then DEF to session B, and then the Enter key to session A. In an EHLLAPI program, the application would have to connect/disconnect each of the sessions since it can interact with only one at a time. An HACL application can just use the objects in any order needed:

```plaintext
ECLPS *PSA, *PSB;
PSA = new ECLPS('A');
PSB = new ECLPS('B');
PSA->Sendkeys("ABC");
PSB->Sendkeys("DEF");
PSA->Sendkeys("[enter]");
```

For applications that interact with multiple connections (sessions), this can greatly simplify the code needed to manage the multiple connections.

In addition to the single working session, EHLLAPI also places constraints on the multithreaded nature of the application. Connecting to the presentation space and disconnecting from the presentation space has to be managed carefully when the application has more than one thread calling the EHLLAPI interface, and even with multiple threads the application can interact with only one session at a time.

The ECLPS does not impose any particular multithreading restrictions on applications. An application can interact with any number of sessions on any number of threads concurrently.
Chapter 2. Host Access Class Library C++

This C++ class library presents a complete object-oriented abstraction of a host connection that includes: reading and writing the host presentation space (screen), enumerating the fields on the screen, reading the Operator Indicator Area (OIA) for status information, accessing and updating information about the visual emulator window, transferring files, and performing asynchronous notification of significant events. The class libraries support IBM VisualAge® C++ and Microsoft Visual C++ compilers.

The Host Access Class Library C++ layer consists of a number of C++ classes arranged in a class hierarchy. Figure 2 illustrates the C++ inheritance hierarchy of the Host Access Class Library C++ layer. Each object inherits from the class immediately above it in the diagram.

![Diagram of C++ class hierarchy]

**Figure 2. Host Access Class Objects**

Figure 2 also shows all the member functions of each class. Note that in addition to the functions shown for each class, classes inherit all the functions of the parent

© Copyright IBM Corp. 1997, 2004

11
class. For example, the function IsReady() is available on ECLSession, ECLPS, ECLOIA, ECLWinMetrics, and ECLXfer classes.

Each class is described briefly in the following sections. See the individual class descriptions in this chapter for more details.

All the examples shown in this chapter are supplied in the ECLSAMPS.CPP file. This file can be used to compile and execute any of the examples using any supported compiler.

The following is a brief overview of the Host Access Class Library C++ classes. Each class name begins with ECL, which is the common prefix for the Host Access Class Library.

- ECLBase, on page 15 is the base class for all ECL objects. It provides some basic utility methods such as the conversion of connection names and handles. Because all ECL objects inherit from this class, these methods can be used on any ECL object.
- ECLConnection, on page 19 represents a single Personal Communications connection and contains connection information such as the connection status, the type of connection (for example, 3270 or 5250), and the name and handle of the connection. This class is also the base class for all the connection-specific ECL objects such as ECLPS and ECLOIA.
- ECLConnList, on page 31 contains a list of all the Personal Communications connections that were in existence at the time the object was created or the last time the Refresh method was called. Each connection is represented by an ECLConnection object.
- ECLConnMgr, on page 38 enumerates all the currently running Personal Communications connections (windows) using the ECLConnList object. Is also provides methods for starting new connections and stopping connections.
- ECLCommNotify, on page 44 is a notification class that an application can use to be notified whenever a connection is disconnected from or connected to a host. It can be used to monitor the status of a connection and take action when a connection is disconnected unexpectedly.
- ECLErr, on page 48 provides a method for returning run-time error information from Host Access Class Library classes.
- ECLField, on page 50 contains information about a single field on the screen, such as the field attributes, field color, position on the screen or length. A method is also supplied to update input fields.
- ECLFieldList, on page 65 contains a collection of ECLField objects. When the Refresh method is called, the current host screen is examined, and the list of fields is extracted and used to build the list of ECLField objects. An application can use this collection to manage fields without having to build the list itself.
- ECLKeyNotify, on page 71 is a notification class that an application can use to be notified of keystroke events. The application can filter (remove) keystrokes, replace them with other keystrokes or discard them.
- ECLListener, on page 75 is the base class for all new HACL event listener objects. It provides common functions for all listener objects.
- ECLOIA, on page 76 provides access to operator status information such as shift indicators, input inhibited conditions and communications errors.
- ECLOIANotify, on page 88 is an abstract base class. Applications create objects derived from this class to receive notification of OIA changes.
- ECLPS, on page 90 represents the presentation space (screen) of a single connection. It contains methods for obtaining a copy of the screen contents in
the form of data planes. Each plane represents a specific aspect of the presentation space, such as the text, field attributes and color attributes. Methods are provided for searching for strings in the presentation space, sending keystrokes to the host, getting and setting the host cursor position, and many other functions. Also provided is an ECLFieldList object that can be used to enumerate the list of fields on the screen.

- ECLSEvent, on page 124 is an event object which is passed to PS event listeners when the presentation space has been updated. It contains information about the event including what caused the update and the portion of the screen which has been updated.

- ECLListener, on page 128 is an abstract base class. Applications create objects derived from this class to receive presentation space update events with all the information provided by the ECLSEvent object.

- ECLNotify, on page 130 is an abstract base class. Applications create objects derived from this class to receive notification of presentation space updates with minimal information.

- ECLRecoNotify, on page 133 is an abstract base class. Applications create objects derived from this class to receive notifications of screen recognitions.

- ECLScreenDesc, on page 135 is a class used to describe a single host screen. Screen description class objects are then used to trigger events when the described host screen appears, or to synchronously wait for a particular host screen.

- ECLScreenReco, on page 141 is a class used to collect a set of screen description objects and generate asynchronous events when any of the screens in the collection appear in the presentation space.

- ECLSession, on page 145 contains a collection of all the connection-specific objects. ECLSession can be used to easily create a complete set of objects for a particular connection.

- ECLStartNotify, on page 151 is a notification class that an application can use to be notified whenever a connection is started or stopped. It can be used to monitor the status of the system and take action when a connection is closed unexpectedly.

- ECLUpdateNotify, on page 155 is a notification class that an application can use to be notified whenever the host screen or OIA is updated.

- ECLWinMetrics, on page 155 represents the physical window in which the emulation is running. Methods are provided for getting and setting the window state (min, max, restored), window size and visibility.

- ECLXfer, on page 172 initiates file transfers to or from the host over the connection.

- ECLPageSettings, on page 177 control and retrieve the settings of the emulator session File > Page Setup dialog.

- ECLPrinterSettings, on page 186 control and retrieve the settings of the emulator session File > Printer Setup dialogs.

**Building C++ ECL Programs**

This section describes the mechanics of how to build a C++ program which uses the ECL. The source code preparation, compiling and linking requirements are described.
IBM Visual Age C++

The following sections describe how to prepare, compile, and link IBM VisualAge C++ applications that use the ECL. Personal Communications supports IBM VisualAge C++ Version 3.5 and later.

Source Code Preparation

Programs that use ECL classes must include the ECL header files to obtain class definitions and other compile-time information. Although, it is possible to include only the subset of header files the application requires, for simplicity, it is recommended that applications include all ECL header files using the ECALL.HPP file.

Any C++ source file which contains references to ECL objects or definitions should have the following statement before the first reference:

```cpp
#include "ecall.hpp"
```

Compiling

The compiler must be instructed to search the PCOMM subdirectory containing the ECL header files. This is done using the /I compiler option.

The application must be compiled for multithreaded execution using the /Gm+ compiler option.

Linking

The linker must be instructed to include the ECL linkable library file (PCSECLVA.LIB). This is done by specifying the fully qualified name of the library file on the linker command line.

Executing

When an application that uses the ECL is executed, the PCOMM libraries must be found in the system path. By default, the PCOMM directory is added to the system path during PCOMM installation.

Example

The following MAKFILE is an example of how to build an IBM VisualAge C++ application using the ECL:

```makefile
#--------------------------------------------------------------
# Sample make file for IBM VisualAge C++
#--------------------------------------------------------------
all: sample.exe

pcomm = c:\program1\person1\samples
debug = /O- /Ti+
msgs = /Word+pro+ret+use+cmd
includes = -I $(pcomm)

iccflags = /c /Gd- /Sm /Re /ss /Q /Gm+ $(msgs) $(debug) $(includes)

.cpp.obj: 
  icc $(iccflags) $*.cpp

# Compile and link SAMPLE.CPP
sample.exe: sample.obj
```
Microsoft Visual C++

The following sections describe how to prepare, compile, and link Microsoft Visual C++ applications that use the ECL. Personal Communications currently supports Microsoft Visual C++ compiler Version 4.2 and later.

Source Code Preparation

Programs that use ECL classes must include the ECL header files to obtain the class definitions and other compile-time information. Although it is possible to include only the subset of header files the application requires, for simplicity it is recommended that applications include all ECL header files using the ECLALL.HPP file.

Any C++ source file which contains references to ECL objects or definitions should have the following statement before the first reference:

#include "eclall.hpp"

Compiling

The compiler must be instructed to search the PCOMM subdirectory containing the ECL header files. This is done using the /I compiler option, or the Developer Studio Project Setting dialog.

The application must be compiled for multithreaded execution by using the /MT (for executable files), or /MD (for DLLs) compiler options.

Linking

The linker must be instructed to include the ECL linkable library file (PCSECLVC.LIB). This is done by specifying the fully qualified name of the library file on the linker command line, or by using the Developer Studio Project Settings dialog.

Executing

When an application that uses the ECL is executed, the PCOMM libraries must be found in the system path. By default, the PCOMM directory is added to the system path during PCOMM installation.

ECLBase Class

ECLBase is the base class for all ECL objects. It provides some basic utility methods such as the conversion of connection names and handles. Because all ECL objects inherit from this class, these methods can be used on any ECL object.

An application should not create objects of this class directly.

Derivation

None
ECLBase Methods

The following shows the methods that are valid for ECLBase classes.

```plaintext
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

This method returns the version of the Host Access Class Library. The value returned is the decimal version number multiplied by 100. For example, version 1.02 would be returned as 102.

**Prototype**

```
int GetVersion(void)
```

**Parameters**

None

**Return Value**

```
int
```

The ECL version number multiplied by 100.

**Example**

```plaintext
//-----------------------------------------------------------------
// 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

This method returns the name (A–Z) of the ECL connection handle specified. Note that this function may return a name even if the specified connection does not exist.

**Prototype**

```
char ConvertHandle2ShortName(long ConnHandle)
```

**Parameters**

```
long ConnHandle
```

The handle of an ECL connection.

**Return Value**

```
char
```

The name of the ECL connection in the range A–Z.
Example

//------------------------------------------------------------------------------
// 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

This method returns the connection handle of the ECL connection with the
specified name. The name must be in the range A–Z. Note that this function may
return a handle even if the specified connection does not exist.

Prototype
cchar ConvertShortName2Handle(char Name)

Parameters
cchar Name
  The name of an ECL connection in the range A–Z.

Return Value
cchar
  The handle of the ECL connection.

Example

//------------------------------------------------------------------------------
// 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%x \n", Handle);
}

} // end sample

ConvertTypeToString

This method converts a connection type returned by
ECLConnection::GetConnType() into a null terminated string. The string returned
is not language sensitive.
Prototype
void ConvertTypeToString(int ConnType, char *Buff)

Parameters
int ConnType  The connection type and must be one of the HOSTTYPE_* constants defined in ECLBASE.HPP.
char *Buff     A buffer of size TYPE_MAXSTRLEN as defined in ECLBase.hpp in which the string will be returned.

<table>
<thead>
<tr>
<th>ConnType</th>
<th>Returned String</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOSTTYPE_3270DISPLAY</td>
<td>&quot;3270 DISPLAY&quot;</td>
</tr>
<tr>
<td>HOSTTYPE_3270PRINTER</td>
<td>&quot;3270 PRINTER&quot;</td>
</tr>
<tr>
<td>HOSTTYPE_5250DISPLAY</td>
<td>&quot;5250 PRINTER&quot;</td>
</tr>
<tr>
<td>HOSTTYPE_5250PRINTER</td>
<td>&quot;5250 PRINTER&quot;</td>
</tr>
<tr>
<td>HOSTTYPE_VT</td>
<td>&quot;ASCII TERMINAL&quot;</td>
</tr>
<tr>
<td>HOSTTYPE_PC</td>
<td>&quot;PC SESSION&quot;</td>
</tr>
<tr>
<td>Any other value</td>
<td>&quot;UNKNOWN&quot;</td>
</tr>
</tbody>
</table>

Return Value
None

Example
//-------------------------------------------------------------------
// 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;
}

ConvertPos
This method is an inline function (macro) to convert an ECL position coordinate into a row/column coordinate given a position and the width of the presentation space. This function is faster than using ECLPS::ConvertPosToRowCol() for applications that already know (or assume) the width of the presentation space.

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

ULONG Pos
The linear positional coordinate to be converted (input).

ULONG *Row
The pointer to the returned row number of the given position (output).

ULONG *Col
The pointer to the returned column number of the given position (output).

ULONG *PSCols
The number of columns in the host presentation space (input).

Return Value
None

Example

//-------------------------------------------------------------------
// 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 Class

ECLConnection contains connection-related information for a given connection.
This object can be created directly by an application, and is also created indirectly
by the ECLConnList object or when creating any object that inherits from
ECLConnection (for example, ECLSession).

The information returned by the methods of this object are current as of the time
the method is called.

ECLConnection is inherited by ECLSession, ECLPS, ECLIOIA, ECLWinMetrics, and
ECLXfer.

Derivation

ECLBase > ECLConnection
ECLConnection

ECLConnection Methods

The following shows the methods that are valid for ECLConnection classes.

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

ECLConnection Constructor

This method constructs an ECLConnection object from either a connection name or a handle.

Prototype
ECLConnection(long ConnHandle)
ECLConnection(char ConnName)

Parameters
long ConnHandle Handle of connection to create a connection object.
char ConnName Name (A–Z) of connection to create a connection object.

Return Value
None

Example
//-------------------------------------------------------------------
// 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 Destructor**

This method destroys an ECLConnection object.

**Prototype**

~ECLConnection()

**Parameters**

None

**Return Value**

None

**Example**

//-------------------------------------------------------------------
// 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**

This method returns the host code page for which the connection is configured.

**Prototype**

unsigned int GetCodePage()
ECLConnection

Parameters
None

Return Value
unsigned int Host code page of the connection.

Example
//-------------------------------------------------------------------
// 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());
    }
}

GetHandle
This method returns the handle of the connection. This handle uniquely identifies
the connection and may be used in other ECL functions that require a connection
handle.

Prototype
long GetHandle()

Parameters
None

Return Value
long Connection handle of the ECLConnection object.

Example
The following example shows how to return the handle of the first connection in
the connection list.
//-------------------------------------------------------------------
// 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
This method returns the connection type. This connection type may change over time (for example, you may reconfigure the connection for a different host). The application should not assume the connection type is fixed. See below for connection types returned.

Note: The ECLBase::ConvertTypeToString function converts the connection type to a null terminated string.

Prototype
int GetConn Type()

Parameters
None

Return Value

<table>
<thead>
<tr>
<th>Value Returned</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOSTTYPE_3270DISPLAY</td>
<td>3270 display</td>
</tr>
<tr>
<td>HOSTTYPE_3270PRINTER</td>
<td>3270 printer</td>
</tr>
<tr>
<td>HOSTTYPE_5250DISPLAY</td>
<td>5250 display</td>
</tr>
<tr>
<td>HOSTTYPE_5250PRINTER</td>
<td>5250 printer</td>
</tr>
<tr>
<td>HOSTTYPE_VT</td>
<td>ASCII VT display</td>
</tr>
<tr>
<td>HOSTTYPE_UNKNOWN</td>
<td>Unknown connection type</td>
</tr>
</tbody>
</table>

Example
The following example shows how use the GetConnType method to return the connection type.

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

ECLConnection

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

This method returns the connection name (a single, alphabetic character from A–Z) of the connection. This name also corresponds to the EHLLAPI session ID.

Prototype
char GetName()

Parameters
None

Return Value
char    Connection short name.

Example
The following example shows how to use the GetName method to return the connection name.

//-------------------------------------------------------------------
// ECLConnection::GetName
//
// Find the first 3270 display connection in the current list of all connections and display its name (PComm 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;
        }
    }
} // end sample
GetEncryptionLevel

This method returns the encryption level of the current connection.

Prototype

int GetEncryptionLevel()

Parameters

None

Return Value

int Encryption level constant. The following table shows the value returned and its meaning.

<table>
<thead>
<tr>
<th>Value Returned</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENCRYPTION_NONE</td>
<td>No Encryption</td>
</tr>
<tr>
<td>ENCRYPTION_40BIT</td>
<td>40 bit encryption</td>
</tr>
<tr>
<td>ENCRYPTION_56BIT</td>
<td>56 bit encryption</td>
</tr>
<tr>
<td>ENCRYPTION_128BIT</td>
<td>128 bit encryption</td>
</tr>
<tr>
<td>ENCRYPTION_168BIT</td>
<td>168 bit encryption</td>
</tr>
<tr>
<td>ENCRYPTION_NOKEY</td>
<td>Encrypted without a key</td>
</tr>
</tbody>
</table>

Example

The following example shows how use the GetEncryptionLevel method to return the encryption level.

//-------------------------------------------------------
// 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\n");
        break;
      case ENCRYPTION_40BIT:
        printf("Encryption Level = 40 BIT\n");
        break;
      case ENCRYPTION_56BIT:
        printf("Encryption Level = 56 BIT\n");
        break;
      case ENCRYPTION_128BIT:
        printf("Encryption Level = 128 BIT\n");
        break;
      case ENCRYPTION_168BIT:
        printf("Encryption Level = 168 BIT\n");
        break;
      case ENCRYPTION_NOKEY:
        printf("Encryption Level = No Encryption\n");
        break;
    }
  }
  delete Info;
}
case ENCRYPTION_128BIT:
    printf("Encryption Level = 128 BIT");
    break;

case ENCRYPTION_168BIT:
    printf("Encryption Level = 168 BIT");
    break;

default:
    }
}
}

IsStarted

This method indicates if the connection is started. A started connection may or may not be connected to a host. Use the IsCommStarted function to determine if the connection is currently connected to a host.

Prototype

BOOL IsStarted()

Parameters

None

Return Value

BOOL TRUE value if the connection is started; FALSE value if the connection is not started.

Example

//-------------------------------------------------------------------
// 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

This method indicates if the connection is currently connected to the host (for example, it indicates if host communications is active for the connection). This function returns a FALSE value if the connection is not started (see “IsStarted”).
Prototype
BOOL IsCommStarted()

Parameters
None

Return Value
BOOL
TRUE value if the connection is connected to the host; FALSE value if the connection is not connected to the host.

Example
//-------------------------------------------------------------------
// 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

This method indicates if the connection is API-enabled. A connection that does not have API enabled cannot be used with the Host Access Class Library. This function returns a FALSE value if the connection is not started.

Prototype
BOOL IsAPIEnabled()

Parameters
None

Return Value
BOOL
TRUE value if API is enabled; FALSE value if API is not enabled.

Example
clide------------------------------------------
// ECLConnection::IsAPIEnabled
//
// Display list of all started connections which have APIs enabled.
clide------------------------------------------
void Sample14()
{
    ECLConnection *Info;  // Pointer to connection object

ECLConnection

```cpp
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

This method indicates that the connection is ready, meaning the connection is started, connected, and API-enabled. This function is faster and easier than calling IsStarted, IsCommStarted, and IsAPIEnabled.

Prototype

```cpp
BOOL IsReady()
```

Parameters

None

Return Value

BOOL TRUE if the connection is started, CommStarted, and API-enabled; FALSE if otherwise.

Example

```cpp
//-------------------------------------------------------------------
// 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
```

IsDBCSHost

This method indicates that the host is using a double byte character set (DBCS) code page.

Prototype

```cpp
BOOL IsDBCSHost()
```
StartCommunication

This method connects the PCOMM emulator to the host data stream. This has the same effect as going to the PCOMM emulator communication menu and choosing Connect.

Prototype
void StartCommunication()

Parameters
None

Return Value
BOOL
TRUE if the host code page is double byte; otherwise FALSE

StopCommunication

This method disconnects the PCOMM emulator from the host data stream. This has the same effect as going to the PCOMM emulator communication menu and choosing Disconnect.

Prototype
void StopCommunication()

Parameters
None
Return Value
None

Example
//-------------------------------------------------------------------
// 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...
", Info->GetName());
        Info->StopCommunication();
    }
}

} // end sample

RegisterCommEvent
This member function registers an application object to receive notification of all communication link connect/disconnect events. To use this function, the application must create an object derived from the ECLCommNotify class. A pointer to that object is then passed to this registration function. Implementation Restriction: An application can register only one object for communication event notification.

After a notify object has been registered with this function, it will be called whenever the connections communication link with the host connects or disconnects. The object will receive notification for all communication events whether they are caused by the StartCommunication() function or explicitly by the user. This event should not be confused with the connection start/stop event which is triggered when a new PCOMM connection starts or stops.

The optional InitEvent parameter causes an initial event to be generated when the object is registered. This can be useful to synchronize an event object with the current state of the communications link. If InitEvent is specified as FALSE, no initial event is generated when the object is registered. The default for this parameter is TRUE.

The application must call UnregisterCommEvent() before destroying the notification object. The object is automatically unregistered if the ECLConnection object where it is registered is destroyed.

See the description of “ECLCommNotify Class” on page 44 for more information.
Prototype
void RegisterCommEvent(ECLCommNotify *NotifyObject, BOOL InitEvent = TRUE)

Parameters
ECLCommNotify *NotifyObject
Pointer to an object derived from ECLCommNotify class.

BOOL InitEvent
Generate an initial event with the current state.

Return Value
None

Example
See "ECLCommNotify Class" on page 44 for an example of ECLConnection::RegisterCommEvent.

UnregisterCommEvent
This member function unregisters an application object previously registered for communication events with the RegisterCommEvent() function. A registered application notify object should not be destroyed without first calling this function to unregister it. If there is no notify object currently registered, or the registered object is not the NotifyObject passed in, this function does nothing (no error is thrown).

When a notify object is unregistered, its NotifyStop() member function will be called.

See the description of "ECLCommNotify Class" on page 44 for more information.

Prototype
void UnregisterCommEvent(ECLCommNotify *NotifyObject)

Parameters
ECLCommNotify *NotifyObject
This is a currently registered application notification object.

Return Value
None

Example
See "ECLCommNotify Class" on page 44 for an example of ECLConnection::UnregisterCommEvent.

ECLConnList Class
ECLConnList obtains information about all host connections on a given machine. An ECLConnList object contains a collection of all the connections that are currently known in the system.

The ECLConnList object contains a collection of ECLConnection objects. Each element of the collection contains information about a single connection. A
connection in this list may be in any state (for example, stopped or disconnected). All started connections appear in this list. The ECLConnection object contains the state of the connection.

The list is a snapshot of the set of connections at the time this object is created, or the last time the Refresh method was called. The list is not dynamically updated as connections are started and stopped. An application can use the RegisterStartEvent member of the ECLConnMgr object to be notified of connection start and stop events.

An ELCConnList object may be created directly by the application or indirectly by the creation of an ECLConnMgr object.

Derivation
ECLBase > ECLConnList

Usage Notes
An ECLConnList object provides a static snapshot of current connections. The Refresh method is automatically called upon construction of the ECLConnList object. If you use the ECLConnList object right after construction it contains an accurate representation of the list of connections at that moment. However, you should call the Refresh method in the ECLConnList object before you start accessing it if some time has passed since its construction.

The application can iterate over the collection by using the GetFirstConnection and GetNextConnection methods. The object pointers returned by GetFirstConnection and GetNextConnection are valid only until the Refresh member is called, or the ECLConnList object is destroyed. The application can locate a specific connection of interest in the list using the FindConnection function. Like GetNextConnection, the returned pointer is valid only until the next Refresh or the ECLConnList object is destroyed.

The order of connections in the connection list is undefined. An application should not make any assumptions about the list order. The order of connections in the list does not change until the Refresh function is called.

An ECLConnList object is automatically created when an ECLConnMgr object is created. However, the ECLConnList object can be created without an ECLConnMgr object.

ECLConnList Methods

The following section describes the methods that are valid for the ECLConnList class.

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

This method creates an ECLConnList object and initializes it with the current list of connections.

Prototype
ECLConnList();

Parameters
None

Return Value
None

Example
//-------------------------------------------------------------------
// 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 Destructor

This method destroys an ECLConnList object.

Prototype
~ECLConnList()

Parameters
None

Return Value
None

Example
polator ECLConnList
// ECLConnList::ECLConnList (Destructor)
// Dynamically construct a connection list object, display number
// of connections in the list, then delete the list.
polator ECLConnList
} // end sample
ECLConnList

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

The GetFirstConnection method returns a pointer to the first connection information object in the ECLConnList collection. See “ECLConnection Class” on page 19 for details on its contents. The returned pointer becomes invalid when the ECLConnList Refresh member is called or the ECLConnList object is destroyed. The application should not delete the returned object. If there are no connections in the list, NULL is returned.

Prototype

ECLConnection *GetFirstConnection()

Parameters

None

Return Value

ECLConnection * Pointer to the first ECLConnection object in the list. If there are no connections in the list, null is returned.

Example

//-------------------------------
// 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(); Info != NULL; Info = ConnList.GetNextConnection(Info)) {
        ECLBase::ConvertTypeToString(Info->GetConnType(), TypeString);
        printf("Connection %c is a %s type connection.\n", Info->GetName(), TypeString);
    }

**GetNextConnection**

This method returns a pointer to the next connection information object in the ECLConnList collection given a connection in the list. The application supplies a pointer to a connection previously returned by this function or GetFirstConnection. See "ECLConnection Class" on page 19 for details on its contents. The returned pointer is not valid after the next ECLConnList Refresh() call, or the ECLConnList object is destroyed. A NULL pointer is returned if there is an attempt to read past the end of the list. Successive calls to this method (supplying the prior pointer on each call) iterates over the list of connections. After the last connection is returned, subsequent calls return a NULL pointer. The first connection in the list can be obtained by supplying NULL for the previous connection.

**Prototype**

ECLConnection *GetNextConnection (ECLConnection *Prev)

**Parameters**

ECLConnection *Prev

Pointer returned by prior call to this function, GetFirstConnection(), or NULL.

**Return Value**

ECLConnection *

This is the pointer to the next ECLConnection object, or NULL if end of list.

**Example**

```cpp
// 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 TypeName[21];  // Type of connection

    for (Info = ConnList.GetFirstConnection(); Info != NULL; Info = ConnList.GetNextConnection(Info)) { // Get next one
        ECLBase::ConvertTypeToString(Info->GetConnType(), TypeName);
        printf("Connection is a %s type connection.\n", TypeName);
        printf("Connection %s is a %s type connection.\n", Info->GetName(), TypeName);
    }
}
```

**FindConnection**

This method searches the current connection list for the connection specified. The desired connection can be specified by handle or by name. There are two signatures for the FindConnection method. If the specified connection is found, a
pointer to the ECLConnection object is returned. If the specified connection is not in the list, NULL is returned. The list is not automatically refreshed by this function; if a new connection has started since the list was constructed or refreshed it is not found. The returned pointer is to an object in the connection list maintained by the ECLConnList object. The returned pointer is invalid after the next ECLConnList::Refresh call or the ECLConnList object is destroyed.

**Prototype**

ECLConnection *FindConnection(Long ConnHandle),

ECLConnection *FindConnection(char ConnName)

**Parameters**

- **Long ConnHandle**  
  Handle of the connection to find in the list.

- **char ConnName**  
  Name of the connection to find in the list.

**Return Value**

- **ECLConnection ***  
  Pointer to the requested ECLConnection object. If the specified connection is not in the list, NULL is returned.

**Example**

```c
//-------------------------------------------------------------------
// 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**

This method returns the number of connections currently in the ECLConnList collection.

**Prototype**

ULONG GetCount()

**Parameters GetCount()**

None
Return Value

ULONG Number of connections in the collection.

Example

//------------------------------------------------------------------------------------------------------------------------
// 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

This method updates the ECLConnList collection with a list of all currently known connections in the system. All pointers previously returned by GetNextConnection, GetFirstConnection and FindConnection become invalid.

Prototype

void Refresh()

Parameters

None

Return Value

None

Example

//------------------------------------------------------------------------------------------------------------------------
// 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 Class**

ECLConnMgr manages all Personal Communications connections on a given machine. It provides methods relating to the management of connections such as starting and stopping connections. It also creates an ECLConnList object to enumerate the list of all known connections on the system (see "ECLConnList Class" on page 31).

**Derivation**

ECLBase > ECLConnMgr

---

**ECLConnMgr Methods**

The following shows the methods that are valid with the ECLConnMgr class.

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

**ECLConnMgr Constructor**

This method constructs an ECLConnMgr object.

**Prototype**

ECLConnMgr()

**Parameters**

None

**Return Value**

None

**Example**

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

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 Deconstructor

This method destroys an ECLConnMgr object.

Prototype

~ECLConnMgr()

Parameters

None

Return Value

None

Example

//----------------------------------------------------------------------------
// ECLConnMgr::"ECLConnMgr" (Destructor)
//
// Create a connection manager 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

This method returns a pointer to an ECLConnList object. See "ECLConnList Class" on page 31 for more information. The ECLConnList object is destroyed when the ECLConnMgr object is destroyed.

Prototype

ECLConnList * GetConnList()
ECLConnMgr

Parameters
None

Return Value
ECLConnList * Pointer to an ECLConnList object

Example
  //-------------------------------------------------------------------
  //
  // 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

This method starts a new Personal Communications emulator connection. The ConfigParms string contains connection configuration information as explained under "Usage Notes."

Prototype
void StartConnection(char *ConfigParms)

Parameters
char *ConfigParms Null terminated connection configuration string.

Return Value
None

Usage Notes
The connection configuration string is implementation-specific. Different implementations of the Host Access Class Library may require different formats or information in the configuration string. This call is asynchronous in nature; the new connection may not yet be started when this call returns. An application can use the RegisterStartEvent function to be notified when a connection starts.

For Personal Communications, the configuration string has the following format:
PROFILE=["]<filename>["] [CONNNAME=<c>] [WINSTATE=<MAX|MIN|RESTORE|HIDE>]

Optional parameters are enclosed in square brackets []. The parameters are separated by at least one blank. Parameters may be in upper, lower, or mixed case and may appear in any order. The meaning of each parameter is as follows:
PROFILE=<filename> Names the Personal Communications workstation profile (.WS file) that contains the connection configuration information. This parameter is not optional; a profile name must be supplied. If the file name contains blanks, the name must be enclosed in double quotation marks. The
CONNNAME=<c>

Specifies the connection name (EHLLAPI short session ID) of the new connection. This value must be a single, alphabetic character (A-Z). If this value is not specified, the next available connection name is assigned automatically. If a connection already exists with the specified name an error is thrown (ERRMAJ_INVALID_SESSION).

WINSTATE=<MAX|MIN|RESTORE|HIDE>

Specifies the initial state of the emulator window. The default if this parameter is not specified is RESTORE.

**Note:** Due to the asynchronous nature of this call, it is possible for this function to return without error, but the connection fails to start. For example, if two connections are started in a short period of time with the same connection name the second StartConnection does not fail because the first connection has not yet started. However, when the second connection finally attempts to register its name it does fail to start because the name is already in use by the first connection. To minimize this possibility, connections should be started without specifying the CONNNNAME parameter if possible.

**Example**
The following is an example of the StartConnection method.

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**

This method stops (terminates) the emulator connection identified by the connection handle. See [“Usage Notes” on page 42](#) for contents of the StopParms string.

**Prototype**

void StopConnection(Long ConnHandle, char *StopParms)

**Parameters**

- **Long ConnHandle**: Handle of the connection to be stopped.
- **char * StopParms**: Null terminated connection stop parameter string.

**Return Value**

None
**Usage Notes**

The connection stop parameter string is implementation-specific. Different implementations of the Host Access Class Library may require a different format and contents of the parameter string. For Personal Communications the string has the following format:

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

Optional parameters are enclosed in square brackets [ ]. The parameters are separated by at least one blank. Parameters may be in upper, lower, or mixed case and may appear in any order. The meaning of the SAVEPROFILE parameter is as follows:

SAVEPROFILE=<YES|NO|DEFAULT> controls the saving of the current connection configuration back to the workstation profile (.WS file). This causes the profile to be updated with any configuration changes you may have made during the connection. If NO is specified, the connection is stopped and the profile is not updated. If YES is specified, the connection is stopped and the profile is updated with the current (possibly changed) configuration. If DEFAULT is specified, the update option is controlled by the File->Save On Exit emulator menu option. If this parameter is not specified, DEFAULT is used.

**Example**

```c
//-------------------------------------------------------------------
// 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 %s\.\n", 
            CM.GetConnList()->GetFirstConnection()->GetName());

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

**RegisterStartEvent**

This method registers an application object to receive notification of all connection start and stop events. To use this function, the application must create an object derived from the ECLStartNotify class. A pointer to that object is then passed to this registration function. **Implementation Restriction:** An application can register only one object for connection start or stop notification.

After a notify object has been registered with this function, it is called whenever a Personal Communications connection is started or stopped. The object receives notification for all connections whether they are started by the StartConnection
function or explicitly by you. This event should not be confused with the
start/stop Communication event, which is triggered when a connection connects or
disconnects from a host system.

See “ECLStartNotify Class” on page 151 for more information.

Prototype
void RegisterStartEvent(ECLStartNotify *NotifyObject)

Parameters
ECLStartNotify *NotifyObject

Pointer to object derived from the ECLStartNotify
class.

Return Value
None

Example
//-------------------------------------------------------------------
// ECLConnMgr::RegisterStartEvent
//
// See “ECLStartNotify Class” on page 151 for example of this method.
//-------------------------------------------------------------------

UnregisterStartEvent

This method unregisters an application object previously registered for connection
start or stop events with the RegisterStartEvent function. A registered application
notify object should not be destroyed without first calling this function to
unregister it. If there is no notify object currently registered, or the registered object
is not the NotifyObject passed in, this function does nothing (no error is thrown).

When a notify object is unregistered, its NotifyStop method is called.

See “ECLStartNotify Class” on page 151 for more information.

Prototype
void UnregisterStartEvent(ECLStartNotify *NotifyObject)

Parameters
None

Return Value
None

Example
//-------------------------------------------------------------------
// ECLConnMgr::UnregisterStartEvent
//
// See “ECLStartNotify Class” on page 151 for example of this method.
//-------------------------------------------------------------------
ECLCommNotify

ECLCommNotify Class

ECLCommNotify is an abstract base class. An application cannot create an instance of this class directly. To use this class, the application must define its own class which is derived from ECLCommNotify. The application must implement the NotifyEvent() member function in its derived class. It may also optionally implement NotifyError() and NotifyStop() member functions.

The ECLCommNotify class is used to allow an application to be notified of communications connect/disconnect events on a PCOMM connection. Connect/disconnect events are generated whenever a PCOMM connection (window) is connected or disconnected from a host system.

To be notified of communications connect/disconnect events, the application must perform the following steps:
1. Define a class derived from ECLCommNotify.
2. Implement the derived class and implement the NotifyEvent() member function.
3. Optionally implement the NotifyError() function, NotifyStop() function or both.
4. Create an instance of the derived class.
5. Register the instance with the ECLConnection::RegisterCommEvent() function.

The example shown demonstrates how this may be done. When the above steps are complete, each time a connection’s communications link is connected or disconnected from a host, the applications NotifyEvent() member function will be called.

If an error is detected during event generation, the NotifyError() member function is called with an ECLErr object. Events may or may not continue to be generated after an error, depending on the nature of the error. When event generation terminates (either due to an error, by calling the ECLConnection::UnregisterCommEvent, or by destruction of the ECLConnection object) the NotifyStop() member function is called. However event notification is terminated, the NotifyStop() member function is always called, and the application object is unregistered.

If the application does not provide an implementation of the NotifyError() member function, the default implementation is used (a simple message box is displayed to the user). The application can override the default behavior by implementing the NotifyError() function in the applications derived class. Likewise, the default NotifyStop() function is used if the application does not provide this function (the default behavior is to do nothing).

Note that the application can also choose to provide its own constructor and destructor for the derived class. This can be useful if the application wants to store some instance-specific data in the class and pass that information as a parameter on the constructor. For example, the application may want to post a message to an application window when a communications event occurs. Rather than define the window handle as a global variable (so it would be visible to the NotifyEvent() function), the application can define a constructor for the class which takes the window handle and stores it in the class member data area.

The application must not destroy the notification object while it is registered to receive events.
Implementation Restriction: Currently the ECLConnection object allows only one notification object to be registered for communications event notification. The ECLConnection::RegisterCommEvent will throw an error if a notify object is already registered for that ECLConnection object.

Derivation

ECLBase > ECLNotify > ECLCommNotify

Example

//-------------------------------------------------------------------
// 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()
{
    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 Methods

The following section describes the methods that are valid for the
ECLCommNotify class:

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
This method is a “pure virtual” member function (the application must implement
this function in classes derived from ECLCommNotify). This function is called
whenever a connection starts or stops and the object is registered for start/stop
events. The Connected BOOL is TRUE if the communications link is connected, or
FALSE if it is not connected to the host.

Prototype
virtual void NotifyEvent (ECLConnection *ConnObj, BOOL Connected)

Parameters
ECLConnection *ConnObj
This is the pointer to ECLConnection object where
the event occurred.

BOOL Connected
This is TRUE if comm link is connected and FALSE
if disconnected.

Return Value
None

NotifyError
This method is called whenever the ECLConnection object detects an error during
event generation. The error object contains information about the error (see
"ECLErr Class" on page 48). Events may continue to be generated after the error,
depending on the nature of the error. If the event generation stops due to an error,
the NotifyStop() function is called. An application can choose to implement this
function or allow the ECLCommNotify base class to handle the error. The base
class will display the error in a message box using the text supplied by the
ECLErr::GetMsgText() function. If the application implements this function in its
derived class, it will override the base class function.

Prototype
virtual void NotifyError (ECLConnection *ConnObj, ECLErr ErrObject)

Parameters
ECLConnection *ConnObj
This is the pointer to ECLConnection object in
which the error occurred.

ECLErr ErrObject
This is the ECLErr object describing the error.

Return Value
None

NotifyStop
This method is called when event generation is stopped for any reason (for
example, due to an error condition or a call to
ECLConnection::UnregisterCommEvent, etc.).

Implementation Note: the reason code is currently unused and will be zero.

Prototype
virtual void NotifyStop (ECLConnection *ConnObj, int Reason)
### Parameters

**ECLConnection *ConnObj**
This is the ptr to ECLConnection object that is stopping notification.

**int Reason**
This is unused (zero).

### Return Value

None

### ECLErr Class

The ECLErr class provides a method of returning run-time error information from Host Access Class Library classes. In error situations, ECLErr objects are created and populated with error and diagnostic information. The ECLErr objects are then thrown as C++ exceptions. The error and diagnostic information can then be queried from the caught ECLErr object.

Applications should not create or throw ECLErr objects directly.

### Derivation

ECLBase > ECLErr

### ECLErr Methods

The following section describes the methods that are valid for the ECLErr class.

```cpp
const int GetMsgNumber()
cost int GetReasonCode()
cost char *GetMsgText()
```

#### GetMsgNumber

This method returns the message number that was set when this ECLErr object was created. Error message numbers are described in ERRORIDS.HPP.

**Prototype**

```cpp
cost int GetMsgNumber()
```

**Parameters**
None

**Return Value**

```cpp
cost int
```

The error message number.

**Example**

```cpp
// ECL::GetMsgNumber
// Cause an 'invalid parameters' error and try the ECL exception.
// The extract the error number and language-sensitive text.
void Sample31() {
  ECLPS *PS = NULL;
  try {
    PS = new ECLPS('A');
    PS->SetCursorPos(999,999); // Invalid parameters
  } catch (const ECLErr &err) {
```

---

**Host Access Class Library**

48
catch (ECLErr ErrObj) {
    printf("The following ECL error was trapped:\n\n");
    printf("%s \nError number: %lu\nReason code: %lu\n", ErrObj.GetMsgText(), ErrObj.GetMsgNumber(), ErrObj.GetReasonCode());
}

if (PS != NULL)
    delete PS;
} // end sample

GetReasonCode

This method gets the reason code (sometimes referred to as the secondary or minor return code) from the ECLErr object. This code is generally used for debugging and diagnostic purposes. It is subject to change in future versions of the Host Access Class Library and should not be used programmatically. Descriptions of the reason codes can be found in ERRORIDS.HPP.

Prototype

const int GetReasonCode()

Parameters

None

Return Value

const int

The ECLErr reason code.

Example

//---------------------------------------------------------------------------------------------
// ECLErr::GetReasonCode
//
// Cause an 'invalid parameters' error and tryp the ECL exception.
// The extract the error number and language-sensitive 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\n");
        printf("%s \nError number: %lu\nReason code: %lu\n", ErrObj.GetMsgText(), ErrObj.GetMsgNumber(), ErrObj.GetReasonCode());
    }

    if (PS != NULL)
        delete PS;
} // end sample

Chapter 2. Host Access Class Library C++  49
GetMsgText

This method returns the message text associated with the error code used to create this ECLErr object. The message text is returned in the language for which Personal Communications is currently installed.

Note: The returned pointer is invalid after the ECLErr object is deleted.

Prototype
const char *GetMsgText()

Parameters
None

Return Value
char * The message text associated with the error code that is part of this ECLErr object.

Example
//-------------------------------------------------------------------
// 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

Usage Notes
The message text is retrieved from the Personal Communications message facility.

ECLField Class

ECLField contains information for a given field in an ECLFieldList object contained by an ECLPS object. An application should not create an object of this type directly. ECLField objects are created indirectly by the ECLFieldList object.

An ECLField object describes a single field of the host presentation space. It has methods for querying various attributes of the field and for updating the text of the field (for example, modifying the field text). Field attributes cannot be modified.
Derivation

ECLBase > ECLField

Copy-Constructor and Assignment Operator

This object supports copy-construction and assignment. This is useful for an application that wants to easily capture fields on a host screen for later processing. Rather than allocate text buffers and copy the string contents of the field, the application can simply store the field in a private ECLField object. The stored copy retains all of the function of an ECLField object including the field’s text value, attributes, starting position, length, etc. For example, suppose an application wanted to capture the first input field of the screen. Table 1 shows two ways this could be accomplished.

<table>
<thead>
<tr>
<th>Save the field as a string</th>
<th>Save the field as an ECLField object</th>
</tr>
</thead>
<tbody>
<tr>
<td>`#include &quot;eclall.hpp&quot;</td>
<td>`#include &quot;eclall.hpp&quot;</td>
</tr>
<tr>
<td>```cpp</td>
<td>```cpp</td>
</tr>
<tr>
<td>char *SavePtr; // Ptr to saved string</td>
<td>ECLField SaveFld; // Saved field</td>
</tr>
<tr>
<td>ECLPS Ps('A'); // PS object</td>
<td>ECLPS Ps('A'); // PS object</td>
</tr>
<tr>
<td>ECLFieldList *List;</td>
<td>ECLFieldList *List;</td>
</tr>
<tr>
<td>ECLField *Fld;</td>
<td>ECLField *Fld;</td>
</tr>
<tr>
<td>// Get fld list and rebuild it</td>
<td>// Get fld list and rebuild it</td>
</tr>
<tr>
<td>List = Ps-&gt;GetFieldList();</td>
<td>List = Ps-&gt;GetFieldList();</td>
</tr>
<tr>
<td>List-&gt;Refresh();</td>
<td>List-&gt;Refresh();</td>
</tr>
<tr>
<td>// See if there is an input field</td>
<td>// See if there is an input field</td>
</tr>
<tr>
<td>Fld = List-&gt;GetFirstField(GetUnmodified);</td>
<td>Fld = List-&gt;GetFirstField(GetUnmodified);</td>
</tr>
<tr>
<td>if (Fld != NULL) {</td>
<td>if (Fld != NULL) {</td>
</tr>
<tr>
<td>// Copy the field's text value</td>
<td>// Copy the field object</td>
</tr>
<tr>
<td>SavePtr = malloc(Fld-&gt;Length() + 1);</td>
<td>SaveFld = *Fld;</td>
</tr>
<tr>
<td>Fld-&gt;GetScreen(SavePtr, Fld-&gt;Length()+1);</td>
<td>}</td>
</tr>
<tr>
<td>}</td>
<td>// We now have captured the field text</td>
</tr>
<tr>
<td>// We now have captured the field text</td>
<td>// including text, position, attrib</td>
</tr>
</tbody>
</table>

There are several advantages to using an ECLField object instead of a string to store a field:

- The ECLField object does all storage management of the field’s text buffer; the application does not have to allocate or free text buffers or calculate the size of the buffer required.
- The saved field retains all of the characteristics of the original field including its attributes and starting position. All of the usual ECLField member functions can be used on the stored field except SetText(). Note that the stored field is a copy of the original — its values are not updated when the host screen changes or when the ECLFieldList::Refresh() function is called. As a result, the field can be stored and used later in the application.

Assignment operator overrides are also provided for character strings and long integer value types. These overrides make it easy to assign new string or numeric values to unprotected fields. For example, the following sets the first two input fields of the screen:
ECLField

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

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

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

Notes:

1. ECLField objects initialized by copy-construction or assignment are read-only copies of the original field object. The SetText() method is invalid for such an object and will cause an ECLErr exception to be thrown. Because the objects are copies, they are not updated or deleted when the original field object is updated or deleted. The application is responsible for deleting copies of field objects when they are no longer needed.

2. Calling any method on an uninitialized ECLField object will return undefined results.

3. An ECLField object created by the application can be reassigned any number of times.

4. Assignments can only be made from another ECLField object, a character string, or a long integer value. Assigning any other data type to an ECLField object is invalid.

5. If an assignment is made to an ECLField object that currently is part of an ECLFieldList, the effect is to update only the field’s text value. This is allowed only if the field object is an unprotected field. For example, the following will modify the 2nd input field of the screen by copying the value from the 1st input field:

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;

Because Fld2 is part of an ECLFieldList, the above assignment is identical to:

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

Note that this will throw an ECLErr exception if Fld2 is protected. Also note that only the text of Fld2 is updated, not its attributes, position, or length.

6. Assigning a string to a field object is equivalent to calling the SetText() method. You can also assign numeric values without first converting to strings:

*Field = 1087;

This is equivalent to converting the number to a string and then calling the SetText() method.
ECLField Methods

The following section describes the methods that are valid for the ECLField class.

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

The following methods are valid for the ECLField class and are supported for Japanese code page 1390/1399 on a Unicode session:

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

Note: HACL C++ support for Personal Communications Unicode sessions is only available on Windows NT and Windows 2000 operating systems.

GetStart

This method returns the position in the presentation space of the first character of the field. There are two signatures for the GetStart method. U_LONG GetStart returns the position as a linear value with the upper left corner of the presentation space being "1". void GetStart(U_LONG *Row, U_LONG *Col) returns the position as a row and column coordinate.

Prototype
U_LONG GetStart(),

void GetStart(U_LONG *Row, U_LONG *Col)

Parameters
U_LONG *Row
This output parameter is a pointer to the row value to be updated.

U_LONG *Col
This output parameter is a pointer to the column value to be updated.

Return Value
U_LONG
Position in the presentation space represented as a linear array.
ECLField

Example
The following example shows how to return the position in the presentation space of the first character of the field.

```c
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(%lu,%lu,%lu) End(%lu,%lu,%lu)
                   Length(%lu)\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
This method returns the starting row position of a given field in the ECLFieldList collection for the connection associated with the ECLPS object.

Prototype
ULONG GetStartRow()

Parameters
None

Return Value
ULONG This is the starting row of a given field.

Example
```c
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(%lu,%lu,%lu) End(%lu,%lu,%lu)
                   Length(%lu)\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
```
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", \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

This method return the starting column position of a given field in the ECLFieldList collection for the connection associated with the ECLPS object.

Prototype
ULONG GetStartCol()

Parameters
None

Return Value
ULONG        This is the starting column of a given field.

Example
/--------------------------------------------------------------------------------------------------------------------------
// 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", \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

This method returns the position in the presentation space of the last character of the field. There are two signatures for the GetEnd method. ULONG GetEnd returns the position as a linear value with the upper left corner of the presentation space being "1". void GetEnd(ULONG *Row, ULONG *Col) returns the position as a row and column coordinate.

Prototype
ULONG GetEnd()

void GetEnd(ULONG *Row, ULONG *Col)

Parameters
ULONG *Row This output parameter is a pointer to the row value to be updated.

ULONG *Col This output parameter is a pointer to the column value to be updated.

Return Value
ULONG Position in the presentation space represented as a linear array.

Example
The following example shows how to return the position in the presentation space of the last character of the field.

// Example
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
    } catch (ECLErr Err) {
        printf("ECL Error: %s
", Err.GetMsgText());
    }
} // end sample
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

This method returns the ending row position of the field.

Prototype

ULONG GetEndRow()

Parameters

None

Return Value

ULONG  This is the ending row in a given field.

Example

 /*********************************************************************/
 /* 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());
        }
    }

Chapter 2. Host Access Class Library C++    57
**ECLField**

```cpp
download pPS;
catch (ECLErr Err) {
    printf("ECL Error: %s\n", Err.GetMsgText());
}
} // end sample
```

**GetEndCol**

This method returns the ending column position of a field.

**Prototype**

ULONG GetEndCol()

**Parameters**

None

**Return Value**

ULONG This is the ending row in a given field.

**Example**

```cpp
// 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(Ln)\n");
        for (pField = pFieldList->GetFirstField();
            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());
        } // end sample
    }
```

58 Host Access Class Library
GetLength

This method returns the length of the field. The length includes the entire field even if it spans multiple lines of the presentation space. It does not include the field attribute character that starts the field.

Prototype
ULONG GetLength()

Parameters
None

Return Value
ULONG Length of the field.

Example
The following example shows how to return the length of the field.

```cpp
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());
    }
}
```

GetScreen

The GetScreen method fills an application-supplied buffer with data from the field. The type of data copied to the buffer is selected with the optional Plane parameter. The default is to return the text plane data. The data returned is the field as it existed at the time this field object was created; it will not reflect the current contents of the field if it has been updated since the ECLFieldList::Refresh function was called.
The length of the data returned is the length of the field (see “GetLength” on page 59). When the TextPlane is copied, an additional null terminating byte is added after the last data byte. Therefore, the application should provide a buffer that is at least 1 byte more than the field length when getting the text plane. If the application buffer is too small the returned data is truncated. The number of bytes of copied to the application buffer is returned as the function result (not including the null terminator for copies of the text plane).

The FieldPlane cannot be obtained with this function. The ECLField::GetAttribute can be used to obtain the field attribute value.

Prototype

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

Parameters

char * Buff Pointer to application buffer to be filled with field data.
ULONG BuffLen Length of application buffer.
PS_PLANE Plane Optional parameter. Enumeration which indicates what plane of field data is to be retrieved. Must be one of TextPlane, ColorPlane, or ExtendedFieldPlane.

Return Value

ULONG Number of bytes copied to application buffer, not including trailing null character for TextPlane data.

1390/1399 Code Page Support

GetScreen is enabled for code page 1390/1399 on a Unicode session.

Prototype:

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

Parameters:

WCHAR *Buff Pointer to application buffer to be filled with field data.
ULONG BuffLen Length of application buffer.
PS_PLANE Plane Optional parameter. Enumeration which indicates what plane of field data is to be retrieved. Must be one of TextPlane, ColorPlane, or ExtendedFieldPlane.

Return Value:

ULONG Number of bytes copied to application buffer, not including trailing null character for TextPlane data.

Example

The following example shows how to return a pointer to the field data indicated by the Plane parameter.
SetText

This method populates a given field in the presentation space with the character string passed in as text. If the text exceeds the length of the field, the text is truncated. If the text is shorter than the field, the field is padded with nulls.

Prototype
void SetText(char *text)

Parameters
char *text Null terminated string to set in field.

Return Value
None

1390/1399 Code Page Support
SetText is enabled for code page 1390/1399 on a Unicode session.

Prototype:
void SetText(WCHAR *text)
Parameters:

WCHAR *text

Null terminated string to set in field.

Return Value:  None

Example

The following example shows how to populate a given field in the presentation space with the character string passed in as text.

```cpp
//-------------------------------------------------------------------
//
// 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**

This method determines if a given field in the presentation space has a particular attribute. The method returns a TRUE value if the field has the attribute or a FALSE value if the field does not have the attribute.

**Prototype**

BOOL IsModified()

BOOL IsProtected()

BOOL IsNumeric()

BOOL IsHighIntensity()

BOOL IsPenDetectable()
BOOL IsDisplay()

**Parameters**
None

**Return Value**

*BOOL*

Returns a TRUE value if the attribute is present; a FALSE value if the attribute is not present.

**Example**

The following example shows how to determine if a given field has an attribute.

```cpp
// 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
    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 ");
    }
  }
}
```

Chapter 2. Host Access Class Library C++ 63
GetAttribute

This method returns the attribute of the field. The value returned contains the bit flags for each of the possible field attributes (modified, protected, numeric, high intensity, pen, and display). See Appendix B, “ECL Planes — Format and Content,” on page 401 for more details on these bits. There is a method provided for each type of attribute (for example, IsModified or IsHighIntensity). This method can be used to obtain complete attribute information in a single call.

Prototype

unsigned char GetAttribute()

Parameters

None

Return Value

unsigned char Attribute bits of the field.

Example

The following example shows how to return the attribute of the field.

// 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(); Field != NULL; Field = FieldList->getNextField(Field)) { // First field
    if (Field->IsPenDetectable())
      printf("Penable ");
    else
      printf("NoPen ");
  }
  if (Field->IsDisplay())
    printf("Display 
");
  else
    printf("Hidden 
");
  delete PS;
} catch (ECLErr Err) {
  printf("ECL Error: %s\n", Err.GetMsgText());
}
} // end sample

//-------------------------------------------------------------------
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 Class

The ECLFieldList class performs operations on a list of fields in a host presentation space. An application should not create an ECLFieldList object directly, but only indirectly by creating an ECLPS object.

ECLFieldList contains a collection of all the fields in the presentation space. Each element of the collection is an ECLField object. See “ECLField Class” on page 50 for details on its properties and methods.

An ECLFieldList object provides a static snapshot of what the presentation space contained when the Refresh method was called. If the presentation space is updated after the call to Refresh(), the field list does not reflect those changes. An application must explicitly call Refresh to refresh the field list.

Once an application has called Refresh it can begin walking through the collection of fields using GetFirstField and GetNextField. If the location of a field is known, FindField can be used to locate it in the list directly.

Note: All ECLField object pointers returned by GetFirstField, GetNextField, and FindField become invalid when Refresh is called or the ECLFieldList object is destroyed.

Derivation

ECLBase > ECLFieldList

Properties

None

ECLFieldList Methods

The following section describes the methods that are valid for the ECLFieldList class.

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
This method gets a snapshot of all the fields currently in the presentation space. All ECLField object pointers previously returned by this object become invalid. To improve performance, the field data can be limited to the planes of interest. Note that the TextPlane and FieldPlane are always obtained.

Prototype
void Refresh(PS_PLANE Planes=TextPlane)

Parameters
PS_PLANE Planes Plane for which fields are built. Valid values are TextPlane, ColorPlane, FieldPlane, ExfieldPlane, and AllPlanes (to build for all). This is an enumeration defined in ECLPS.HPP. This optional parameter defaults to TextPlane.

Return Value
None

Example
The following example shows how to use the Refresh method to get a snapshot of all the fields currently in the presentation space.

```c
///-------------------------------------------------------------------
// 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
This method returns the number of fields present in the ECLFieldList collection (based on the most recent call to the Refresh method).

Prototype
ULONG GetFieldCount()
Parameters
None

Return Value
ULONG  Number of fields in the ECLFieldList collection.

Example
The following example shows how to use the GetFieldCount method to return the number of fields present in the ECLFieldList collection.

```cpp
// 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
This method returns a pointer to the first ECLField object in the collection.
ECLFieldList contains a collection of ECLField objects. See "ECLField Class" on page 50 for more information. The method returns a NULL pointer if there are no fields in the collection.

Prototype
ECLField * GetFirstField();

Parameters
None

Return Value
ECLField *  Pointer to an ECLField object. If there are no fields in the connection, a null is returned.

Example
The following example shows how to use the GetFirstField method to return a pointer to the first ECLField object in the collection.

```cpp
// ECLFieldList::GetFirstField
// Display starting position of every input (unprotected) field.
```
ECLFieldList

//-------------------------------------------------------------------------------
// ECLFieldList::Sample41
//-------------------------------------------------------------------------------
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:
");
    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

This method returns the next ECLField object in the collection after a given object.
If there are no more objects in the collection after the given object, a NULL pointer
is returned. An application can make repeated calls to this method to iterate over
the ECLField objects in the collection.

Prototype
ECLField *GetNextField(ECLField *Prev)

Parameters
ECLField *Prev
  A pointer to any ECLField object in the collection.
The returned pointer will be the next object after
this one. If this value is NULL a pointer to the first
object in the collection is returned. This pointer is a
pointer returned by the GetFirstField,
GetNextField, or FindField member functions.

Return Value
ECLField *
  A pointer to the next object in the collection. If
there are no more objects in the collection after the
Prev object, NULL is returned.

Example
The following example shows how to use the GetNextFieldInfo method to return a
pointer to the next ECLField object in the collection.

  ///-------------------------------------------------------------------
  // ECLFieldList::Sample41
  //
  // 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

        // Iterate 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

This method finds a field in the ECLFieldList collection using either text or a position. The position can be either a linear position or a row, column position. If a field contains the text or the position, a pointer to an ECLField object for that field is returned. The returned pointer is to an object in the field list collection. NULL is returned if the field is not found. When searching for text, the search begins at row1 column1 unless you specify a starting position. Also for text, this method will search forward in the list as a default; however, you can specify the direction to search explicitly.

Note: A search for text will be successful even if the text spans multiple fields. The field object returned will be the field where the found text begins.

Prototype

    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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ULONG Pos</td>
<td>Linear position to search for OR linear position to begin text search.</td>
</tr>
<tr>
<td>ULONG Row</td>
<td>Row position to search for OR row to begin text search.</td>
</tr>
<tr>
<td>ULONG Col</td>
<td>Column position to search for OR column to begin text search.</td>
</tr>
</tbody>
</table>
ECLFieldList

char *text  
String to search

PS_DIR Dir  
Direction to search

Return Value

ECLField *  
Pointer to an ECLField object if field is found. NULL if field is not found. Returned pointer is invalid after the next call to Refresh.

Example

The following is an example of the FindField method.

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

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

} // end sample
//-------------------------------------------------------------------

Host Access Class Library
ECLKeyNotify Class

ECLKeyNotify is an abstract base class. An application cannot create an instance of this class directly. To use this class, the application must define its own class which is derived from ECLKeyNotify. The application must implement the NotifyEvent() member function in its derived class. It may also optionally implement NotifyError() and NotifyStop() member functions.

The ECLKeyNotify class is used to allow an application to be notified of keystroke events. The application can also choose to filter (remove) the keystrokes so they are not sent to the host screen, or replace them with other keystrokes. Keystroke notifications are queued so that the application will always receive a notification for each and every keystroke. Only keystrokes made by the real physical keyboard are detected by this object; keystrokes sent to the host by other ECL objects (such as ECLPS::SendKeys) do not cause keystroke notification events.

To be notified of keystroke events, the application must perform the following steps:
1. Define a class derived from ECLKeyNotify.
2. Implement the derived class and implement the NotifyEvent() member function.
3. Optionally implement the NotifyError() and/or NotifyStop() functions.
4. Create an instance of the derived class.
5. Register the instance with the ECLPS::RegisterKeyEvent() function.

The example shown demonstrates how this may be done. When the above steps are complete, each keystroke in the emulator window will cause the applications NotifyEvent() member function to be called. The function is passed parameters indicating the type of keystroke (plain ASCII key, or special function key), and the value of the key (a single ASCII character, or a keyword representing a function key). The application may perform any functions required in the NotifyEvent() procedure, including calling other ECL functions such as ECLPS::SendKeys(). The application returns a value from NotifyEvent() to indicate if the keystroke is to be filtered or not (return 1 to filter (discard) the keystroke, return 0 to have it processed normally).

If an error is detected during keystroke event generation, the NotifyError() member function is called with an ECLErr object. Keystroke events may or may not continue to be generated after an error, depending on the nature of the error. When event generation terminates (either due to an error, by calling ECLPS::UnregisterKeyEvent, or by destruction of the ECLPS object) the NotifyStop() member function is called. However event notification is terminated, the NotifyStop() member function is always called, and the application object is unregistered.

If the application does not provide an implementation of the NotifyError() member function, the default implementation is used (a simple message box is displayed to the user). The application can override the default behavior by implementing the NotifyError() function in the applications derived class. Likewise, the default NotifyStop() function is used if the application does not provide this function (the default behavior is to do nothing).

Note that the application can also choose to provide its own constructor and destructor for the derived class. This can be useful if the application wants to store some instance-specific data in the class and pass that information as a parameter.
on the constructor. For example, the application may want to post a message to an application window when a keystroke occurs. Rather than define the window handle as a global variable (so it would be visible to the NotifyEvent() function), the application can define a constructor for the class which takes the window handle and stores it in the class member data area.

The application must not destroy the notification object while it is registered to receive events.

The same instance of a keystroke notification object can be registered with multiple ECLPS objects to receive keystrokes for multiple connections. Thus an application can use a single instance of this object to process keystrokes on any number of sessions. The member functions are passed a pointer to the ECLPS object for which the event occurred so an application can distinguish between events on different connections. The sample shown uses the same object to process keystrokes on two connections.

Implementation Restriction: Currently the ECLPS object allows only one notification object to be registered for a given connection. The ECLPS::RegisterKeyEvent will throw an error if a notify object is already registered for that ECLPS object.

Derivation

ECLBase > ECLNotify > ECLKeyNotify

Example

The following is an example of how to construct and use an ECLKeyNotify object.

```c++
// ECLKeyNotify class

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

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());
}

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
    }
}
ECLKeyNotify

// 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", Err.GetMsgText());
}
}
} // end sample

//-------------------------------------------------------------------

ECLKeyNotify Methods

The following section describes the methods that are valid for the ECLKeyNotify class.

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

This method is a “pure virtual” member function (the application must implement
this function in classes derived from ECLKeyNotify). This function is called
whenever a keystroke event occurs and the object is registered for keystroke
events. The return value indicates the disposition of the keystroke (return 1 to
discard, 0 to process).

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

Parameters

ECLPS *PSObj This is a ptr to ECLPS object in which the event
occurred.

char const KeyType[2] This is a null terminated 1–char string indicating
the type of key:

      A = Plain ASCII keystroke
      M = Mnemonic keyword

const char * const KeyString This is a null terminated string containing
the keystroke or mnemonic keyword. Keywords will
always be in lowercase (for example, "[enter]"). See Appendix A, “Sendkeys Mnemonic Keywords,” on page 397 for a list of mnemonic keywords.

Return Value

int This is the filter indicator.

1 = Filter (discard) keystroke
0 = Process keystroke (send to host)

NotifyError

This method is called whenever the ECLPS object detects an error during keystroke event generation. The error object contains information about the error (see “ECLErr Class” on page 48). Keystroke events may continue to be generated after the error, depending on the nature of the error. If keystroke event generation stops due to an error, the NotifyStop() function will be called.

Prototype

virtual void NotifyError (ELLPS *PSobj, ECLErr ErrObject)

Parameters

ECLPS *PSObj This is the ptr to ECLPS object in which the error occurred.

ECLErr ErrObject This is the ECLErr object describing the error.

Return Value

None

NotifyStop

This method is called when keystroke event generation is stopped for any reason (for example, due to an error condition, a call to ECLPS::UnregisterKeyEvent, destruction of the ECLPS object, etc.).

Prototype

virtual void NotifyStop (ELLPS *PSObj, int Reason)

Parameters

ECLPS *PSObj This is the ptr to ECLPS object in which events are stopping.

int Reason This is unused (zero).

Return Value

None

ECLListener Class

ECLListener is the base class for all HACL "listener" objects. Listeners are objects which are registered to receive particular types of asynchronous events. Methods on the listener objects are called when events occur or errors are detected.

There are no public methods on the ECLListener class.
Derivation

ECLBase > ECLListener

Usage Notes

Applications do not use this class directly, but create instances of classes which are derived from it (for example, ECLPSListener).

ECLOIA Class

ECLOIA provides Operator Information Area (OIA) services.

Because ECLOIA is derived from ECLConnection, you can obtain all the information contained in an ECLConnection object. See "ECLConnection Class" on page 19 for more information.

The ECLOIA object is created for the connection identified upon construction. You may create an ECLOIA object by passing either the connection name (a single, alphabetic character from A-Z) or the connection handle, which is usually obtained from the ECLConnList object. There can be only one Personal Communications connection with a given name or handle open at a time.

Derivation

ECLBase > ECLConnection > ECLOIA

Usage Notes

The ECLSession class creates an instance of this object. If the application does not need other services, this object may be created directly. Otherwise, consider using an ECLSession object to create all the objects needed.

ECLOIA Methods

The following section describes the methods that are valid for the ECLOIA class.

ECLOIA(char ConnName)
ECLOIA(long ConnHandle)
~ECLOIA()
BOOL IsAlphanumeric()
BOOL IsAPL()
BOOL IsKatakana()
BOOL IsHiragana()
BOOL IsDBCS()
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 Constructor

This method creates an ECLOIA object from a connection name (a single, alphabetic character from A-Z) or a connection handle. There can be only one Personal Communications connection started with a given name.

Prototype

ECLOIA(char ConnName)

ECLOIA(long ConnHandle)

Parameters

char ConnName  One-character short name of the connection (A-Z)

long ConnHandle  Handle of an ECL connection.

Return Value

None

Example

The following example shows how to create an ECLOIA object using the connection name.

```cpp
// 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());
    }
}
```
Return Value

**BOOL** TRUE if the keyboard is in alphanumeric mode; FALSE if the keyboard is not in alphanumeric mode.

Example

The following example shows how to determine if the OIA indicates that the keyboard is in alphanumeric mode.

```c
//-------------------------------------------------------------------
// 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

This method checks to determine if the OIA indicates that the keyboard is in APL mode.

Prototype

**BOOL** IsAPL()

Parameters

None

Return Value

**BOOL** TRUE if the keyboard is in APL mode; FALSE if the keyboard is not in APL mode.

Example

The following example shows how to determine if the OIA indicates that the keyboard is in APL mode.

```c
//-------------------------------------------------------------------
// 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
```
IsKatakana

This method checks to determine if the OIA indicates that Katakana characters are enabled.

Prototype

BOOL IsKatakana()

Parameters

None

Return Value

BOOL TRUE if Katakana characters are enabled; FALSE if Katakana characters are not enabled.

Example

The following example shows how to determine if the OIA indicates that Katakana characters are enabled.

```
// ECLOIA::IsKatakana
//
// Determine status of connection 'A' OIA indicator
//........................................................................
void Sample48()
{
  ECLOIA OIA('A'); // OIA object for connection A
  if (OIA.IsKatakana())
    printf("Katakana.\n");
  else
    printf("Not Katakana.\n");
} // end sample
```

IsHiragana

This method checks to determine if the OIA indicates that Hiragana characters are enabled.

Prototype

BOOL IsHiragana()

Parameters

None

Return Value

BOOL TRUE if Hiragana characters are enabled; FALSE if Hiragana characters are not enabled.

Example

```
//........................................................................
// ECLOIA::IsHiragana
//
// Determine status of connection 'A' OIA indicator
//........................................................................
void Sample49()
{
  ECLOIA OIA('A'); // OIA object for connection A
  if (OIA.IsHiragana())
    printf("Hiragana.\n");
  ```
IsDBCS

This method checks to determine if the OIA indicates that the cursor is at a Double Byte Character Set (DBCS) location.

Prototype

BOOL IsDBCS()

Parameters

None

Return Value

BOOL TRUE if the DBCS characters are enabled; FALSE if the DBCS characters are not enabled.

Example

The following example shows how to determine if the OIA indicates that double byte character set (DBCS) characters are enabled.

```c
void Sample50() {
    ECLOIA OIA('A');  // OIA object for connection A
    if (OIA.IsDBCS())
        printf("DBCS.\n");
    else
        printf("Not DBCS.\n");
} // end sample
```
// 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
This method checks to determine if the OIA indicates that the cursor is at a numeric-only location.

Prototype
BOOL IsNumLock()

Parameters
None

Return Value
BOOL
   TRUE if Numeric is on; FALSE if not Numeric.

Example
The following example shows how to determine if the OIA indicates that the cursor is at a numeric location.
// 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
This method checks to determine if the OIA indicates that the keyboard has Caps Lock on.

Prototype
BOOL IsCapsLock()

Parameters
None
**Return Value**

**BOOL**

TRUE if Caps Lock is on; FALSE if Caps Lock is not on.

**Example**

The following example shows how to determine if the OIA indicates that the keyboard has Caps Lock on.

```c
// 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**

This method checks to determine if the OIA indicates that the keyboard is in insert mode.

**Prototype**

**BOOL** IsInsertMode()

**Parameters**

None

**Return Value**

**BOOL**

TRUE if the keyboard is in insert mode; FALSE if the keyboard is not in insert mode.

**Example**

The following example shows how to determine if the OIA indicates that the keyboard is in insert mode.

```c
// 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
This method checks to determine if the OIA indicates that a communications error reminder condition exists.

Prototype
BOOL IsCommErrorReminder()

Parameters
None

Return Value
BOOL TRUE if a condition exists; FALSE if a condition does not exist.

Example
The following example shows how to determine if the OIA indicates that a communications error reminder condition exists.

//-------------------------------------------------------------------
// ECLIOIA::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");
}
//-------------------------------------------------------------------

IsMessageWaiting
This method checks to determine if the OIA indicates that the message waiting indicator is on. This can only occur for 5250 connections.

Prototype
BOOL IsMessageWaiting()

Parameters
None

Return Value
BOOL TRUE if the message waiting indicator is on; FALSE if the indicator is not on.

Example
The following example shows how to determine if the OIA indicates that the message waiting indicator is on.

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

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

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

}  // end sample

WaitForInputReady

The WaitForInputReady method waits until the OIA of the connection associated with the autECLOIA object indicates that the connection is able to accept keyboard input.

Prototype
BOOL WaitForInputReady( long nTimeOut = INFINITE )

Parameters
long nTimeOut          The maximum length of time to wait in milliseconds, this parameter is optional. The default is INFINITE.

Return Value
The method returns TRUE if the condition is met, or FALSE if nTimeOut (in milliseconds) has elapsed.

WaitForSystemAvailable

The WaitForSystemAvailable method waits until the OIA of the session connected with the ECLOIA object indicates that session is connected to a host system.

Prototype
BOOL WaitForSystemAvailable( long nTimeOut = INFINITE )

Parameters
long nTimeOut          The maximum length of time to wait in milliseconds, this parameter is optional. The default is INFINITE.

Return Value
The method returns TRUE if the condition is met, or FALSE if nTimeOut (in milliseconds) has elapsed.

WaitForAppAvailable

The WaitForAppAvailable method waits while the OIA of the connected session indicates that the application is initialized and ready for use.

Prototype
BOOL WaitForAppAvailable( long nTimeOut = INFINITE )

Parameters
long nTimeOut          The maximum length of time to wait in milliseconds, this parameter is optional. The default is INFINITE.
Return Value
The method returns TRUE if the condition is met, or FALSE if nTimeOut (in milliseconds) has elapsed.

WaitForTransition
The WaitForTransition method waits for the value at the specified position in the OIA of the connected session to change.

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

Parameters
BYTE nIndex The 1 byte Hex position of the OIA to monitor. This parameter is optional. The default is 3.
long nTimeOut The maximum length of time to wait in milliseconds, this parameter is optional. The default is INFINITE.

Return Value
The method returns TRUE if the condition is met, or FALSE if nTimeOut (in milliseconds) has elapsed.

InputInhibited
This method returns an enumerated value that indicates whether input is inhibited or not. If input is inhibited, the reason for the inhibit can be determined. If input is inhibited for more than one reason the highest value enumeration is returned (for example, if there is a communications error and a protocol programming error, the ProgCheck value is returned).

Prototype
INHIBIT_REASON InputInhibited ()

Parameters
None

Return Value
INHIBIT_REASON Returns one of the INHIBIT_REASON values as defined in ECLOIA.HPP. The value NotInhibited is returned if input is currently not inhibited.

Example
The following example shows how to determine whether input is inhibited or not.

```c++
//-------------------------------------------------------------------
// ECLOIA::InputInhibited
//-------------------------------------------------------------------
void Sample57() {
    ECLOIA OIA('A'); // OIA object for connection A
    switch (OIA.InputInhibited()) {
        case NotInhibited:
            printf("Input not inhibited.\n");
            break;
        case SystemWait:
```

Chapter 2. Host Access Class Library C++  85
ECLOIA

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

This method returns a set of status bits that represent various OIA indicators. This method can be used to collect a set of OIA indicators in a single call rather than making calls to several different IsXXX methods. Each bit returned represents a single OIA indicator where a value of 1 means the indicator is on (TRUE), and 0 means it is off (FALSE). A set of bitmask constants are defined in the ECLOIA.HPP header file for isolating individual indicators in the returned 32-bit value.

Prototype
ULONG GetStatusFlags()

Parameters
None

Return Value
ULONG
Set of bit flags defined as follows:

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Mask Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31 (msb)</td>
<td>OIAFLAG_ALPHANUM</td>
<td>IsAlphanumeric</td>
</tr>
<tr>
<td>30</td>
<td>OIAFLAG_APL</td>
<td>IsAPL</td>
</tr>
<tr>
<td>29</td>
<td>OIAFLAG_KATAKANA</td>
<td>IsKatakana</td>
</tr>
<tr>
<td>28</td>
<td>OIAFLAG_HIRAGANA</td>
<td>IsHiragana</td>
</tr>
<tr>
<td>27</td>
<td>OIAFLAG_DBCS</td>
<td>IsDBCS</td>
</tr>
<tr>
<td>26</td>
<td>OIAFLAG_UPSHIFT</td>
<td>IsUpperShift</td>
</tr>
<tr>
<td>25</td>
<td>OIAFLAG_NUMERIC</td>
<td>IsNumeric</td>
</tr>
<tr>
<td>24</td>
<td>OIAFLAG_CAPSLOCK</td>
<td>IsCapsLock</td>
</tr>
<tr>
<td>23</td>
<td>OIAFLAG_INSERT</td>
<td>IsInsertMode</td>
</tr>
<tr>
<td>22</td>
<td>OIAFLAG_COMMERR</td>
<td>IsCommErrorReminder</td>
</tr>
<tr>
<td>21</td>
<td>OIAFLAG_MSGWAIT</td>
<td>IsMessageWaiting</td>
</tr>
<tr>
<td>20</td>
<td>OIAFLAG_ENCRYPTED</td>
<td>IsConnectionEncrypted</td>
</tr>
<tr>
<td>19-4</td>
<td>&lt;reserved&gt;</td>
<td></td>
</tr>
</tbody>
</table>
### RegisterOIAEvent

This member function registers an application object to receive notifications of OIA update events. To use this function the application must create an object derived from ECLOIANotify. A pointer to that object is then passed to this registration function. Any number of notify objects may be registered at the same time. The order in which multiple listeners receive events is not defined and should not be assumed.

After an ECLOIANotify object is registered with this function, its NotifyEvent() method will be called whenever a update to the OIA occurs. Multiple updates to the OIA in a short time period may be aggregated into a single event.

The application must unregister the notify object before destroying it. The object will automatically be unregistered if the ECLOIA object is destroyed.

**Prototype**

```cpp
void RegisterOIAEvent(ECLOIANotify * notify)
```

**Parameters**

- `ECLOIANotify *`: Pointer to the ECLOIANotify object to be registered.

**Return Value**

- None

### UnregisterOIAEvent

This member function unregisters an application object previously registered with the RegisterOIAEvent function. An object registered to receive events should not be destroyed without first calling this function to unregister it. If the specific object is not currently registered, no action is taken and no error occurs.

When an ECLOIANotify object is unregistered its NotifyStop() method is called.

**Prototype**

```cpp
void UnregisterOIAEvent(ECLOIANotify * notify)
```

**Parameters**

- `ECLPSNotify *`: Pointer to the ECLOIANotify object to be unregistered.

**Return Value**

- None

---

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Mask Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-0</td>
<td>OIAFLAG_INHIBMASK</td>
<td>InputInhibited:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0=NotInhibited</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1=SystemWait</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2=CommCheck</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3=ProgCheck</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4=MachCheck</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5=OtherInhibit</td>
</tr>
</tbody>
</table>

---

Chapter 2. Host Access Class Library C++  87
ECLOIANotify Class

ECLOIANotify is an abstract base class. An application cannot create an instance of this class directly. To use this class, the application must define its own class which is derived from ECLOIANotify. The application must implement the NotifyEvent() member function in its derived class. It may also optionally implement NotifyError() and NotifyStop() member functions.

The ECLOIANotify class is used to allow an application to be notified of updates to the Operator Information Area. Events are generated whenever any indicator on the OIA is updated.

Derivation

ECLBase > ECLNotify > ECLOIANotify

Usage Notes

To be notified of OIA updates using this class, the application must perform the following steps:

1. Define a class derived from ECLOIANotify.
2. Implement the NotifyEvent method of the ECLOIANotify-derived class.
3. Optionally implement other member functions of ECLOIANotify.
4. Create an instance of the derived class.
5. Register the instance with the ECLOIA::RegisterOIAEvent() method.

After registration is complete, updates to the OIA indicators will cause the NotifyEvent() method of the ECLOIANotify-derived class to be called.

Note that multiple OIA updates which occur in a short period of time may be aggregated into a single event notification.

An application can choose to provide its own constructor and destructor for the derived class. This can be useful if the application needs to store some instance-specific data in the class and pass that information as a parameter on the constructor.

If an error is detected during event registration, the NotifyError() member function is called with an ECLErr object. Events may or may not continue to be generated after an error. When event generation terminates (due to an error or some other reason) the NotifyStop() member function is called. The default implementation of NotifyError() will present a message box to the user showing the text of the error messages retrieved from the ECLErr object.

When event notification stops for any reason (error or a call the ECLOIA::UnregisterOIAEvent) the NotifyStop() member function is called. The default implementation of NotifyStop() does nothing.

ECLOIANotify Methods

The following section describes the methods that are valid for the ECLOIANotify class and all classes derived from it.

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

**NotifyEvent**

This method is a **pure virtual** member function (the application **must** implement this function in classes derived from ECLOIANotify). This method is called whenever the OIA is updated and this object is registered to receive update events.

Multiple OIA updates may be aggregated into a single event causing only a single call to this method.

**Prototype**

virtual void NotifyEvent(ECLOIA * OIAObj) = 0

**Parameters**

ECLOIA * Pointer to the ECLOIA object which generated this event.

**Return Value**

None

**NotifyError**

This method is called whenever the ECLOIA object detects an error during event generation. The error object contains information about the error (see the ECLErr class description). Events may continue to be generated after the error depending on the nature of the error. If the event generation stops due to an error, the NotifyStop() method is called.

An application can choose to implement this function or allow the base ECLOIANotify class handle it. The default implementation will display the error in a message box using text supplied by the ECLErr::GetMsgText() method. If the application implements this function in its derived class it overrides this behavior.

**Prototype**

virtual void NotifyError(ECLOIA * OIAObj, ECLErr ErrObj)

**Parameters**

ECLOIA * Pointer to the ECLOIA object which generated this event.

ECLErr An ECLErr object which describes the error.

**Return Value**

None

**NotifyStop**

This method is called when event generation is stopped for any reason (for example, due to an error condition or a call to ECLOIA::UnregisterOIAEvent).

The reason code parameter is currently unused and will be zero.

The default implementation of this function does nothing.

**Prototype**

virtual void NotifyStop(ECLOIA * OIAObj, int Reason)
**ECLOIANotify**

**Parameters**

ECLOIA * 

Pointer to the ECLOIA object which generated this event.

int 

Reason event generation has stopped (currently unused and will be zero).

**Return Value**

None

---

**ECLPS Class**

The ECLPS class performs operations on a host presentation space.

The ECLPS object is created for the connection identified upon construction. You may create an ECLPS object by passing either the connection name (a single, alphabetic character from A-Z) or the connection handle, which is usually obtained from an ECLConnection object. There can be only one Personal Communications connection with a given name or handle open at a time.

**Derivation**

ECLBase > ECLConnection > ECLPS

**Properties**

None

**Usage Notes**

The ECLSession class creates an instance of this object. If the application does not need other services, this object may be created directly. Otherwise, you may want to consider using an ECLSession object to create all the objects needed.

---

**ECLPS Methods**

The following section describes the methods available for ECLPS.

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, Cam
ULONG StartPos, PS_DIR Dir=SrchForward, BOOL FoldCase=FALSE)
ULONG SearchText(const 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, PS_PLANE Plane=TextPlane)
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)
void SetText(char *text);
void SetText(char *text, ULONG AtPos);
void SetText(char *text, ULONG AtRow, ULONG AtCol);
void ConvertPosToRowCol(ULONG pos, ULONG *row, ULONG *col)
ULONG ConvertRowColToPos(ULONG row, ULONG col)
ULONG ConvertPosToCol(ULONG Pos)
BOOL WaitForCursor(int Row, int Col, long nTimeOut=INFINITE,
    BOOL bWaitForIR=TRUE)
BOOL WaitForWhileCursor(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 WaitForWhileString(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 WaitForWhileStringInRect(char* WaitString, int sRow, int sCol,
    int eRow, int eCol, long nTimeOut=INFINITE, BOOL bWaitForIR=TRUE,
The following methods are available for ECLPS and are supported for Japanese code page 1390/1399 on a Unicode session:

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)

Note: HACL C++ support for Personal Communications Unicode sessions is only available on Windows NT and Windows 2000 operating systems.

ECLPS Constructor
This method uses a connection name or handle to create an ECLPS object.

Prototype
ECLPS(char ConnName)
ECLPS(long ConnHandle)

Parameters
char ConnName One-character short name of the connection (A-Z).
long ConnHandle Handle of an ECL connection.
Return Value
None

Example
The following example shows how to use a connection name to create an ECLPS object.

```cpp
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 Destructor
This method destroys the ECLPS object.

Prototype
~ECLPS()

Parameters
None

Return Value
None

Example
The following example shows how to destroy an ECLPS object.

```cpp
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.\n", 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**

The GetPCCodePage method retrieves the number designating the code page in force for the personal computer.

**Prototype**

int GetPCCodePage()

**Parameters**

None

**Return Value**

int Number of the code page.

**GetHostCodePage**

The GetHostCodePage method retrieves the number designating the code page in force for the host computer.

**Prototype**

int GetHostCodePage()

**Parameters**

None

**Return Value**

int Number of the code page.

**GetOSCodePage**

The GetOSCodePage method retrieves the number designating the code page in force for the operating system on the personal computer.

**Prototype**

int GetOSCodePage()

**Parameters**

None

**Return Value**

int Number of the code page.

**GetSize**

This method returns the size of the presentation space for the connection associated with the ECLPS object. There are two signatures of the GetSize method. Using ULONG GetSize(), the size is returned as a linear value and represents the total number of characters in the presentation space. With void GetSize(ULONG *Rows, ULONG *Cols), the number of rows and columns of the presentation space is returned.
Prototype
ULONG GetSize()

void GetSize(ULONG *Rows, ULONG *Cols)

Parameters
ULONG *Rows This output parameter is the number of rows in the presentation space.
ULONG *Cols This output parameter is the number of columns in the presentation space.

Return Value
ULONG Size of the presentation space as a linear value.

Example
The following is an example of using the GetSize method.
//-------------------------------------------------------------------
// 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);
}

GetSizeRows
This method returns the number of rows in the Presentation Space for the connection associated with the ECLPS object.

Prototype
ULONG GetSizeRows()

Parameters
None

Return Value
ULONG This is the number of rows in the Presentation Space.

Example
The following is an example of using the GetSizeRows method.
//-------------------------------------------------------------------
// ECLPS::GetSizeRows
// // Display dimensions of connection 'A'
//-------------------------------------------------------------------
void Sample59() {
ECLPS PS('A'); // PS object for connection A
ULONG Rows, Cols, Len;
PS.GetSizeRows(); // Get num of rows
// 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
ECLPS

// 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);
}

GetSizeCols
This method returns the number of columns in the Presentation Space for the connection associated with the ECLPS object.

Prototype
ULONG GetSizeCols()

Parameters
None

Return Value
ULONG This is the number of columns in the Presentation Space.

Example
The following is an example of using the GetSizeCols method.
//-------------------------------------------------------------------
// 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);
}

GetCursorPos
This method returns the position of the cursor in the presentation space for the connection associated with the ECLPS object. There are two signatures for the
GetCursorPos method. Using ULONG GetCursorPos(), the position is returned as a linear (1-based) position. With void GetCursorPos(ULONG *Row, ULONG *Col), the position is returned as a row and column coordinate.

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

Parameters
ULONG *Row
This output parameter is the row coordinate of the host cursor.

ULONG *Col
This output parameter is the column coordinate of the host cursor.

Return Value
ULONG
Cursor position represented as a linear value.

Example
The following is an example of using the GetCursorPos method.
//-------------------------------------------------------------------
// 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
//@
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
This method returns the row position of the cursor in the Presentation Space for the connection associated with the ECLPS object.

Prototype
ULONG GetCursorPosRow()

Parameters
None

Return Value
ULONG
This is the row position of the cursor in the Presentation Space.
Example
The following is an example of using the GetCursorPosRow method.

```c
// Example of using GetCursorPosRow
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

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

GetCursorPosCol
This method returns the column position of the cursor in the Presentation Space for the connection associated with the ECLPS object.

Prototype
ULONG GetCursorPosCol()

Parameters
None

Return Value
ULONG
This is the column position of the cursor in the Presentation Space.

Example
The following is an example of using the GetCursorPosCol method.

```c
// Example of using GetCursorPosCol
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

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

The SetCursorPosition method sets the position of the cursor in the presentation space for the connection associated with the ECLPS object. There are two signatures for the SetCursorPosition method. The position can be specified as a linear (1-based) position using void SetCursorPosition(ULONG pos), or as a row and column coordinate using void SetCursorPosition(ULONG Row, ULONG Col).

Prototype
void SetCursorPosition(ULONG pos),

void SetCursorPosition(ULONG Row, ULONG Col)

Parameters

ULONG pos  Cursor position as a linear position.
ULONG Row  Cursor row coordinate.
ULONG Col   Cursor column coordinate.

Return Value
None

Example
The following is an example of using the SetCursorPosition method.

```c
void Sample61() {
  ECLPS PS('A'); // PS object for connection A
  PS.SetCursorPosition(2, 1); // Put cursor at row 2, column 1
  printf("Cursor of connection A set to row 2 column 1.\n");
  } // end sample
```
processed from left to right until the end of the string is reached. For example, the following string would cause the characters ABC to be typed at the current cursor position, followed by a 3270 Erase-end-of-field keystroke, followed by XYZ and a PF1 key:

ABC[eraseeof][tab]XYZ[pf1]

Note: Blank characters in the string are written to the host presentation space like any other plain text character. Therefore, blanks should not be used to separate keywords or text.

To send a left or right square bracket character to the host, it must be doubled in the text string (for example, it must occur twice to cause a single bracket to be written). The following example causes the string “A [:]” to be written to the presentation space.

A[[:]]

If you attempt to write keystrokes to a protected position on the screen, the keyboard locks and the remainder of the keystrokes are discarded.

Refer to Appendix A, “Sendkeys Mnemonic Keywords,” on page 397 for a list of keywords.

Prototype

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

Parameters

Char *text String of keys to send to the presentation space.
ULONG AtPos Position at which to start writing keystrokes.
ULONG AtRow Row at which to start writing keystrokes.
ULONG AtCol Column at which to start writing keystrokes.

Return Value

None

1390/1399 Code Page Support

SendKeys is enabled for code page 1390/1399 on a Unicode session.

Prototype:

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

Parameters:

WCHAR *text Unicode string to send to the presentation space.
ULONG AtPos Position at which to start writing keystrokes.
ULONG AtRow Row at which to start writing keystrokes.
ULONG AtCol Column at which to start writing keystrokes.

Return Value: None
Note: Before sending keystrokes to the Personal Communications session, be sure that the session is a Unicode session and that the current platform is Windows NT or Windows 2000. If the session is an ANSI session or the current platform is either Windows 98 or Windows ME, and a Unicode string is sent, junk characters will be displayed.

Example
The following is an example of using the SendKeys method.

```cpp
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[] = "[eraseoff]PCOMM 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
The SearchText method searches for text in the presentation space of the connection associated with the ECLPS object. The method returns the linear position at which the text is found, or zero if the text is not found. The search may be made in the forward (left to right, top to bottom) or backward (right to left, bottom to top) directions using the optional Dir parameter. The search can be case-sensitive or case folded (insensitive) using the optional FoldCase parameter.

If no starting position is given, the search starts at the beginning of the screen for forward searches, or at the end of the screen for backward searches. A starting position may be given in terms of a linear position or row and column coordinates. If a starting position is given it indicates the position at which to begin the search. Forward searches search from the starting position (inclusive) to the last character of the screen. Backward searches search from the starting position (inclusive) to the first character of the screen.

The search string must exist completely within the search area for the search to be successful (for example, if the search string spans over the specified starting position it will not be found).

The returned linear position may be converted to row and column coordinates using the base class ConvertPosToRowCol method.
Prototype

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 * const text, ULONG StartRow, 
                  ULONG StartCol, PS_DIR Dir=SrchForward, BOOL FoldCase=FALSE)

Parameters

char *text          Null-terminated string to search for.
PS_DIR Dir          Optional parameter indicating the direction in which to search. If specified, must be one of SrchForward or SrchBackward. The default is SrchForward.
BOOL FoldCase      Optional parameter indicating the case-sensitivity of the search. If specified as FALSE the text string must exactly match the presentation space including the use of uppercase and lowercase characters. If specified as TRUE, the text string will be found without regard to uppercase or lowercase. The default is FALSE.
ULONG StartPos      Indicates the starting linear position of the search. This position will be included in the search.
ULONG StartRow      Indicates the row in which to start the search.
ULONG StartCol      Indicates the column in which to start the search.

Return Value

ULONG          Linear position of the found string, or zero if not found.

1390/1399 Code Page Support

SearchText is enabled for code page 1390/1399 on a Unicode session.

Prototype:

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)

Parameters:

WCHAR *text          Null-terminated string to search for.
PS_DIR Dir           Optional parameter indicating the direction in which to search. If specified, must be one of SrchForward or SrchBackward. The default is SrchForward.
BOOL FoldCase        Optional parameter indicating the case-sensitivity of the search. If specified as FALSE the text string
must exactly match the presentation space including the use of uppercase and lowercase characters. If specified as TRUE, the text string will be found without regard to uppercase or lowercase. The default is FALSE.

**ULONG StartPos**
Indicates the starting linear position of the search.
This position will be included in the search.

**ULONG StartRow**
Indicates the row in which to start the search.

**ULONG StartCol**
Indicates the column in which to start the search.

**Return Value:**
**ULONG**
Linear position of the found string, or zero if not found.

**Example**
The following is an example of using the SearchText method.

```cpp
void Sample63() {  
    ECLPS PS('A');  // PS object  
    char FindStr[] = "IBM";  // String to search for  
    ULONG LastOne;  // Position of search result  

    // Case insensitive search of entire screen  
    if (PS.SearchText(FindStr, TRUE) != 0)  
        printf("Yes\n");  
    else  
        printf("No\n");  

    // Backward, case sensitive search on line 1  
    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 occurrence on the screen is at row %lu, column %lu.\n",  
            PS.ConvertPosToRow(LastOne), PS.ConvertPosToCol(LastOne));  
} // end sample
```

**GetScreen**
This method retrieves data from the presentation space of the connection associated with the ECLPS object. The data is returned as a linear array of byte values, one byte per presentation space character position. The array is not null terminated except when data is retrieved from the TextPlane, in which case a single null termination byte is appended.
The application must supply a buffer for the returned data, and the length of the buffer. If the requested data does not fit into the buffer it is truncated. For TextPlane data, the buffer must include at least one extra byte for the terminating null. The method returns the number of bytes copied to the application buffer (not including the terminating null for TextPlane copies).

The application must specify the number of bytes of data to retrieve from the presentation space. If the starting position plus this length exceeds the size of the presentation space an error is thrown. Data is returned starting at the given starting position or row 1, column 1 if no starting position is specified. Returned data is copied from the presentation space in a linear fashion from left to right, top to bottom spanning multiple rows up to the length specified. If the application wants to get screen data for a rectangular area of the screen, the GetScreenRect method should be used.

The application can specify any plane for which to retrieve data. If no plane is specified, the TextPlane is retrieved. See Appendix B, “ECL Planes — Format and Content,” on page 401 for details on the different ECL planes.

Prototype

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 Pointer to application supplied buffer of at least BuffLen size.
ULONG BuffLen Number of bytes in the supplied buffer.
ULONG StartPos Linear position in the presentation space at which to start the copy.
ULONG StartRow Row in the presentation space at which to start the copy.
ULONG StartCol Column in the presentation space at which to start the copy.
ULONG Length Linear number of bytes to copy from the presentation space.
PS_PLANE plane Optional parameter specifying which presentation space plane is to be copied. If specified, must be one of TextPlane, ColorPlane, FieldPlane, and ExfieldPlane. The default is TextPlane. See Appendix B, “ECL Planes — Format and Content,” on page 401 for the content and format of the different ECL planes.

Return Value

ULONG Number of data bytes copied from the presentation space. This value does not include the trailing null byte for TextPlane copies.
1390/1399 Code Page Support
GetScreen is enabled for code page 1390/1399 on a Unicode session.

Prototype:

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)

Parameters:
WCHAR *Buff     The string length should indicate the number of Unicode characters to be send. If not, parameter error will be returned by the function.
ULONG BuffLen    Number of Unicode characters in the supplied buffer.
ULONG StartPos   Linear position in the presentation space at which to start the copy.
ULONG StartRow   Row in the presentation space at which to start the copy.
ULONG StartCol   Column in the presentation space at which to start the copy.
ULONG Length     Linear position to copy from the presentation space.
PS_PLANE plane   Optional parameter specifying which presentation space plane is to be copied. If specified, must be one of TextPlane, ColorPlane, RawTextPlane, FieldPlane, and ExtendedFieldPlane. The default is TextPlane. See Appendix B, “ECL Planes — Format and Content,” on page 401 for the content and format of the different ECL planes.

Data String     Pre-allocated target Unicode string. When the Set Sessions Parameters function with Extended Attribute Bytes (EAB) option is issued, the length of the data string must be twice the size of the presentation space.

Return Value:
ULONG           Number of data bytes copied from the presentation space. This value does not include the trailing null byte for TextPlane copies.

Example
The following is an example of using the GetScreen method.

    //-------------------------------------------------------------------
    // ECLPS::GetScreen
    // Get text and other planes of data from the presentation space.
    //-------------------------------------------------------------------
    void Sample64() {
ECLPS

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;
}

GetScreenRect

This method retrieves data from the presentation space of the connection associated with the ECLPS object. The data is returned as a linear array of byte values, one byte per presentation space character position. The array is not null terminated.

The application supplies a starting and ending coordinate in the presentation space. These coordinates form the opposing corner points of a rectangular area. The presentation space within the rectangular area is copied to the application buffer as a single linear array. The starting and ending points may be in any spatial relationship to each other. The copy is defined to start from the row containing the uppermost point to the row containing the lowermost point, and from the left-most column to the right-most column. Both coordinates must be within the bounds of the size of the presentation space or an error is thrown. The coordinates may be specified in terms of linear position or row and column numbers.

The supplied application buffer must be at least large enough to contain the number of bytes in the rectangle. If the buffer is too small, no data is copied and zero is returned as the method result. Otherwise the method returns the number of bytes copied.

The application can specify any plane for which to retrieve data. If no plane is specified, the TextPlane is retrieved. See Appendix B, “ECL Planes — Format and Content,” on page 401 for details on the different ECL planes.

Prototype

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                  Pointer to application supplied buffer of at least BuffLen size.
ULONG BuffLen Number of bytes in the supplied buffer.
ULONG StartPos Linear position in the presentation space of one corner of the copy rectangle.
ULONG EndPos Linear position in the presentation space of one corner of the copy rectangle.
ULONG StartRow Row in the presentation space of one corner of the copy rectangle.
ULONG StartCol Column in the presentation space of one corner of the copy rectangle.
ULONG EndRow Row in the presentation space of one corner of the copy rectangle.
ULONG EndCol Column in the presentation space of one corner of the copy rectangle.
PS_PLANE plane Optional parameter specifying which presentation space plane is to be copied. If specified, must be one of TextPlane, ColorPlane, FieldPlane, or ExfieldPlane. The default is TextPlane. See Appendix B, “ECL Planes — Format and Content,” on page 401 for the content and format of the different ECL planes.

Return Value
ULONG Number of data bytes copied from the presentation space.

Example
The following is an example of using the GetScreenRect method.

----------------------------
// 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 (OfficeVision/VM main menu calendar)
PS.GetScreenRect(Buff, sizeof(Buff),
    5, 51,
    13, 76);

// Same as previous (specify any 2 opposite corners of the rectangle)
PS.GetScreenRect(Buff, sizeof(Buff),
    13, 51,
    5, 76);

}
ECLPS

// 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

The SetText method sends a character array to the Presentation Space for the
connection associated with the ECLPS object. Although this is similar to the
SendKeys method, it is different in that it does not send mnemonic keystrokes (for
example, [enter] or [pf1]).

If a position is not specified, the text is written starting at the current cursor
position.

Prototype

void SetText(char *text);

void SetText(char *text, ULONG AtPos);

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

Parameters

cchar *text Null terminated string of characters to copy to the
presentation space.

ULONG AtPos Linear position in the presentation space at which
to begin the copy.

ULONG AtRow Row in the presentation space of which to begin
the copy.

ULONG AtCol Column in the presentation space at which to begin
the copy.

Return Value

None

Example

The following is an example of using the SetText method.

//-------------------------------------------------------------------
// 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

//-------------------------------

ConvertPosToRowCol
The ConvertPosToRowCol method converts a position in the presentation space represented as a linear array to a position in the presentation space given in row and column coordinates. The position converted is in the presentation space for the connection associated with the ECLPS object.

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

Parameters
ULONG pos Position to convert in the presentation space represented as a linear array.
ULONG *row Converted row coordinate in the presentation space.
ULONG *col Converted column coordinate in the presentation space.

Return Value
None

Example
The following example shows how to convert a position in the presentation space represented as a linear array to a position shown in row and column coordinates.

//-------------------------------
// 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("IBM", 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

The ConvertRowColToPos method converts a position in the presentation space in row and column coordinates to a position in the presentation space represented as a linear array. The position converted is in the presentation space for the connection associated with the ECLPS object.

Prototype
ULONG ConvertRowColToPos(ULONG row, ULONG col)

Parameters
ULONG row Row coordinate to convert in the presentation space.
ULONG col Column coordinate to convert in the presentation space.

Return Value
ULONG Converted position in the presentation space represented as a linear array.

Example
The following example shows how to convert a position in the presentation space shown in row and column coordinates to a linear array position.

```c
 void Sample67() {
     ECLPS PS('A'); // PS Object
     ULONG FoundPos; // Linear position
     ULONG FoundRow,FoundCol;

     FoundPos = PS.SearchText("IBM", 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");
     }
```
Parameters
ULONG Pos
This is the linear position in the Presentation Space to convert.

Return Value
ULONG
This is the row position for the linear position.

Example
The following is an example of using the ConvertPosToRow method.

```cpp
void Sample67()
{
ECLPS PS('A'); // PS Object
ULONG FoundPos; // Linear position
ULONG FoundRow,FoundCol;

FoundPos = PS.SearchText("IBM", 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
This method takes a linear position value in the Presentation Space and returns the column in which it resides for the connection associated with the ECLPS object.

Prototype
ULONG ConvertPosToCol(ULONG Pos)

Parameters
ULONG Pos
This is the linear position in the Presentation Space to convert.

Return Value
ULONG
This is the column position for the linear position.

Example
The following is an example of using the ConvertPosToCol method.

```cpp
void Sample67()
{
    ECLPS PS('A'); // PS Object
    ULONG FoundPos; // Linear position
    ULONG FoundRow,FoundCol;

    FoundPos = PS.SearchText("IBM", 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
```
ECLPS

ECLPS PS('A');    // PS Object
ULONG FoundPos;  // Linear position
ULONG FoundRow, FoundCol;

FoundPos = PS.SearchText("IBM", 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

The RegisterKeyEvent function registers an application-supplied object to receive notification of operator keystroke events. The application must construct an object derived from the ECLKeyNotify abstract base class. When an operator keystroke occurs, the NotifyEvent() method of the application supplied object is called. The application can choose to have the keystroke filtered or passed on and processed in the usual way. See "ECLKeyNotify Class" on page 71 for more details.

Implementation Restriction: Only one object may be registered to receive keystroke events at a time.

Prototype
void RegisterKeyEvent(ECLKeyNotify *NotifyObject)

Parameters

ECLKeyNotify *NotifyObject  Application object derived from ECLKeyNotify class.

Return Value

None

Example

The following example shows how to register an application-supplied object to receive notification of operator keystroke events. See the "ECLKeyNotify Class" on page 71 for a RegisterKeyEvent example.

// 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

The UnregisterKeyEvent method unregisters an application object previously registered for keystroke events with the RegisterKeyEvent function. A registered application notify object should not be destroyed without first calling this function to unregister it. If there is no notify object currently registered, or the registered object is not the NotifyObject passed in, this function does nothing (no error is thrown).

Prototype

virtual UnregisterKeyEvent(ECLKeyNotify *NotifyObject )

Parameters

ECLKeyNotify *NotifyObject Object currently registered for keystroke events.

Return Value

None

Example

See the "ECLKeyNotify Class" on page 71 for a UnregisterKeyEvent example.

GetFieldList

This method returns a pointer to an ECLFieldList object. The field list object can be used to iterate over the list of fields in the host presentation space. The ECLFieldList object returned by this function is automatically destroyed when the ECLPS object is destroyed. See "ECLFieldList Class" on page 65 for more information about this object.
Prototype
ECLFieldList *GetFieldList()

Parameters
None

Return Value
ECLFieldList * Pointer to ECLFieldList object.

Example
The following example shows how to return a pointer to an ECLFieldList object.

```cpp
// 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
The WaitForCursor method waits for the cursor in the presentation space of the connection associated with the ECLPS object to be located at a specified position.

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

Parameters
int Row Row position of the cursor. If negative, this value indicates the Row position from the bottom of the PS.
int Col Column position of the cursor. If negative, this value indicates the Cursor position from the edge of the PS.
long nTimeOut The maximum length of time in milliseconds to wait. This parameter is optional. The default is INFINITE.
BOOL bWaitForIR If this value is true, after meeting the wait condition the function will wait until the OIA
indicates the PS is ready to accept input. This parameter is optional and is defaulted to TRUE.

**Return Value**
The method returns TRUE if the condition is met, or FALSE if nTimeOut (in milliseconds) has elapsed.

**Note:** This method will block if nTimeOut is default value (INFINITE) when the test condition would return FALSE.

**Example**
```cpp
// 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**
The WaitWhileCursor method waits while the cursor in the presentation space of the connection associated with the ECLPS object is located at a specified position.

**Prototype**
```cpp
BOOL WaitWhileCursor(int Row, int Col, long nTimeOut=INFINITE, BOOL bWaitForIR=TRUE)
```

**Parameters**

- **int Row**
  Row position of the cursor. If negative, this value indicates the Row position from the bottom of the PS.

- **int Col**
  Column position of the cursor. If negative, this value indicates the Cursor position from the edge of the PS.

- **long nTimeOut**
  The maximum length of time in milliseconds to wait. This parameter is optional. The default is INFINITE.

- **BOOL bWaitForIR**
  If this value is true, after meeting the wait condition the function will wait until the OIA indicates the PS is ready to accept input. This parameter is optional and is defaulted to TRUE.

**Return Value**
The method returns TRUE if the condition is met, or FALSE if nTimeOut (in milliseconds) has elapsed.

**Note:** This method will block if nTimeOut is default value (INFINITE) when the test condition would return FALSE.

**Example**
```cpp
// 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
The WaitForString method waits for the specified string to appear in the presentation space of the connection associated with the ECLPS object. If the optional Row and Column parameters are used, the string must begin at the specified position. If 0,0 are passed for Row,Col the method searches the entire PS.

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

Parameters
char* WaitString The string which will be the subject of the wait.
int Row Row position of the cursor. If negative, this value indicates the Row position from the bottom of the PS. The default is zero.
int Col Column position of the cursor. If negative, this value indicates the Cursor position from the edge of the PS. The default is zero.
long nTimeOut The maximum length of time in milliseconds to wait. This parameter is optional. The default is INFINITE.
BOOL bWaitForIR If this value is true, after meeting the wait condition the function will wait until the OIA indicates the PS is ready to accept input. This parameter is optional and is defaulted to TRUE.
BOOL bCaseSens If this value is True, the wait condition is verified as case-sensitive. This parameter is optional. The default is TRUE.

Return Value
The method returns TRUE if the condition is met, or FALSE if nTimeOut (in milliseconds) has elapsed.

Note: This method will block if nTimeOut is default value (INFINITE) when the test condition would return FALSE.

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

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

// do the processing for the screen
**WaitWhileString**

The WaitWhileString method waits while the specified string is in the presentation space of the connection associated with the ECLPS object. If the optional Row and Column parameters are used, the string must begin at the specified position. If 0,0 are passed for Row,Col the method searches the entire PS.

**Prototype**

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

**Parameters**

- **char* WaitString**
  The string which will be the subject of the wait.

- **int Row**
  Start Row position of the string. If negative, this value indicates the Row position from the bottom of the PS. The default is zero.

- **int Col**
  Start Column position of the string. If negative, this value indicates the Cursor position from the edge of the PS. The default is zero.

- **long nTimeOut**
  The maximum length of time in milliseconds to wait. This parameter is optional. The default is INFINITE.

- **BOOL bWaitForIR**
  If this value is true, after meeting the wait condition the function will wait until the OIA indicates the PS is ready to accept input. This parameter is optional and is defaulted to TRUE.

- **BOOL bCaseSens**
  If this value is True, the wait condition is verified as case-sensitive. This parameter is optional. The default is TRUE.

**Return Value**

The method returns TRUE if the condition is met, or FALSE if nTimeOut (in milliseconds) has elapsed.

**Note:** This method will block if nTimeOut is default value (INFINITE) when the test condition would return FALSE.

**Example**

```cpp
// 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**

The WaitForStringInRect method waits for the specified string to appear in the presentation space of the connection associated with the ECLPS object in the specified Rectangle.
Prototype

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

Parameters

char* WaitString  The string which will be the subject of the wait.
int Row           Start Row position of the rectangle.
int Col           Start Column position of the rectangle.
int eRow          Ending row position of the search rectangle.
int eCol          Ending column position of the search rectangle.
long nTimeOut     The maximum length of time in milliseconds to wait. This parameter is optional. The default is INFINITE.
BOOL bWaitForIR   If this value is true, after meeting the wait condition the function will wait until the OIA indicates the PS is ready to accept input. This parameter is optional and is defaulted to TRUE.
BOOL bCaseSens    If this value is True, the wait condition is verified as case-sensitive. This parameter is optional. The default is TRUE.

Return Value

The method returns TRUE if the condition is met, or FALSE if nTimeOut (in milliseconds) has elapsed.

Note: This method will block if nTimeOut is default value (INFINITE) when the test condition would return FALSE.

Example

// 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

The WaitForStringInRect method waits while the specified string is in the presentation space of the connection associated with the ECLPS object in the specified Rectangle.

Prototype

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

Parameters

char* WaitString  The string which will be the subject of the wait.
int Row           Start Row position of the rectangle.
int Col                  Start Column position of the rectangle.
int eRow                 Ending row position of the search rectangle.
int eCol                 Ending column position of the search rectangle.
long nTimeOut            The maximum length of time in milliseconds to wait. This parameter is optional. The default is INFINITE.

BOOL bWaitForIR          If this value is true, after meeting the wait condition the function will wait until the OIA indicates the PS is ready to accept input. This parameter is optional and is defaulted to TRUE.

BOOL bCaseSens           If this value is True, the wait condition is verified as case-sensitive. This parameter is optional. The default is TRUE.

**Return Value**
The method returns TRUE if the condition is met, or FALSE if nTimeOut (in milliseconds) has elapsed.

**Note:** This method will block if nTimeOut is default value (INFINITE) when the test condition would return FALSE.

**Example**
// 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**
The WaitForAttrib method will wait until the specified Attribute value appears in the presentation space of the connection associated with the ECLPS object at the specified Row/Column position. The optional MaskData parameter can be used to control which values of the attribute you are looking for. The optional plane parameter allows you to select any of the four PS planes.

**Prototype**

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

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int Row</td>
<td>Row position of the attribute.</td>
</tr>
<tr>
<td>int Col</td>
<td>Column position of the attribute.</td>
</tr>
<tr>
<td>unsigned char AttribDatum</td>
<td>The 1 byte HEX value of the attribute to wait for.</td>
</tr>
<tr>
<td>unsigned char MskDatum</td>
<td>The 1 byte HEX value to use as a mask with the attribute. This parameter is optional. The default value is 0xFF.</td>
</tr>
<tr>
<td>PS_PLANE plane</td>
<td>The plane of the attribute to get. The plane can</td>
</tr>
</tbody>
</table>
have the following values: **TextPlane, ColorPlane, FieldPlane, and ExfieldPlane.** See [Appendix B, “ECL Planes — Format and Content,” on page 401](#) for the content and format of the different ECL planes.

This parameter is optional. The default is FieldPlane.

```
long nTimeOut
```

The maximum length of time in milliseconds to wait. This parameter is optional. The default is INFINITE.

```
BOOL bWaitForIR
```

If this value is true, after meeting the wait condition the function will wait until the OIA indicates the PS is ready to accept input. This parameter is optional and is defaulted to TRUE.

**Return Value**

The method returns TRUE if the condition is met, or FALSE if nTimeOut (in milliseconds) has elapsed.

**Note:** This method will block if nTimeOut is default value (INFINITE) when the test condition would return FALSE.

**Example**

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

// do the wait
BOOL waitOK = ps.WaitForAttrib(10, 16, 0x00, 0xFF, FieldPlane, INFINITE, FALSE);

// do the processing for when the screen goes away
```

**WaitWhileAttrib**

The WaitWhileAttrib method waits while the specified Attribute value appears in the presentation space of the connection associated with the ECLPS object at the specified Row/Column position. The optional MaskData parameter can be used to control which values of the attribute you are looking for. The optional plane parameter allows you to select any of the four PS planes.

**Prototype**

```
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
```

Row position of the attribute.

```
int Col
```

Column position of the attribute unsigned.

```
char AttribDatum
```

The 1 byte HEX value of the attribute to wait for.

```
unsigned char MskDatum
```

The 1 byte HEX value to use as a mask with the attribute. This parameter is optional. The default value is 0xFF.
**PS_PLANE plane**
The plane of the attribute to get. The plane can have the following values: **TextPlane**, **ColorPlane**, **FieldPlane**, and **ExfieldPlane**. See [Appendix B., “ECL Planes — Format and Content,” on page 401](#) for the content and format of the different ECL planes.

This parameter is optional. The default is FieldPlane.

**long nTimeOut**
The maximum length of time in milliseconds to wait. This parameter is optional. The default is INFINITE.

**BOOL bWaitForIR**
If this value is true, after meeting the wait condition the function will wait until the OIA indicates the PS is ready to accept input. This parameter is optional and is defaulted to TRUE.

**Return Value**
The method returns TRUE if the condition is met, or FALSE if nTimeOut (in milliseconds) has elapsed.

**Note:** This method will block if nTimeOut is default value (INFINITE) when the test condition would return FALSE.

**Example**
```
// 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**
Synchronously waits for the screen described by the ECLScreenDesc parameter to appear in the Presentation Space.

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

**Parameters**
- **ECLScreenDesc**
  screenDesc Object that describes the screen (see “ECLScreenDesc Class” on page 135).
- **long nTimeOut**
The maximum length of time in milliseconds to wait. This parameter is optional. The default is INFINITE.

**Return Value**
The method returns TRUE if the condition is met, or FALSE if nTimeOut (in milliseconds) has elapsed.

**Note:** This method will block if nTimeOut is default value (INFINITE) when the test condition would return FALSE.
ECLPS

Example
// 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
Synchronously waits until the screen described by the ECLScreenDesc parameter is no longer in the Presentation Space.

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

Parameters
ECLScreenDesc screenDesc Object that describes the screen (see “ECLScreenDesc Methods” on page 135).

long nTimeOut The maximum length of time in milliseconds to wait. This parameter is optional. The default is INFINITE.

Return Value
The method returns TRUE if the condition is met, or FALSE if nTimeOut (in milliseconds) has elapsed.

Note: This method will block if nTimeOut is default value (INFINITE) when the test condition would return FALSE.

Example
// 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
This member function registers an application object to receive notifications of PS update events. To use this function the application must create an object derived from either ECLPSNotify or ECLPSListener. A pointer to that object is then passed
to this registration function. Any number of notify or listener objects may be registered at the same time. The order in which multiple listeners receive events is not defined and should not be assumed.

Different prototypes for this function allow different types of update events to be generated, and different levels of detail about the updates. The simplest update event is registered with an ECLPSNotify object. The type of registration produces an event for every PS update. No information about the update is generated. See the description of the ECLPSNotify object for more information.

For applications with need more information about the update, the ECLPSListener object can be registered. Registration of this object gives the application the ability to ignore some types of updates (for example, local terminal functions such as keystrokes) and to determine the region of the screen which was updated. See the description of the ECLPSListener object for more information. When registering an ECLPSListener object, the application can optionally specify the type of updates which are to cause events.

After an ECLPSNotify or ECLPSListener object is registered with this function, it’s NotifyEvent() method will be called whenever a update to the presentation space occurs. Multiple updates to the PS in a short time period may be aggregated into a single event.

The application must unregister the notify/listener object before destroying it. The object will automatically be unregistered if the ECLPS object is destroyed.

**Prototype**

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

**Parameters**

- `ECLPSNotify *` Pointer to the ECLPSNotify object to be registered.
- `ECLPSListener *` Pointer to the ECLPSListener object to be registered.
- `int` Type of updates which will cause events:
  - `USER_EVENTS` (local terminal functions)
  - `HOST_EVENTS` (host updates)
  - `ALL_EVENTS` (all updates)

**Return Value**

None

**StartMacro**

The StartMacro method runs the Personal Communications macro file indicated by the MacroName parameter.

**Prototype**

```c++
void StartMacro(String MacroName)
```

**Parameters**

- `String MacroName` Name of macro file located in the Personal Communications user-class application data
directory (specified at installation), without the file extension. This method does not support long file names.

**Return Value**
None

**Usage Notes**
You must use the short file name for the macro name. This method does not support long file names.

**Example**
The following example shows how to start a macro.
Dim PS as Object
Set PS = CreateObject("PCOMM.autECLPS")
PS.StartMacro "mymacro"

### UnregisterPSEvent
This member function unregisters an application object previously registered with the RegisterPSEvent function. An object registered to receive events should not be destroyed without first calling this function to unregister it. If the specific object is not currently registered, no action is taken and no error occurs.

When an ECLPSNotify or ECLPSListener object is unregistered its NotifyStop() method is called.

**Prototype**
void UnregisterPSEvent(ECLPSNotify * notify)
void UnregisterPSEvent(ECLPSListener * listener)
void UnregisterPSEvent(ECLPSListener * listener, int type)

**Parameters**
- **ECLPSNotify ***
  Pointer to the ECLPSNotify object to be unregistered.
- **ECLPSListener ***
  Pointer to the ECLPSListener object to be unregistered.
- **int**
  Type of updates which where registered:
  - USER_EVENTS (local terminal functions)
  - HOST_EVENTS (host updates)
  - ALL_EVENTS (all updates)

**Return Value**
None

### ECLPSEvent Class
ECLPSEvent objects are passed to ECLListener objects when the presentation space has been updated. This event object represents the presentation space update event and contains information about the update.

There are two sets of functions an application can use to determine the region of the presentation space which was updated. The GetStart() and GetEnd() methods return a linear position indicating the starting position and ending position of the
update region in the presentation space. Linear addressing starts at 1 for the upper-left-most character and proceeds left-to-right wrapping from row to row. A corresponding set of functions (GetStartRow, GetStartCol, GetEndRow, GetEndCol) return the same information in row/column coordinates.

The update region includes all PS characters from the starting character to the ending character (inclusive). If the start and end position are not on the same row then the update region wraps from the end of one row to the first column of the next row. Note that the update region is (generally) not rectangular. If the starting position is greater than the ending position, the update region starts at the starting position, wraps from the last character of the screen to the first, and continues to the ending position.

Note that the update region may encompass more than the actual changed portion of the presentation space, but it is guaranteed to cover at least the changed area. When multiple PS updates occur in a short period of time the changes may be aggregated into a single event in which the update region spans the sum of all the updates.

Derivation
ECLBase > ECLEvent > ECLPSEvent

Usage Notes
Applications do not use this class directly. Applications create ECLListener-derived objects which receive ECLPSEvent objects on the ECLListener::NotifyEvent method.

ECLPSEvent Methods
The following section describes the methods that are valid for the ECLPSEvent class and all classes derived from it.

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

GetPS
This method returns the ECLPS object which generated this event.

Prototype
ECLPS * GetPS()

Parameters
None

Return Value
ECLPS * Pointer to ECLPS object which generated the event.
GetType
This method returns the type of presentation space update which generated this event. The return value is on of USER_EVENTS or HOST_EVENTS. User events are defined as any PS update which occurs as a local terminal function (for example, keystrokes entered by the user or by a programming API). Host events are PS updates which occur from host outbound datastreams.

Prototype
int GetType()

Parameters
None

Return Value
int
Returns USER_EVENTS or HOST_EVENTS constants.

GetStart
This method returns the linear location in the presentation space of the start of the update region. Note that the row/column coordinate of this location is dependant on the number of columns currently defined for the presentation space. If this value is greater than that returned by GetEnd(), then the update region starts at this location, wraps at the end of the screen to the beginning of the screen, and continues to the ending position.

Prototype
ULONG GetStart()

Parameters
None

Return Value
ULONG
Linear position of start of the update region.

GetEnd
This method returns the linear location in the presentation space of the end of the update region. Note that the row/column coordinate of this location is dependant on the number of columns currently defined for the presentation space. If this value is less than that returned by GetStart(), then the update region starts at the GetStart() location, wraps at the end of the screen to the beginning of the screen, and continues to this position.

Prototype
ULONG GetEnd()

Parameters
None

Return Value
ULONG
Linear position of end of the update region.

GetStartRow
This method returns the row number in the presentation space of the start of the update region. If the starting row/column position is greater than that of the
ending row/column position, then the update region starts at this location, wraps at the end of the screen to the beginning of the screen, and continues to the ending position.

**Prototype**
ULONG GetStartRow()

**Parameters**
None

**Return Value**
ULONG
Row number of start of the update region.

**GetStartCol**
This method returns the column number in the presentation space of the start of the update region. If the starting row/column position is greater than that of the ending row/column position, then the update region starts at the starting row/column, wraps at the end of the screen to the beginning of the screen, and continues to the ending position.

**Prototype**
ULONG GetStartCol()

**Parameters**
None

**Return Value**
ULONG
Column number of start of the update region.

**GetEndRow**
This method returns the row number in the presentation space of the end of the update region. If the starting row/column position is greater than that of the ending row/column position, then the update region starts at the starting row/column, wraps at the end of the screen to the beginning of the screen, and continues to the ending row/column.

**Prototype**
ULONG GetEndRow()

**Parameters**
None

**Return Value**
ULONG
Row number of end of the update region.

**GetEndCol**
This method returns the column number in the presentation space of the end of the update region. If the starting row/column position is greater than that of the ending row/column position, then the update region starts at the starting row/column, wraps at the end of the screen to the beginning of the screen, and continues to the ending row/column.

**Prototype**
ULONG GetEndCol()
### Parameters

None

### Return Value

ULONG  
Column number of end of the update region.

---

**ECLPSListener Class**

ECLPSListener is an abstract base class. An application cannot create an instance of this class directly. To use this class, the application must define its own class which is derived from ECLPSListener. The application must implement all the methods in this class.

The ECLPSListener class is used to allow an application to be notified of updates to the presentation space. Events are generated whenever the host screen is updated (any data in the presentation space is changed in any plane).

This class is similar to the ECLPSNotify class in that it is used to receive notifications of PS updates. It differs however in that it receives much more information about the cause and scope of the update than the ECLPSNotify class.

In general using this class will be more expensive in terms of processing time and memory since more information has to be generated with each event. For applications which need to efficiently update a visual representation of the host screen this class may be more efficient than redrawing the representation each time an update occurs. Using this class the application can update only the portion of the visual representation that has changed.

This class also differs from ECLPSNotify in that all the methods are pure virtual and therefore must be implemented by the application (there is no default implementation of any methods).

#### Derivation

ECLBase > ECLListener > ECLPSListener

#### Usage Notes

To be notified of PS updates using this class, the application must perform the following steps:

1. Define a class derived from ECLPSListener.
2. Implement all methods of the ECLPSListener-derived class.
3. Create an instance of the derived class.
4. Register the instance with the ECLPS::RegisterPSEvent() method.

After registration is complete, updates to the presentation space will cause the NotifyEvent() method of the ECLPSListener-derived class to be called. The application can then use the ECLPSEvent object supplied on the method call to determine what caused the PS update and the region of the screen affected.

Note that multiple PS updates which occurred in a short period of time may be aggregated into a single event notification.
An application can choose to provide its own constructor and destructor for the derived class. This can be useful if the application needs to store some instance-specific data in the class and pass that information as a parameter on the constructor.

If an error is detected during event registration, the NotifyError() member function is called with an ECLErr object. Events may or may not continue to be generated after an error. When event generation terminates (due to an error or some other reason) the NotifyStop() member function is called.

### ECLPSListener Methods

The following section describes the methods that are valid for the ECLPSListener class and all classes derived from it. Note that all methods except the constructor and destructor are pure virtual methods.

```cpp
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**

This method is a *pure virtual* member function (the application *must* implement this function in classes derived from ECLPSListener). This method is called whenever the presentation space is updated and this object is registered to receive update events. The ECLPSEvent object passed as a parameter contains information about the event including the region of the screen that was modified. See [“ECLPSEvent Class” on page 124](#) for details.

Multiple PS updates may be aggregated into a single event causing only a single call to this method. The changed region contained in the ECLPSEvent object will encompass the sum of all the modifications.

Events may be restricted to only a particular type of PS update by supplying the appropriate parameters on the ECLPS::RegisterPSEvent() method. For example the application may choose to be notified only for updates from the host and not for local keystrokes.

**Prototype**

```cpp
virtual void NotifyEvent(ECLPSEvent * event) = 0
```

**Parameters**

- **ECLPSEvent** * Pointer to an ECLPSEvent object which represents the PS update.

**Return Value**

None

**NotifyError**

This method is called whenever the ECLPS object detects an error during event generation. The error object contains information about the error (see [“ECLErr Class” on page 48](#)). Events may continue to be generated after the error depending on the nature of the error. If the event generation stops due to an error, the NotifyStop() method is called.
This is a *pure virtual* method which the application must implement.

**Prototype**

```c
virtual void NotifyError(ECLPS * PObj, ECLErr ErrObj) = 0
```

**Parameters**

- **ECLPS**
  - Pointer to the ECLPS object which generated this event.

- **ECLErr**
  - An ECLErr object which describes the error.

**Return Value**

None

**NotifyStop**

This method is called when event generation is stopped for any reason (for example, due to an error condition or a call to ECLPS::UnregisterPSEvent).

This is a *pure virtual* method which the application must implement.

The reason code parameter is currently unused and will be zero.

**Prototype**

```c
virtual void NotifyStop(ECLPS * PObj, int Reason) = 0
```

**Parameters**

- **ECLPS**
  - Pointer to the ECLPS object which generated this event.

- **int**
  - Reason event generation has stopped (currently unused and will be zero).

**Return Value**

None

---

**ECLPSNotify Class**

ECLPSNotify is an abstract base class. An application cannot create an instance of this class directly. To use this class, the application must define its own class which is derived from ECLPSNotify. The application must implement the NotifyEvent() member function in its derived class. It may also optionally implement NotifyError() and NotifyStop() member functions.

The ECLPSNotify class is used to allow an application to be notified of updates to the presentation space. Events are generated whenever the host screen is updated (any data in the presentation space is changed in any plane).

This class is similar to the ECLPSListener class in that it is used to receive notifications of PS updates. It differs however in that it receives no information about the cause and scope of the update than the ECLPSNotify class. In general using this class will be more efficient in terms of processing time and memory since no information has to be generated with each event. This class may be used for applications which only need notification of updates and do not need the details of what caused the event or what part of the screen was updated.
This class also differs from ECLPSListener in that default implementations are provided for the NotifyError() and NotifyStop() methods.

**Derivation**

ECLBase > ECLNotify > ECLPSNotify

**Usage Notes**

To be notified of PS updates using this class, the application must perform the following steps:

1. Define a class derived from ECLPSNotify.
2. Implement the NotifyEvent method of the ECLPSNotify-derived class.
3. Optionally implement other member functions of ECLPSNotify.
4. Create an instance of the derived class.
5. Register the instance with the ECLPS::RegisterPSEvent() method.

After registration is complete, updates to the presentation space will cause the NotifyEvent() method of the ECLPSNotify-derived class to be called.

Note that multiple PS updates which occur in a short period of time may be aggregated into a single event notification.

An application can choose to provide its own constructor and destructor for the derived class. This can be useful if the application needs to store some instance-specific data in the class and pass that information as a parameter on the constructor.

If an error is detected during event registration, the NotifyError() member function is called with an ECLErr object. Events may or may not continue to be generated after an error. When event generation terminates (due to an error or some other reason) the NotifyStop() member function is called. The default implementation of NotifyError() will present a message box to the user showing the text of the error messages retrieved from the ECLErr object.

When event notification stops for any reason (error or a call the ECLPS::UnregisterPSEvent) the NotifyStop() member function is called. The default implementation of NotifyStop() does nothing.

**ECLPSNotify Methods**

The following section describes the methods that are valid for the ECLPSNotify class and all classes derived from it.

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

**NotifyEvent**

This method is a *pure virtual* member function (the application *must* implement this function in classes derived from ECLPSNotify). This method is called whenever the presentation space is updated and this object is registered to receive update events.
Multiple PS updates may be aggregated into a single event causing only a single call to this method.

**Prototype**

virtual void NotifyEvent(ECLPS * PSObj)

**Parameters**

ECLPS * Pointer to the ECLPS object which generated this event.

**Return Value**

None

**NotifyError**

This method is called whenever the ECLPS object detects an error during event generation. The error object contains information about the error (see “ECLErr Class” on page 48). Events may continue to be generated after the error depending on the nature of the error. If the event generation stops due to an error, the NotifyStop() method is called.

An application can choose to implement this function or allow the base ECLPSNotify class handle it. The default implementation will display the error in a message box using text supplied by the ECLErr::GetMsgText() method. If the application implements this function in its derived class it overrides this behavior.

**Prototype**

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

**Parameters**

ECLPS * Pointer to the ECLPS object which generated this event.

ECLErr An ECLErr object which describes the error.

**Return Value**

None

**NotifyStop**

This method is called when event generation is stopped for any reason (for example, due to an error condition or a call to ECLPS::UnregisterPSEvent).

The reason code parameter is currently unused and will be zero.

The default implementation of this function does nothing.

**Prototype**

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

**Parameters**

ECLPS * Pointer to the ECLPS object which generated this event.

int Reason event generation has stopped (currently unused and will be zero).
ECLRecoNotify Class

ECLRecoNotify can be used to implement an object which will receive and handle ECLScreenReco events. Events are generated whenever any screen in the PS is matched to an ECLScreenDesc object in ECLScreenReco. Special events are generated when event generation stops and when errors occur during event generation.

To be notified of ECLScreenReco events, the application must perform the following steps:
1. Define a class which derives from the ECLRecoNotify class.
2. Implement the NotifyEvent(), NotifyStop(), and NotifyError() methods.
3. Create an instance of the new class.
4. Register the instance with the ECLScreenReco::RegisterScreen() method.

See "ECLScreenReco Class" on page 141 for an example.

Derivation

ECLBase > ECLNotify > ECLRecoNotify

ECLRecoNotify Methods

Valid methods for ECLRecoNotify are listed below:

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

ECLRecoNotify Constructor

Creates an empty instance of ECLRecoNotify.

Prototype
ECLRecoNotify()

Parameters
None

Return Value
None

Example
See "ECLScreenReco Class" on page 141 for an example.

ECLRecoNotify Destructor

Destroys the instance of ECLRecoNotify

Prototype
~ECLRecoNotify()
ECLRecoNotify

Parameters
None

Return Value
None

Example
See “ECLScreenReco Class” on page 141 for an example.

NotifyEvent
Called when the ECLScreenDesc registered with the ECLRecoNotify object on ECLScreenReco appears in the presentation space.

Prototype
void NotifyEvent(ECLPS *ps, ECLScreenDesc *sd)

Parameters
ECLPS ps The ECLPS object that you registered.
ECLScreenDesc sd ECLScreenDesc that you registered.

Return Value
None

Example
See “ECLScreenReco Class” on page 141 for an example.

NotifyStop
Called when the ECLScreenReco object stops monitoring its ECLPS objects for the registered ECLScreenDesc objects.

Prototype
void NotifyStop(ECLPS *ps, ECLScreenDesc *sd)

Parameters
ECLPS ps The ECLPS object that you registered.
ECLScreenDesc sd ECLScreenDesc that you registered.

Return Value
None

Example
See “ECLScreenReco Class” on page 141 for an example.

NotifyError
Called when the ECLScreenReco object encounters an error.

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

Parameters
ECLPS ps The ECLPS object that you registered.
ECLScreenDesc sd ECLScreenDesc that you registered.
ECLErr e

ECLErr object that contains the error information.

Return Value
None

Example
See “ECLScreenReco Class” on page 141 for an example.

ECLScreenDesc Class

ECLScreenDesc is the class that is used to describe a screen for the IBM Host Access Class Library screen recognition technology. It uses all four major planes of the presentation space to describe it (TEXT, FIELD, EXFIELD, COLOR), as well as the cursor position.

Using the methods provided on this object, the programmer can set up a detailed description of what a given screen looks like in a host side application. Once an ECLScreenDesc object is created and set, it may be passed to either the synchronous WaitFor... methods provided on ECLPS, or it may be passed to ECLScreenReco, which fires an asynchronous event if the screen matching the ECLScreenDesc object appears in the PS.

Derivation
ECLBase > ECLScreenDesc

ECLScreenDesc Methods

Valid methods for ECLScreenDesc are listed below:

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 Constructor

Creates an empty instance of ECLScreenDesc.

Prototype
ECLScreenDesc()

Parameters
None

Return Value
None
**ECLScreenDesc**

**Example**

```
// 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 Destructor**

Destroys the instance of ECLScreenDesc.

**Prototype**

```
~ ECLScreenDesc()
```

**Parameters**

None

**Return Value**

None

**Example**

```
// 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**

Adds an attribute value at the given position to the screen description.

**Prototype**

```
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**: Attribute value to add.
- **int row**: Row position.
- **int col**: Column position.
FieldPlane, and ExfieldPlane. See Appendix B, “ECL Planes — Format and Content,” on page 401 for the content and format of the different ECL planes.

Return Value

None

Example

// 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

Sets the cursor position for the screen description to the given position.

Prototype

void AddCursorPos(uint row, uint col)

Parameters

uint row Row position.

uint col Column position.

Return Value

None

Example

// 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());
ECLScreenDesc

**AddNumFields**

Adds the number of input fields to the screen description.

**Prototype**

```c
void AddNumFields(uint num)
```

**Parameters**

- `uint num`: Number of fields.

**Return Value**

None

**Example**

```c
// 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**

Adds the number of input fields to the screen description.

**Prototype**

```c
void AddNumInputFields(uint num)
```

**Parameters**

- `uint num`: Number of input fields.

**Return Value**

None

**Example**

```c
// 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
```
AddOIAlnhibitStatus

Sets the type of OIA monitoring for the screen description.

Prototype
void AddOIAlnhibitStatus(OIASstatus type=NOTINHIBITED)

Parameters
OIASstatus type Type of OIA status. Current valid values are
DONTCARE and NOTINHIBITED.

Return Value
None

Example
// 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 = ec1PS.WaitForScreen(ec1SD, timeInt.intValue());

AddString

Adds a string at the given location to the screen description. If row and column
are not provided, string may appear anywhere in the PS.

Note: Negative values are absolute positions from the bottom of the PS. For
example, row=-2 is row 23 out of 24 rows.

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

Parameters
LPCSTR s String to add.
uint row Row position.
uint col Column position.
BOOL caseSense If this value is TRUE, the strings are added as
case-sensitive. This parameter is optional. The
default is TRUE.

Return Value
None
ECLScreenDesc

Example
// 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
Adds a string in the given rectangle to the screen description.

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

Parameters
char * str            String to add.
int Top               Upper left row position. This parameter is optional.
                       The default is the first row.
int Left              Upper left column position. This parameter is optional. The default is the first column.
int Bottom            Lower right row position. This parameter is optional. The default is the last row.
int Right             Lower right column position. This parameter is optional. The default is the last column.
BOOL caseSense        If this value is TRUE, the strings are added as case-sensitive. This parameter is optional. The default is TRUE.

Return Value
None

Example
// 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(ecISD, timeInt.intValue());

Clear

Removes all description elements from the screen description.

Prototype

void Clear()

Parameters

None

Return Value

None

Example

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

// set up screen description
ECLScreenDesc ecISD = new ECLScreenDesc();
ecISD.AddAttrib(0xe8, 1, 1, ColorPlane);
ecISD.AddCursorPos(23, 1);
ecISD.AddNumFields(45);
ecISD.AddNumInputFields(17);
AddOIAInhibitStatus(NOTINHIBITED);
ecISD.AddString("LOGON\n", 23, 11, TRUE);
ecISD.AddStringInRect("PASSWORD", 23, 1, 24, 80, FALSE);

// do the wait
int TimeOut = 5000;
BOOL waitOK = eclPS.WaitForScreen(ecISD, timeInt.intValue());

// do processing for the screen
ecISD.Clear() // start over for a new screen

ECLScreenReco Class

The ECLScreenReco class is the engine for the Host Access Class Library screen recognition system. It contains the methods for adding and removing descriptions of screens. It also contains the logic for recognizing those screens and for asynchronously calling back to your handler code for those screens.

Think of an object of the ECLScreenReco class as a unique recognition set. The object can have multiple ECLPS objects that it watches for screens, multiple screens to look for, and multiple callback points to call when it sees a screen in any of the ECLPS objects.

All you need to do is set up your ECLScreenReco objects at the start of your application, and when any screen appears in any ECLPS that you want to monitor, your code will get called by ECLScreenReco. You do absolutely no legwork in monitoring screens!

Here’s an example of a common implementation:
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();
    }
}

**Derivation**

ECLBase > ECLScreenReco

**ECLScreenReco Methods**

The following methods are valid for ECLScreenReco:
ECLScreenReco

ECLScreenReco()
~ECLScreenReco()
AddPS(ECLPS*)
IsMatch(ECLPS*, ECLScreenDesc*)
RegisterScreen(ECLScreenDesc*, ECLRecoNotify*)
RemovePS(ECLPS*)
UnregisterScreen(ECLScreenDesc*)

**ECLScreenReco Constructor**

Creates an empty instance of ECLScreenReco

**Prototype**
ECLScreenReco()

**Parameters**
None

**Return Value**
None

**Example**
See the example of a common implementation provided in “ECLScreenReco Class” on page 141.

**ECLScreenReco Destructor**

Destroys the instance of ECLScreenReco

**Prototype**
~ECLScreenReco()

**Parameters**
None

**Return Value**
None

**Example**
See the example of a common implementation provided in “ECLScreenReco Class” on page 141.

**AddPS**

Adds Presentation Space object to monitor.

**Prototype**
AddPS(ECLPS*)

**Parameters**
ECLPS* PS object to monitor.

**Return Value**
None

**Example**
See the example of a common implementation provided in “ECLScreenReco Class” on page 141.
ECLScreenReco

**IsMatch**
Static member method that allows for passing an ECLPS object and an ECLScreenDesc object and determining if the screen description matches the PS. It is provided as a static method so any routine can call it without creating an ECLScreenReco object.

**Prototype**
IsMatch(ECLPS*, ECLScreenDesc*)

**Parameters**
- ECLPS* ECLPS object to compare.
- ECLScreenDesc* ECLScreenDesc object to compare.

**Return Value**
TRUE if the screen in PS matches, FALSE otherwise.

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

// set up screen description
ECLScreenDesc ec1SD = new ECLScreenDesc();
ec1SD.AddAttrib(0xe8, 1, 1, ColorPlane);
ec1SD.AddCursorPos(23, 1);
ec1SD.AddNumFields(45);
ec1SD.AddNumInputFields(17);
Add0IAInhibitStatus(NOTINHIBITED);
ec1SD.AddString("LOGON", 23, 11, TRUE);
ec1SD.AddStringInRect("PASSWORD", 23, 1, 24, 80, FALSE);
if(ECLScreenReco::IsMatch(ps, ec1SD)) {
    // Handle Screen Match here . . .
}
```

**RegisterScreen**
Begins monitoring all ECLPS objects added to the screen recognition object for the given screen description. If the screen appears in the PS, the NotifyEvent method on the ECLRecoNotify object will be called.

**Prototype**
RegisterScreen(ECLScreenDesc*, ECLRecoNotify*)

**Parameters**
- ECLScreenDesc* Screen description object to register.
- ECLRecoNotify* Object that contains the callback code for the screen description.

**Return Value**
None

**Example**
See the example of a common implementation provided in "ECLScreenReco Class" on page 141.

**RemovePS**
Removes the ECLPS object from screen recognition monitoring.
Prototype
RemovePS(ECLPS*)

Parameters
ECLPS* ECLPS object to remove.

Return Value
None

Example
See the example of a common implementation provided in “ECLScreenReco Class” on page 141.

UnregisterScreen
Removes the screen description and its callback code from screen recognition monitoring.

Prototype
UnregisterScreen(ECLScreenDesc*)

Parameters
ECLScreenDesc* Screen description object to remove.

Return Value
None

Example
See the example of a common implementation provided in “ECLScreenReco Class” on page 141.

ECLSession Class

ECLSession provides general emulator connection-related services and contains pointers to instances of other objects in the Host Access Class Library.

Derivation
ECLBase > ECLConnection > ECLSession

Properties
None

Usage Notes
Because ECLSession is derived from ECLConnection, you can obtain all the information contained in an ECLConnection object. See “ECLConnection Class” on page 19 for more information.

Although the objects ECLSession contains are capable of standing on their own, pointers to them exist in the ECLSession class. When an ECLSession object is created, ECLPS, ECLOIA, ECLXfer, and ECLWinMetrics objects are also created.
ECLSession Methods

The following section describes the methods that are valid for the ECLSession class:

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 Constructor

This method creates an ECLSession object from a connection name (a single, alphabetic character from A-Z) or a connection handle. There can be only one Personal Communications connection open with a given name. For example, there can only be one connection "A" open at a time.

Prototype
ECLSession(char Name)
ECLSession(long Handle)

Parameters
char Name
One-character short name of the connection (A-Z).
long Handle
Handle of an ECL connection.

Return Value
None

Example
//-------------------------------------------------------------------
// 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**

This method destroys an ECLSession object.

**Prototype**

~ECLSession();

**Parameters**

None

**Return Value**

None

**Example**

```c++
// 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**

This method returns a pointer to the ECLPS object contained in the ECLSession object. Use this method to access the ECLPS object methods. See “ECLPS Class” on page 90 for more information.

**Prototype**

ECLPS *GetPS()

**Parameters**

None

**Return Value**

ECLPS * ECLPS object pointer.

**Example**

```c++
// ECLSession::GetPS
//
// Get PS object from session object and use it.
//-------------------------------------------------------------------
void Sample69() {
    ECLSession *Sess;    // Pointer to Session object for connection A
```
ECLSession

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

This method returns a pointer to the ECLOIA object contained in the ECLSession object. Use this method to access the ECLOIA methods. See “ECLOIA Class” on page 76 for more information.

Prototype

ECLOIA *GetOIA()

Parameters

None

Return Value

ECLOIA * ECLOIA object pointer.

Example

//-------------------------------------------------------------------
// 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

This method returns a pointer to the ECLXfer object contained in the ECLSession object. Use this method to access the ECLXfer methods. See “ECLXfer Class” on page 172 for more information.
Prototype
ECLXfer *GetXfer()

Parameters
None

Return Value
ECLXfer * ECLXfer object pointer.

Example
//-------------------------------------------------------------------
// 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
This method returns a pointer to the ECLWinMetrics object contained in the
ECLSession object. Use this method to access the ECLWinMetrics methods. See
"ECLWinMetrics Class” on page 155 for more information.

Prototype
ECLWinMetrics *GetWinMetrics()

Parameters
None

Return Value
ECLWinMetrics * ECLWinMetrics object pointer.

Example
//-------------------------------------------------------------------
// 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');
GetPageSettings

This method returns a pointer to the ECLPageSettings object contained in the ECLSession object. Use this method to access the ECLPageSettings methods. See “ECLPageSettings Class” on page 177 for more information.

Prototype
ECLPageSettings *GetPageSettings() const;

Parameters
None

Return Value
ECLPageSettings *
    ECLPageSettings object pointer.

Example
//--------------------------------------------------------------------
// 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

This method returns a pointer to the ECLPrinterSettings object contained in the ECLSession object. Use this method to access the ECLPrinterSettings methods. See “ECLPageSettings Class” on page 177 for more information.

Prototype
ECLPrinterSettings *GetPrinterSettings() const;

Parameters
None
Return Value

ECLPrinterSettings *
ECLPrinterSettings object pointer.

Example

//--------------------------------------------------------------------
// 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

Deprecated. See ECLPS::RegisterPSEvent in “RegisterPSEvent” on page 122.

UnregisterUpdateEvent

Deprecated. See ECLPS::UnregisterPSEvent in “UnregisterPSEvent” on page 124.

ECLStartNotify Class

ECLStartNotify is an abstract base class. An application cannot create an instance of this class directly. To use this class, the application must define its own class which is derived from ECLStartNotify. The application must implement the NotifyEvent() member function in its derived class. It may also optionally implement NotifyError() and NotifyStop() member functions.

The ECLStartNotify class is used to allow an application to be notified of the starting and stopping of PCOMM connections. Start/stop events are generated whenever a PCOMM connection (window) is started or stopped by any means, including the ECLConnMgr start/stop methods.

To be notified of start/stop events, the application must perform the following steps:
1. Define a class derived from ECLStartNotify.
2. Implement the derived class and implement the NotifyEvent() member function.
3. Optionally implement the NotifyError() and/or NotifyStop() functions.
4. Create an instance of the derived class.
5. Register the instance with the ECLConnMgr::RegisterStartEvent() function.
The example shown demonstrates how this may be done. When the above steps are complete, each time a connection is started or stopped the application NotifyEvent() member function will be called. The function is passed two parameters giving the handle of the connection, and a BOOL start/stop indicator. The application may perform any functions required in the NotifyEvent() procedure, including calling other ECL functions. Note that the application cannot prevent the stopping of a connection; the notification is made after the session is already stopped.

If an error is detected during event generation, the NotifyError() member function is called with an ECLErr object. Events may or may not continue to be generated after an error, depending on the nature of the error. When event generation terminates (either due to an error, by calling the ECLConnMgr::UnregisterStartEvent, or by destruction of the ECLConnMgr object) the NotifyStop() member function is called. However event notification is terminated, the NotifyStop() member function is always called, and the application object is unregistered.

If the application does not provide an implementation of the NotifyError() member function, the default implementation is used (a simple message box is displayed to the user). The application can override the default behavior by implementing the NotifyError() function in the applications derived class. Likewise, the default NotifyStop() function is used if the application does not provide this function (the default behavior is to do nothing).

Note that the application can also choose to provide its own constructor and destructor for the derived class. This can be useful if the application wants to store some instance-specific data in the class and pass that information as a parameter on the constructor. For example, the application may want to post a message to an application window when a start/stop event occurs. Rather than define the window handle as a global variable (so it would be visible to the NotifyEvent() function), the application can define a constructor for the class which takes the window handle and stores it in the class member data area.

The application must not destroy the notification object while it is registered to receive events.

Implementation Restriction: Currently, the ECLConnMgr object allows only one notification object to be registered for a start/stop event notification. The ECLConnMgr::RegisterStartEvent will throw an error if a notify object is already registered for that ECLConnMgr object.

**Derivation**

ECLBase > ECLNotify > ECLStartNotify

**Example**

```cpp
// ECLStartNotify class

// This sample demonstrates the use of:

// ECLStartNotify::NotifyEvent
// ECLStartNotify::NotifyError
// ECLStartNotify::NotifyStop
// ECLConnMgr::RegisterStartEvent
// ECLConnMgr::UnregisterStartEvent
```

// ECLStartNotify class

// This sample demonstrates the use of:

// ECLStartNotify::NotifyEvent
// ECLStartNotify::NotifyError
// ECLStartNotify::NotifyStop
// ECLConnMgr::RegisterStartEvent
// ECLConnMgr::UnregisterStartEvent
```
Define a class derived from ECLStartNotify

```cpp
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;
};
```

```cpp
MyStartNotify::MyStartNotify(HANDLE DataHandle) // Constructor
{
    MyDataH = DataHandle; // Save data handle for later use
}
```

```cpp
void MyStartNotify::NotifyEvent(ECLConnMgr *CMObj, long ConnHandle, BOOL Started) // Constructor
{
    // 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;
}
```

```cpp
void Sample75() // Create the class and begin start/stop monitoring.
{
    try {
        ECLConnMgr CMgr; // Connection manager object
        MyStartNotify *Event; // Ptr to my event handling object
        HANDLE InstData; // Handle to application data block (for example)

        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);
    }
```
ECLStartNotify

// 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 Methods

The following section describes the methods that are valid for the ECLStartNotify class.

ECLStartNotify()
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

This method is a pure virtual member function (the application must implement this function in classes derived from ECLStartNotify). This function is called whenever a connection starts or stops and the object is registered for start/stop events. The Started BOOL is TRUE if the connection is started, or FALSE if it is stopped.

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

Parameters

ECLConnMgr *CMObj This is the pointer to ECLConnMgr object in which the event occurred.
long ConnHandle This is the handle of the connection that started or stopped.
BOOL Started This is TRUE if the connection is started, or FALSE if the connection is stopped.

Return Value
None

NotifyError

This method is called whenever the ECLConnMgr object detects an error event generation. The error object contains information about the error (see the ECLErr class description). Events may continue to be generated after the error, depending on the nature of the error. If event generation stops due to an error, the NotifyStop() function is called.
The ConnHandle contains the handle of the connection that is related to the error. This value may be zero if the error is not related to any specific connection.

An application can choose to implement this function or allow the ECLStartNotify base class to handle the error. The base class will display the error in a message box using the text supplied by the ECLErr::GetMsgText() function. If the application implements this function in its derived class it will override the base class function.

**Prototype**

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

**Parameters**

- **ECLConnMgr *CMObj**
  - This is the ptr to ECLConnMgr object in which the error occurred.

- **long ConnHandle**
  - This is the handle of the connection related to the error or zero.

- **ECLErr ErrObject**
  - This is the ECLErr object describing the error.

**Return Value**

None

**NotifyStop**

This method is called when event generation is stopped for any reason (for example, due to an error condition or a call to ECLConnMgr::UnregisterStartEvent).

**Prototype**

```cpp
virtual void NotifyStop (ECLConnMgr *CMObj int Reason)
```

**Parameters**

- **ECLConnMgr *CMObj**
  - This is the ptr to ECLConnMgr object that is stopping notification.

- **int Reason**
  - This is the unused zero.

**Return Value**

None

---

**ECLUpdateNotify Class**

*Deprecated.* See the class descriptions in "ECLPSListener Class" on page 128 and "ECLOIA Class" on page 76.

---

**ECLWinMetrics Class**

The ECLWinMetrics class performs operations on a Personal Communications connection window. It allows you to perform window rectangle and position manipulation (for example, SetWindowRect,GetXpos or SetWidth), as well as window state manipulation (for example, SetVisible or IsRestored).

**Derivation**

```
ECLBase > ECLConnection > ECLWinMetrics
```
Properties

None

Usage Notes

Because ECLWinMetrics is derived from ECLConnection, you can obtain all the information contained in an ECLConnection object. See “ECLConnection Class” on page 19 for more information.

The ECLWinMetrics object is created for the connection identified upon construction. You may create an ECLWinMetrics object by passing either the connection ID (a single, alphabetical character from A-Z) or the connection handle, which is usually obtained from the ECLConnection object. There can be only one Personal Communications connection with a given name or handle open at a time.

Note: There is a pointer to the ECLWinMetrics object in the ECLSession class. If you just want to manipulate the connection window, create ECLWinMetrics on its own. If you want to do more, you may want to create an ECLSession object.

ECLWinMetrics Methods

The following methods apply to the ECLWinMetrics class.

ECLWinMetrics(char Name)
ECLWinMetrics(long Handle)
~ECLWinMetrics()
const char *GetWindowTitle()
void SetWindowTitle(char *NewTitle)
longGetXpos()
void SetXpos(long NewXpos)
longGetYpos()
void SetYpos(long NewYpos)
longGetWidth()
void SetWidth(long NewWidth)
long GetHeight()
void SetHeight(long NewHeight)
voidgetWindowRect(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 Constructor

This method creates an ECLWinMetrics object from a connection name or connection handle. There can be only one Personal Communications connection open with a given name. For example, there can be only one connection “A” open at a time.
Prototype
ECLWinMetrics(char Name)

ECLWinMetrics(long Handle)

Parameters
char Name  One-character short name of the connection (A-Z).
long Handle Handle of an ECL connection.

Return Value
None

Example
//-------------------------------------------------------------------
// 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 Destructor
This method destroys a ECLWinMetrics object.

Prototype
~ECLWinMetrics()

Parameters
None

Return Value
None

Example
//----------------------------------------------------------------------------
// ECLWinMetrics::ECLWinMetrics (Destructor)
//
// Build WinMetrics object from name.
//----------------------------------------------------------------------------
void Sample78()
{
    ECLWinMetrics *Metrics;  // Ptr to object
    try {

ECLWinMetrics

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

The GetWindowTitle method returns a pointer to a null terminate string containing
the title that is currently in the title bar for the connection associated with the
ECLWinMetrics object. Do not assume that the string returned is persistent over
time. You must either make a copy of the string or make a call to this method each
time you need it.

Prototype

const char *GetWindowTitle()

Parameters

None

Return Value

Pointer to null terminated string that contains the title.

Example

//-------------------------------------------------------------------
// 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

The SetWindowTitle method changes the title currently in the title bar for the
connection associated with the ECLWinMetrics object to the title passed in the
input parameter. A null string can be used to reset the title to the default title.
Prototype
void SetWindowTitle(char *NewTitle)

Parameters
char *NewTitle Null-terminated title string.

Return Value
None

Example
//-------------------------------------------------------------------
// 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

Usage Notes
If NewTitle is a null string, SetWindowTitle will restore the window title to its original setting.

GetXpos
The GetXpos method returns the x position of the upper left point of the connection window rectangle.

Prototype
long GetXpos()

Parameters
None

Return Value
long x position of connection window.
Example

//-------------------------------------------------------------------
// 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

The SetXpos method sets the x position of the upper left point of the connection window rectangle.

Prototype

void SetXpos(long NewXpos)

Parameters

long NewXpos The new x coordinate of the window rectangle.

Return Value

None

Example

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

The GetYpos method returns the y position of the upper left point of the connection window rectangle.

Prototype

long GetYpos()

Parameters

None

Return Value

long y position of the connection window.

Example

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
ECLWinMetrics

SetYpos
The SetYpos method sets the y position of the upper left point of the connection window rectangle.

Prototype
void SetYpos(long NewYpos)

Parameters
long NewYpos New y coordinate of the window rectangle.

Return Value
None

Example
//-------------------------------------------------------------------
// 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.
        ");
        } 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
This method returns the width of the connection window rectangle.

Prototype
long GetWidth()

Parameters
None

Return Value
long Width of the connection window.
Example
//--------------------------------------------------------------------------------------------------
// 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
The SetWidth method sets the width of the connection window rectangle.

Prototype
void SetWidth(long NewWidth)

Parameters
long NewWidth New width of the window rectangle.

Return Value
None

Example
//--------------------------------------------------------------------------------------------------
// 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...\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
ECLWinMetrics

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
The GetHeight method returns the height of the connection window rectangle.

Prototype
long GetHeight()

Parameters
None

Return Value
long
Height of the connection window.

Example
//-------------------------------------------------------------------
// 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
This method sets the height of the connection window rectangle.

Prototype
void SetHeight(Long NewHeight)

Parameters
long NewHeight          New height of the window rectangle.

Return Value
None

Example
The following example shows how to use the SetHeight method to set the height of the connection window rectangle.

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
This method returns the bounding points of the connection window rectangle.

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

Parameters
long *left          This output parameter is set to the left coordinate of the window rectangle.
long *top           This output parameter is set to the top coordinate of the window rectangle.
long *right         This output parameter is set to the right coordinate of the window rectangle.
long *bottom        This output parameter is set to the bottom coordinate of the window rectangle.

Return Value
None

Example
//-------------------------------------------------------------------
// 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
This method sets the bounding points of the connection window rectangle.

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

Parameters
long left                The left coordinate of the window rectangle.
long top                 The top coordinate of the window rectangle.
long right               The right coordinate of the window rectangle.
long bottom              The bottom coordinate of the window rectangle.

Return Value
None

Example
//-------------------------------------------------------------------
// 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
    Metrics = new ECLWinMetrics('A'); // Create for connection A
} catch (ECLErr Err) {
    printf("ECL Error: %s\n", Err.GetMsgText());
}
} // end sample
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, Width/2, Height/2); // Size window
}

delte Metrics;
}
catch (ECLErr Err) {
    printf("ECL Error: %s\n", Err.GetMsgText());
}
} // end sample

IsVisible
This method returns the visibility state of the connection window.

Prototype
BOOL IsVisible()

Parameters
None

Return Value
Visibility state. TRUE value if the window is visible, FALSE value if the window is not visible.

Example
//-------------------------------------------------------------------
// 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
This method sets the visibility state of the connection window.

Prototype
void SetVisible(BOOL SetFlag)

Parameters
BOOL SetFlag TRUE for visible, FALSE for invisible.

Return Value
None
ECLWinMetrics

**Example**

```c
//-------------------------------------------------------------------------------
// 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**

This method returns the focus state of the connection window.

**Prototype**

```c
BOOL Active()
```

**Parameters**

None

**Return Value**

`BOOL` Focus state. TRUE if active, FALSE if not active.

**Example**

```c
// 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**

This method sets the focus state of the connection window.

**Prototype**

```c
void SetActive(BOOL SetFlag)
```

**Parameters**

`Bool SetFlag` New state. TRUE for active, FALSE for inactive.

**Return Value**

None

**Example**

The following is an example of the SetActive method.
ECLWinMetrics

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

This method returns the minimize state of the connection window.

**Prototype**

BOOL IsMinimized()

**Parameters**

None

**Return Value**

**BOOL**

Minimize state. TRUE value returned if the window is minimized; FALSE value returned if the window is not minimized.

**Example**

//-------------------------------------------------------------------------------
// 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

This method sets the connection window to minimized

**Prototype**

void SetMinimized()

**Parameters**

None
Return Value
None

Example
//-------------------------------------------------------------------
// ECLWinMetrics::SetMinimized
// Get current state of window, and then toggle it.
polator: void Sample94()
{
ECLWinMetrics Metrics('A');   // Window metrics class
BOOL CurrState;

CurrState = Metrics.IsMinimized();  // Get state
if (CurrState)
    Metrics.SetMinimized();              // Set state
else
    Metrics.SetRestored();              // Set state
} // end sample

IsMaximized

This method returns the maximize state of the connection window.

Prototype
BOOL IsMaximized()

Parameters
None

Return Value
BOOL
Maximize state. TRUE value if the window is maximized; FALSE value if the window is not maximized.

Example
//-------------------------------------------------------------------
// ECLWinMetrics::IsMaximized
// Get current state of window, and then toggle it.
polator: void Sample97()
{
ECLWinMetrics Metrics('A');   // Window metrics class
BOOL CurrState;

CurrState = Metrics.IsMaximized();  // Get state
if (CurrState)
    Metrics.SetMaximized();              // Set state
else
    Metrics.SetRestored();              // Set state
} // end sample

SetMaximized

This method sets the connection window to maximized.

Prototype
void SetMaximized()
Parameters
None

Return Value
None

Example
//-------------------------------------------------------------------
// 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
This method returns the restore state of the connection window.

Prototype
BOOL IsRestored()

Parameters
None

Return Value
BOOL Restore state. TRUE value if the window is restored; FALSE value if the window is not restored.

Example
//-------------------------------------------------------------------
// 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
ECLWinMetrics

**SetRestored**
The SetRestored method sets the connection window to restored.

**Prototype**
void SetRestored()

**Parameters**
None

**Return Value**
None

**Example**
//-------------------------------------------------------------------
// 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 Class**

ECLXfer provides file transfer services.

**Derivation**
ECLBase > ECLConnection > ECLXfer

**Properties**
None

**Usage Notes**
Because ECLXfer is derived from ECLConnection, you can obtain all the information contained in an ECLConnection object. See “ECLConnection Class” on page 19 for more information.

The ECLXfer object is created for the connection identified upon construction. You may create an ECLXfer object by passing either the connection ID (a single, alphabetic character from A-Z) or the connection handle, which is usually obtained from the ECLConnList object. There can be only one Personal Communications connection with a given name or handle open at a time.

**Note:** There is a pointer to the ECLXfer object in the ECLSession class. If you only want to manipulate the connection window, create an ECLXfer object on its own. If you want to do more, you may want to create an ECLSession object.
ECLXfer Methods

The following section describes the methods that are valid for the ECLXfer class:

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

ECLXfer Constructor

This method creates an ECLXfer object from a connection ID (a single, alphabetic character from A-Z) or a connection handle. There can be only one Personal Communications connection open with a given ID. For example, there can be only one connection “A” open at a time.

Prototype
ECLXfer(char Name)
ECLXfer(long Handle)

Parameters
char Name One-character short name of the connection (A-Z).
long Handle Handle of an ECL connection.

Return Value
None

Example
//-------------------------------------------------------------------
// ECLXfer::ECLXfer (Constructor)
//-------------------------------------------------------------------
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 Destructor

This method destroys an ECLXfer object.

Prototype
~ECLXfer();
Parameters
None

Return Value
None

Example
//-------------------------------------------------------------------
// 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
This method sends a file from the workstation to the host.

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

Parameters
char *PCFile Pointer to a string containing the workstation file name to be sent to the host.
char *HostFile Pointer to a string containing the host file name to be created or updated on the host.
char *Options Pointer to a string containing the options to be used during the transfer.

Return Value
int EHLLAPI return code as documented in Emulator Programming for the SendFile EHLLAPI function.

Example
//-------------------------------------------------------------------
// ECLXfer::SendFile
//
// Send a file to a VM/CMS host with ASCII translation.
//-------------------------------------------------------------------
void Sample101() {
  ECLXfer *Xfer; // Pointer to Xfer object
  int Rc;
  try {

ECLXfer
Xfer = new ECLXfer('A'); // Create object for connection A

printf("Sending file...
");
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 sucessful.\n");
        break;
    case 4:
        printf("File transfer sucessful, 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

Usage Notes
File transfer options are host-dependent. The following is a list of some of the valid host options for a VM/CMS host:
   ASCII
   CRLF
   APPEND
   LRECL
   RECFM
   CLEAR/NOCLEAR
   PROGRESS
   QUIET

Refer to Emulator Programming for the list of supported hosts and associated file transfer options.

ReceiveFile
This method receives a file from the host and sends the file to the workstation.

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

Parameters
char *PCFile  Pointer to a string containing the workstation file name to be sent to the host.
char *HostFile  Pointer to a string containing the host file name to be created or updated on the host.
ECLXfer

char *Options      Pointer to a string containing the options to be used during the transfer.

Return Value

int      EHLLAPI return code as documented in Emulator Programming for the ReceiveFile EHLLAPI function.

Example

//..................................................................................
// 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...
");
    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

Usage Notes

File transfer options are host-dependent. The following is a list of some of the valid host options for a VM/CMS host:

ASCII
CRLF
APPEND
LRECL
RECFM
CLEAR/NOCLEAR
PROGRESS
QUIET
Refer to Emulator Programming for the list of supported hosts and associated file transfer options.

**ECLPageSettings Class**

The ECLPageSettings class performs operations on the session page settings. It enables you to retrieve and configure the File > Page Setup dialog settings, such as CPI, LPI, and Face Name. Only the settings in the Text tab of the dialog are supported.

This class is not supported for DBCS and bidirectional sessions.

**Derivation**

ECLBase > ECLConnection > ECLPageSettings

**Properties**

None

**Restrictions**

The connection associated with each method must be in a particular state for the method to succeed. If the restrictions are not met, an appropriate exception is raised.

The following restrictions apply when any method of the ECLPageSettings class is invoked. If the restrictions are not met, an exception is thrown.

- The connection Page Setup and Printer Setup dialogs must not be in use.
- The connection must not be printing.
- The associated connection must not be in PDT mode.

Additional restrictions might apply for each specific method.

**Usage Notes**

Because ECLPageSettings is derived from ECLConnection, you can obtain all the information contained in an ECLConnection object. See “ECLConnection Class” on page 19 for more information.

The ECLPageSettings object is created for the connection identified upon construction. You can create an ECLPageSettings object by passing the connection ID (a single alphabetical character from A to Z) or the connection handle (usually obtained from the ECLConnection object). There can be only one Personal Communications connection with a given name or handle open at one time.

The ECLSession class creates an instance of this object. If the application does not need other services provided by ECLSession, you can create this object independently. Otherwise, consider creating an ECLSession object and use the objects created by ECLSession. See “ECLSession Class” on page 145 for more information.

Each method supports only certain connection types of the connection associated with the ECLPageSettings object. The supported connection types are provided in each method section. If a method is called on an unsupported connection, an exception is thrown. Use the method GetConnType to determine the connection type.
CPI, LPI and FontSize are dependent on the property FaceName. Therefore, if CPI, LPI, and FontSize are set before the FaceName is set, and if the values are not valid for the FaceName property, then different CPI, LPI, or FontSize values might be reconfigured in the connection. You should set the FaceName value before setting the CPI, LPI, or FontSize. Or you can query CPI, LPI, and FontSize each time you set FaceName to ensure that they use the desired values.

**ECLPageSettings Methods**

The following sections describe the methods that are valid for the ECLPageSettings class.

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

**Connection types**

The valid connection types for the ECLPageSettings methods are as follows:

<table>
<thead>
<tr>
<th>Connection Type</th>
<th>String Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>3270 display</td>
<td>HOSTTYPE_3270DISPLAY</td>
</tr>
<tr>
<td>5250 display</td>
<td>HOSTTYPE_5250DISPLAY</td>
</tr>
<tr>
<td>3270 printer</td>
<td>HOSTTYPE_3270PRINTER</td>
</tr>
<tr>
<td>VT (ASCII) emulation</td>
<td>HOSTTYPE_VT</td>
</tr>
</tbody>
</table>

**ECLPageSettings Constructor**

This method uses a connection name or handle to create an ECLPageSettings object.

**Prototype**

ECLPageSettings(char Name)
ECLPageSettings(long Handle)

**Parameters**

<table>
<thead>
<tr>
<th>char Name</th>
<th>One-character short name of the connection. Valid values are A–Z.</th>
</tr>
</thead>
<tbody>
<tr>
<td>long Handle</td>
<td>Handle of an ECL connection.</td>
</tr>
</tbody>
</table>
Return Value
None

Example
The following example shows how to create an ECLPageSettings object using the connection name and the connection handle.

```c++
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
This method sets the CPI (characters per inch) value in the connection. If this method is called without any arguments, it sets the Font CPI in the connection.

Prototype
```c++
void SetCPI(ULONG CPI=FONT_CPI);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPI</td>
<td>Characters per inch. This parameter is optional. The default value is FONT_CPI.</td>
</tr>
</tbody>
</table>

Return Value
None

Example
```c++
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

This method returns the CPI (characters per inch) value of the connection. Even if Font CPI is selected in the associated connection, this method returns the value of the CPI selected for the font in the associated connection.

If Font CPI is configured in the connection, this method does not return the constant FONT_CPI. Use the IsFontCPI method to determine whether Font CPI is set in the connection.

Prototype
ULONG GetCPI() const;

Parameters
None

Return Value
ULONG CPI Characters per inch.

Example
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

This method returns an indication of whether Font CPI is set in the connection.

Prototype
BOOL IsFontCPI();

Parameters
None

Return Value
BOOL Possible values are as follows:
• TRUE if Font CPI is set in the connection.
• FALSE if Font CPI is not set in the connection.

Example
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
printf("FontCPI
");
else
  printf("Not FontCPI\n");
} // end sample

SetLPI

This method sets the LPI (lines per inch) value in the connection. If this method is called without any arguments, it sets the Font LPI in the connection.

Prototype
void SetLPI(ULONG LPI=FONT_LPI);

Parameters
ULONG LPI

Lines per inch. This parameter is optional. The default value is FONT_LPI.

Return Value
None

Example
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

This method returns the LPI (lines per inch) value of the connection. Even if Font LPI is selected in the associated connection, this method returns the value of the LPI selected for the font in the associated connection.

If Font LPI is configured in the connection, this method does not return the constant FONT_LPI. Use the IsFontLPI method to determine whether Font LPI is set in the connection.

Prototype
ULONG GetLPI() const;

Parameters
None

Return Value
ULONG LPI

Lines per inch.

Example
void Sample110() {
  ECLPageSettings PgSet('A');
  PgSet.SetLPI(10);
  ULONG lpi = PgSet.GetLPI();
ECLPageSettings

printf("LPI = %ld
", lpi);
if (PgSet.IsFontLPI())
    printf("FontLPI
");
else
    printf("Not FontLPI
");
} // end sample

IsFontLPI

This method returns an indication of whether **Font LPI** is set in the associated connection.

**Prototype**

```c
BOOL IsFontLPI();
```

**Parameters**

None

**Return Value**

`BOOL`

Possible values are as follows:

- **TRUE** if **Font LPI** is set in the connection.
- **FALSE** if **Font LPI** is not set in the connection.

**Example**

```c
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

This method sets the font face in the connection.

**Prototype**

```c
void SetFontFaceName(const char *const FaceName);
```

**Parameters**

* `char *FaceName` - A null-terminated string that contains the font face name.

**Return Value**

None

**Example**

```c
void Sample111() {
    ECLPageSettings PgSet('A');

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

This method returns a pointer to a null-terminated string. The string contains the face name of the font that is currently chosen in the page settings for the connection that is associated with the ECLPageSettings object. The method might not return the same string each time.

The string is valid only for the lifetime of the object. You must either make a copy of the string or make a call to this method each time you need it.

Prototype

const char *GetFontFaceName() const;

Parameters

None

Return Value

char * A pointer to a null-terminated string that contains the face name of the font.

Example

void Sample111() {
    ECLPageSettings PgSet('A');
    const char *Face;

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

SetFontSize

This method sets the size of the font.

Prototype

void SetFontSize(ULONG FontSize);

Parameters

ULONG FontSize Size of the font to set in the connection.

Return Value

None

Example

void Sample112() {
    ECLPageSettings PgSet('A');

    if (PgSet.IsDBCSHost()) {
        PgSet.SetFontSize(10);
        ULONG size = PgSet.GetFontSize();
        printf("FontSize = %ld\n", size);
    }
} // end sample

GetFontSize

This method returns the size of the font that is set in the connection.
ECLPageSettings

Note: This method is valid only if the connection uses a DBCS host code page. That is, the method IsDBCSHost must return TRUE for the connection. For SBCS connections, this method fails with an exception.

Prototype
ULONG GetFontSize();

Parameters
None

Return Value
ULONG The size of the font that is set in the connection.

Example
void Sample112() {
   ECLPageSettings PgSet('A');
   if (PgSet.IsDBCSHost()) {
      PgSet.SetFontSize(10);
      ULONG size = PgSet.GetFontSize();
      printf("FontSize = %ld\n", size);
   }
} // end sample

SetMaxLinesPerPage
This method sets the maximum number of lines that can be printed per page.

Prototype
void SetMaxLinesPerPage(ULONG MPL);

Parameters
ULONG MPL The maximum lines per page (Maximum Print Lines). Valid values are in the range 1–255.

Return Value
None

Example
void Sample113() {
   ECLPageSettings PgSet('A');
   PgSet.SetMaxLinesPerPage(40);
   ULONG MPL = PgSet.GetMaxLinesPerPage();
   printf("MaxLinesPerPage = %ld\n", MPL);
} // end sample

GetMaxLinesPerPage
This method returns the maximum number of lines that can be printed per page.

Prototype
ULONG GetMaxLinesPerPage() const;

Parameters
None
Return Value

ULONG The maximum lines per page (Maximum Print Lines).

Example

```cpp
void Sample113() {
    ECLPageSettings PgSet('A');
    PgSet.SetMaxLinesPerPage(40);
    ULONG MPL = PgSet.GetMaxLinesPerPage();
    printf("MaxLinesPerPage = %ld\n", MPL);
} // end sample
```

SetMaxCharsPerLine

This method sets the maximum number of characters that can be printed per line.

Prototype

```cpp
void SetMaxCharsPerLine(ULONG MPP);
```

Parameters

ULONG MPP The maximum number of characters that can be printed per line (Maximum Print Position). Valid values are in the range 1–255.

Return Value

None

Example

```cpp
void Sample114() {
    ECLPageSettings PgSet('A');
    PgSet.SetMaxCharsPerLine(50);
    ULONG MPP = PgSet.GetMaxCharsPerLine();
    printf("MaxCharsPerLine=%ld\n", MPP);
} // end sample
```

GetMaxCharsPerLine

This method returns the maximum number of characters that can be printed per line.

Prototype

```cpp
ULONG GetMaxCharsPerLine() const;
```

Parameters

None

Return Value

ULONG The maximum number of characters that can be printed per line (Maximum Print Position).

Example

```cpp
void Sample114() {
    ECLPageSettings PgSet('A');
```
ECLPageSettings

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

**RestoreDefaults**

This method restores the system default values of the property pages specified in the nFlags field of the PageSetup panel. This is equivalent to clicking the Default button in the connection Page Setup dialog property pages.

**Prototype**

void RestoreDefaults(ULONG Flags=PAGE_TEXT) const;

**Parameters**

ULONG Flags

This parameter is optional. The following flag describes the name of the specified Page Setup dialog property page. This flag can be bitwise ORed to restore the property page (defined in PCSAPI32.H).

- **PAGE_TEXT**
  This flag describes the Text property page. This is the only property page currently supported.

**Return Value**

None

**Example**

void Sample115() {
   ECLPageSettings PgSet('A');
   PgSet.RestoreDefaults(PAGE_TEXT);
} // end sample

---

**ECLPrinterSettings Class**

The ECLPrinterSettings class performs operations on the printer settings of the Personal Communications connection. It enables you to retrieve and configure the File + Printer Setup dialog settings, such as Printer and PDT Mode.

This class is not supported for DBCS and bidirectional sessions.

**Derivation**

ECLBase > ECLConnection > ECLPrinterSettings

**Properties**

None

**Restrictions**

The connection associated with each method must be in a particular state for the method to succeed. If the restrictions are not met, an appropriate exception is raised.
The following restrictions apply when any method of the ECLPrinterSettings class is invoked. If the restrictions are not met, an exception is thrown.

- The connection Page Setup and Printer Setup dialogs must not be in use.
- The connection must not be printing.

Additional restrictions might apply for each specific method.

**Usage Notes**

Because ECLPrinterSettings is derived from ECLConnection, you can obtain all the information contained in an ECLConnection object. See “ECLConnection Class” on page 19 for more information.

The ECLPrinterSettings object is created for the connection identified upon construction. You can create an ECLPrinterSettings object by passing either the connection ID (a single alphabetical character from A to Z) or the connection handle (usually obtained from the ECLConnection object). There can be only one Personal Communications connection with a given name or handle open at one time.

The ECLSession class creates an instance of this object. If the application does not need other services provided by ECLSession, you can create this object independently. Otherwise, consider creating an ECLSession object and use the objects created by ECLSession. See “ECLSession Class” on page 145 for more information.

**ECLPrinterSettings Methods**

The following sections describe the methods that are valid for the ECLPrinterSettings class.

- 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() const
- void SetPrtToDskSeparate(const char*const FileName = NULL)
- const char *GetPrtToDskSeparateFile() const
- void SetSpecificPrinter(const char*const PrinterName)
- void SetWinDefaultPrinter()
- const char*GetPrinterName()
- void SetPromptDialog(BOOL Prompt=TRUE)
- BOOL IsPromptDialogEnabled()
ECLPrinterSettings

Parameters

char Name  One-character short name of the connection. Valid values are A–Z.

long Handle  Handle of an ECL connection.

Return Value

None

Example

The following example shows how to create an ECLPrinterSettings object using the connection name and the connection handle.

```c
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.",
                   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

This method sets the connection in PDT mode with the given PDT file, or it sets the connection in non-PDT mode (GDI mode).

Note: If this method is called with PDTMode set to FALSE, PrintMode of the associated connection must already be SpecificPrinter or WinDefaultPrinter.

Prototype

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

Parameters

BOOL PDTMode  This parameter is optional. Possible values are as follows:

- **TRUE** to set the connection to PDT mode. This is the default value.
- **FALSE** to set the connection in non-PDT mode.

char *PDTFile  Null-terminated string containing the name of the PDT file.

This parameter is optional. It is used only if PDTMode is TRUE. The parameter is ignored if PDTMode is FALSE.

Possible values are as follows:
• **NULL**
  The PDT file configured in the connection is used. If there is no PDT file already configured in the connection, this method fails with an exception. This is the default value.

• File name without the path
  PDTFile in the PDPDT subfolder in the Personal Communications installation path is used.

• Fully qualified path name of the file
  If PDTFile does not exist, this method fails with an exception.

### Return Value
None

### Example
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
This method returns the PDT file configured in the connection. The method might not return the same string each time.

The string is valid only for the lifetime of the object. You must either make a copy of the string or make a call to this method each time you need it.

### Prototype
const char *GetPDTFile() const;

### Parameters
None

### Return Value
char *

Possible values are as follows:

• A null-terminated string containing the fully qualified path name of the PDT file of the connection.

• **NULL** if no PDT file is configured in the connection.
Example

```c
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", Err.GetMsgText());
    }
} // end sample
```

IsPDTMode

This method returns the state of the PDT mode of the connection.

Prototype

```c
BOOL IsPDTMode() const;
```

Parameters

None

Return Value

```c
BOOL
```

Possible values are as follows:

- TRUE if the connection is in PDT mode.
- FALSE if the connection is not in PDT mode.

Example

```c
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", Err.GetMsgText());
    }
} // end sample
```

GetPrintMode

This method returns an enumerated value that indicates the PrintMode of the connection. The enum data type ECLPrinterSettings::PrintMode is defined in ECLPRSET.HPP.
PrintMode can be one of the following:

- **PrtToDskAppend (Print to Disk-Append mode)**
  
  This is equivalent to selecting the **Append** option in the host session **Printer Setup → Printer → Print to Disk** dialog.

- **PrtToDskSeparate (Print to Disk-Separate mode)**
  
  This is equivalent to selecting the **Separate** option in the host session **Printer Setup → Printer → Print to Disk** dialog.

- **WinDefaultPrinter (Windows Default Printer mode)**
  
  This is equivalent to selecting the **Use Windows Default Printer** option in the host session **Printer Setup** dialog.

- **SpecificPrinter (Specific Printer mode)**
  
  This is equivalent to selecting a printer in the host session **Printer Setup** dialog, while leaving **Use Windows Default Printer** unchecked.

**Prototype**

```
ECLPrinterSettings::PrintMode GetPrintMode() const;
```

**Parameters**

None

**Return Value**

```
ECLPrinterSettings::PrintMode
```

One of the PrintMode values defined in **ECLPRSET.HPP**.

**Example**

```cpp
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**

This method sets the PrintMode to **Print to Disk-Append** mode and sets the appropriate file for this mode.

**Notes:**

1. The associated connection must be in PDT mode.
2. The folder where this file is to be set must have write access. If it does not, this method fails with an exception.
3. If the file exists, it will be used. Otherwise, it will be created when printing is complete.

Prototype

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

Parameters

- `char *FileName`
  - Null-terminated string containing the name of the Print to Disk-Append file. This parameter is optional.
  - Possible values are as follows:
    - `NULL`
      - The file that is currently configured for this PrintMode in the connection is used. If there is no file already configured in the connection, the method fails with an exception. This is the default value.
    - `File name, without the path`
      - The user-class application data directory path will be used to locate the file.
    - `Fully qualified path name of the file`
      - The directory must exist in the path, or the method will fail with an exception. It is not necessary that the file exist in the path.

Return Value

None

Example

```c
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

This method returns the file configured for Print to Disk-Append mode. This file is called the Print to Disk-Append file. The method might not return the same string each time.

The string is valid only for the lifetime of the object. You must either make a copy of the string or make a call to this method each time you need it.

Prototype

```c
const char *GetPrtToDskAppendFile();
```

Parameters

None
Return Value

char *

Possible values are as follows:
- A null-terminated string that contains the fully qualified path name of the Print to Disk-Append file of the connection.
- NULL if the Print to Disk-Append file is not configured in the connection.

Example

```cpp
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

This method sets the connection in Print to Disk-Separate mode and sets the appropriate file for this mode.

Notes:
1. The associated connection must be in PDT mode.
2. The folder where this file is to be set must have write access. If it does not, this method fails with an exception.
3. The file name must not contain an extension. If it contains an extension, the method fails with an exception.

Prototype

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

Parameters

char *FileName

Null-terminated string containing the name of the Print to Disk-Separate file. This parameter is optional.

Possible values are as follows:
- NULL
  - The file that is currently configured for this PrintMode in the connection is used. If there is no file already configured in the connection, the method fails with an exception. This is the default value.
- File name, without the path
  - The user-class application data directory path will be used to locate the file.
- Fully qualified path name of the file
ECLPrinterSettings

The directory must exist in the path, or the method will fail with an exception. It is not necessary that the file exist in the path.

Return Value
None

Example

```c
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

This method returns the file configured for Print to Disk-Separate mode. This file is called the Print to Disk-Separate file. The method might not return the same string each time.

The string is valid only for the lifetime of the object. You must either make a copy of the string or make a call to this method each time you need it.

Prototype

```c
const char *GetPrtToDskSeparateFile();
```

Parameters
None

Return Value

char *

Possible values are as follows:

- A null-terminated string that contains the fully qualified path name of the Print to Disk-Separate file.
- NULL, if no Print to Disk-Separate file is configured in the connection.

Example

```c
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

This method sets the connection in SpecificPrinter mode with the printer specified in the Printer parameter.

Prototype

```cpp
void SetSpecificPrinter(const char *const Printer);
```

Parameters

```cpp
char *Printer
```

A null-terminated string that contains the printer name and the port name. If the printer does not exist, this method fails with an exception.

The value must have the following format:

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

For example:

- IBM InfoPrint 40 PS on Network Port
- HP LaserJet 4050 Series PCL 6 on LPT1

Return Value

None

Example

```cpp
void Sample121() {
    ECLPrinterSettings PrSet('A');
    try {
        PrSet.SetSpecificPrinter("IBM 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

This method sets the connection in WinDefaultPrinter mode—that is, the connection is made to use the Windows default printer. If no Windows default printer is configured in the machine, the method fails with an exception.

Prototype

```cpp
void SetWinDefaultPrinter();
```

Parameters

None

Return Value

None

Example

```cpp
void Sample122() {
    ECLPrinterSettings PrSet('A');
    try {
        PrSet.SetWinDefaultPrinter();
    }
```
ECLPrinterSettings

cnst char *Printer = PrSet.GetPrinterName();
printf("Windows Default Printer = %s\n", Printer);
} catch (ECLErr Err) {
    printf("ECL Error: %s\n", Err.GetMsgText());
}
} // end sample

GetPrinterName

This method returns NULL or the name of the printer configured in the connection. The method might not return the same string each time.

The string is valid only for the lifetime of the object. You must either make a copy of the string or make a call to this method each time you need it.

PrinterName must have the following format:
<Printer name> on <Port Name>

For example:
• IBM InfoPrint 40 PS on Network Port
• HP LaserJet 4050 Series PCL 6 on LPT1

Prototype
cnst char *GetPrinterName();

Parameters
None

Return Value
char *

Possible values are as follows:
• A null-terminated string that contains the name of the specific printer, if the PrintMode of the connection is SpecificPrinter.
• A null-terminated string that contains the name of the Windows default printer, if the PrintMode of the connection is WinDefaultPrinter.
• NULL if no Printer is configured in the connection, or if the PrintMode of the connection is PrtToDskAppend or PrtToDskSeparate.

Example
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

196  Host Access Class Library
SetPromptDialog

This method sets or resets the option to show the Printer Setup dialog before printing.

Prototype
void SetPromptDialog(BOOL bPrompt=TRUE);

Parameters

BOOL bPrompt
This parameter is optional. Possible values are as follows:
- TRUE to show the Printer Setup dialog before printing. This is the default value.
- FALSE to not show the Printer Setup dialog before printing.

Return Value
None

Example

```c++
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

This method checks whether the Printer Setup dialog is shown before printing or not.

Prototype
BOOL IsPromptDialogEnabled();

Parameters
None

Return Value

BOOL Possible values are as follows:
- TRUE if the Printer Setup dialog is shown before printing.
- FALSE if the Printer Setup dialog is not shown before printing.

Example

```c++
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
Chapter 3. Host Access Class Library Automation Objects

The Host Access Class Library Automation Objects allow the Personal Communications product to support Microsoft COM-based automation technology (formerly known as OLE automation). The ECL Automation Objects are a series of automation servers that allow automation controllers, for example, Microsoft Visual Basic, to programmatically access Personal Communications data and functionality.

An example of this would be sending keys to Personal Communications presentation space. This can be accomplished by manually typing keys in the Personal Communications window, but it can also be automated through the appropriate Personal Communications automation server (autECLPS in this case). Using Visual Basic you can create the autECLPS object and then call the SendKeys method in that object with the string that is to be placed in the presentation space.

In other words, applications that are enabled for controlling the automation protocol (automation controller) can control some Personal Communications operations (automation server). Personal Communications supports Visual Basic Script, which uses ECL Automation objects. Refer to the Personal Communications Macro/Script support for more details.

Personal Communications offers several automation servers to accomplish this. These servers are implemented as real-world, intuitive objects with methods and properties that control Personal Communications operability. Each object begins with autECL, for automation Host Access Class Library. The objects are as follows:

- autECLConnList, Connection List, on page 201 contains a list of Personal Communications connections for a given system. This is contained by autECLConnMgr, but may be created independently of autECLConnMgr.
- autECLConnMgr, Connection Manager, on page 207 provides methods and properties to manage Personal Communications connections for a given system. A connection in this context is a Personal Communications window.
- autECLFieldList, Field List, on page 211 performs operations on fields in an emulator presentation space.
- autECLOIA, Operator Information Area, on page 219 provides methods and properties to query and manipulate the Operator Information Area. This is contained by autECLSession, but may be created independently of autECLSession.
- autECLPS, Presentation Space, on page 233 provides methods and properties to query and manipulate the presentation space for the related Personal Communications connection. This contains a list of all the fields in the presentation space. It is contained by autECLSession, but may be created independently of autECLSession.
- autECLScreenDesc, Screen Description, on page 261 provides methods and properties to describe a screen. This may be used to wait for screens on the autECLPS object or the autECLScreenReco object.
- autECLScreenReco, Screen Recognition, on page 267 provides the engine of the HACL screen recognition system.
- autECLSession, Session, on page 271 provides general session-related functionality and information. For convenience, it contains the autECLPS, autECLOIA, autECLXfer, autECLWinMetrics, autECLPageSettings, and autECLPrinterSettings objects.
- autECLWinMetrics, Window Metrics, on page 281, provides methods to query the window metrics of the Personal Communications session associated with this object. For example, use this object to minimize or maximize a Personal Communications window. This is contained by autECLSession, but may be created independently of autECLSession.

- autECLXfer, File Transfer, on page 294, provides methods and properties to transfer files between the host and the workstation over the Personal Communications connection associated with this file transfer object. This is contained by autECLSession, but may be created independently of autECLSession.

- autECLPageSettings, Page Settings, on page 304, provides methods and properties to query and manipulate commonly used settings such as CPI, LPI, and Face Name of the session Page Setup dialog. This is contained by autECLSession, but may be created independently of autECLSession.

- autECLPrinterSettings, Printer Settings, on page 313, provides methods and properties to query and manipulate settings such as the Printer and PDT modes of the session Printer Setup dialog. This is contained by autECLSession, but may be created independently of autECLSession.

Figure 3 is a graphical representation of the autECL objects:

![Graphical representation of autECL objects](image)

**Figure 3. Host Access Class Library Automation Objects**

This chapter describes each object’s methods and properties in detail and is intended to cover all potential users of the automation object. Because the most common way to use the object is through a scripting application such as Visual Basic, all examples are shown using a Visual Basic format.

**autSystem Class**

The autSystem Class provides two utility functions that may be useful for use with some programming languages. See "autSystem Class" on page 303 for more information.
autECLConnList Class

autECLConnList contains information about all started connections. Its name in the registry is PCOMM.autECLConnList.

The autECLConnList object contains a collection of information about connections to a host. Each element of the collection represents a single connection (emulator window). A connection in this list may be in any state (for example, stopped or disconnected). All started connections appear in this list. The list element contains the state of the connection.

An autECLConnList object provides a static snapshot of current connections. The list is not dynamically updated as connections are started and stopped. The Refresh method is automatically called upon construction of the autECLConnList object. If you use the autECLConnList object right after its construction, your list of connections is current. However, you should call the Refresh method in the autECLConnList object before accessing its other methods if some time has passed since its construction to ensure that you have current data. Once you have called Refresh you may begin walking through the collection.

Properties

This section describes the properties for the autECLConnList object.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long</td>
<td>Count</td>
<td>Read-only</td>
</tr>
</tbody>
</table>

Count

This is the number of connections present in the autECLConnList collection for the last call to the Refresh method. The Count property is a Long data type and is read-only. The following example uses the Count property.

```
Dim autECLConnList as Object
Dim Num as Long

Set autECLConnList = CreateObject("PCOMM.autECLConnList")
autECLConnList.Refresh
Num = autECLConnList.Count
```

The following table shows Collection Element Properties, which are valid for each item in the list.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>Name</td>
<td>Read-only</td>
</tr>
<tr>
<td>Long</td>
<td>Handle</td>
<td>Read-only</td>
</tr>
<tr>
<td>String</td>
<td>ConnType</td>
<td>Read-only</td>
</tr>
<tr>
<td>Long</td>
<td>CodePage</td>
<td>Read-only</td>
</tr>
<tr>
<td>Boolean</td>
<td>Started</td>
<td>Read-only</td>
</tr>
<tr>
<td>Boolean</td>
<td>CommStarted</td>
<td>Read-only</td>
</tr>
<tr>
<td>Boolean</td>
<td>APIEnabled</td>
<td>Read-only</td>
</tr>
<tr>
<td>Boolean</td>
<td>Ready</td>
<td>Read-only</td>
</tr>
</tbody>
</table>
**Name**

This collection element property is the connection name string of the connection. Personal Communications only returns the short character ID (A-Z) in the string. There can be only one Personal Communications connection open with a given name. For example, there can be only one connection “A” open at a time. Name is a String data type and is read-only. The following example uses the Name collection element property.

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

Set autECLConnList = CreateObject("PCOMM.autECLConnList")
autECLConnList.Refresh
Str = autECLConnList(1).Name
```

**Handle**

This collection element property is the handle of the connection. There can be only one Personal Communications connection open with a given handle. Handle is a Long data type and is read-only. The following example uses the Handle property.

```vba
Dim autECLConnList as Object
Dim Hand as Long

Set autECLConnList = CreateObject("PCOMM.autECLConnList")
autECLConnList.Refresh
Hand = autECLConnList(1).Handle
```

**ConnType**

This collection element property is the connection type. This type may change over time. ConnType is a String data type and is read-only. The following example shows the ConnType property.

```vba
Dim Type as String
Dim autECLConnList as Object

Set autECLConnList = CreateObject("PCOMM.autECLConnList")
autECLConnList.Refresh
Type = autECLConnList(1).ConnType
```

Connection types for the ConnType property are:

<table>
<thead>
<tr>
<th>String Returned</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISP3270</td>
<td>3270 display</td>
</tr>
<tr>
<td>DISP5250</td>
<td>5250 display</td>
</tr>
<tr>
<td>PRNT3270</td>
<td>3270 printer</td>
</tr>
<tr>
<td>PRNT5250</td>
<td>5250 printer</td>
</tr>
<tr>
<td>ASCII</td>
<td>VT emulation</td>
</tr>
</tbody>
</table>

**CodePage**

This collection element property is the code page of the connection. This code page may change over time. CodePage is a Long data type and is read-only. The following example shows the CodePage property.

```vba
Dim CodePage as Long
Dim autECLConnList as Object
```
```vba
Set autECLConnList = CreateObject("PCOMM.autECLConnList")
autECLConnList.Refresh
CodePage = autECLConnList(1).CodePage

**Started**
This collection element property indicates whether the emulator window is started. The value is True if the window is open; otherwise, it is False. Started is a Boolean data type and is read-only. The following example shows the Started property.

```vba
Dim autECLConnList as Object
Set autECLConnList = CreateObject("PCOMM.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**
This collection element property indicates the status of the connection to the host. The value is True if the host is connected; otherwise, it is False. CommStarted is a Boolean data type and is read-only. The following example shows the CommStarted property.

```vba
Dim autECLConnList as Object
Set autECLConnList = CreateObject("PCOMM.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**
This collection element property indicates whether the emulator is API-enabled. A connection may be enabled or disabled depending on the state of its API settings (in a Personal Communications window, choose *File* -> *API Settings*). The value is True if the emulator is enabled; otherwise, it is False. APIEnabled is a Boolean data type and is read-only. The following example shows the APIEnabled property.

```vba
Dim autECLConnList as Object
Set autECLConnList = CreateObject("PCOMM.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**
This collection element property indicates whether the emulator window is started, API-enabled, and connected. This property checks for all three properties. The
value is True if the emulator is ready; otherwise, it is False. Ready is a Boolean data type and is read-only. The following example shows the Ready property.

```vba
Dim autECLConnList as Object
Set autECLConnList = CreateObject("PCOMM.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 Methods

The following section describes the methods that are valid for the autECLConnList object.

- `void Refresh()`
- `Object FindConnectionByHandle(Long Hand)`
- `Object FindConnectionByName(String Name)`

## Collection Element Methods

The following collection element methods are valid for each item in the list.

- `void StartCommunication()`
- `void StopCommunication()`

### Refresh

The Refresh method gets a snapshot of all the started connections.

**Note:** You should call this method before accessing the autECLConnList collection to ensure that you have current data.

#### Prototype

```vba
void Refresh()
```

#### Parameters

None

#### Return Value

None

#### Example

The following example shows how to use the Refresh method to get a snapshot of all the started connections.

```vba
Dim autECLPSObj as Object
Dim autECLConnList as Object
Set autECLPSObj = CreateObject("PCOMM.autECLPS")
Set autECLConnList = CreateObject("PCOMM.autECLConnList")

' Initialize the connection
autECLConnList.Refresh
autECLPSObj.SetConnectionByHandle(autECLConnList(1).Handle)
```
FindConnectionByHandle

This method finds an element in the autECLConnList object for the handle passed in the **Hand** parameter. This method is commonly used to see if a given connection is alive in the system.

**Prototype**

Object FindConnectionByHandle(Long Hand)

**Parameters**

Long Hand  Handle to search for in the list.

**Return Value**

Object  Collection element dispatch object.

**Example**

The following example shows how to find an element by the connection handle.

```vbnet
Dim Hand as Long
Dim autECLConnList as Object
Dim ConnObj as Object

Set autECLConnList = CreateObject("PCOMM.autECLConnList")

' Initialize the collection
autECLConnList.Refresh
' Assume Hand obtained earlier
Set ConnObj = autECLConnList.FindConnectionByHandle(Hand)
Hand = ConnObj.Handle
```

FindConnectionByName

This method finds an element in the autECLConnList object for the name passed in the **Name** parameter. This method is commonly used to see if a given connection is alive in the system.

**Prototype**

Object FindConnectionByName(String Name)

**Parameters**

String Name  Name to search for in the list.

**Return Value**

Object  Collection element dispatch object.

**Example**

The following example shows how to find an element in the autECLConnList object by the connection name.

```vbnet
Dim Hand as Long
Dim autECLConnList as Object
Dim ConnObj as Object

Set autECLConnList = CreateObject("PCOMM.autECLConnList")

' Initialize the collection
autECLConnList.Refresh
' Assume Hand obtained earlier
Set ConnObj = autECLConnList.FindConnectionByName("A")
Hand = ConnObj.Handle
```
StartCommunication

The StartCommunication collection element method connects the PCOMM emulator to the host data stream. This has the same effect as going to the PCOMM emulator Communication menu and choosing Connect.

Prototype

void StartCommunication()

Parameters

None

Return Value

None

Example

The following example shows how to connect a PCOMM emulator session to the host.

Dim autECLConnList as Object

Set autECLConnList = CreateObject("PCOMM.autECLConnList")

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

StopCommunication

The StopCommunication collection element method disconnects the PCOMM emulator to the host data stream. This has the same effect as going to the PCOMM emulator Communication menu and choosing Disconnect.

Prototype

void StopCommunication()

Parameters

None

Return Value

None

Example

The following example shows how to disconnect a PCOMM emulator session from the host.

Dim autECLConnList as Object

Set autECLConnList = CreateObject("PCOMM.autECLConnList")

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

'Interact programmatically with host
autECLConnList.Refresh

'Stop the first session
autECLConnList(1).StartCommunication()
autECLConnMgr Class

autECLConnMgr manages all Personal Communications connections on a given machine. It contains methods relating to the connection management such as starting and stopping connections. It also creates an autECLConnList object to enumerate the list of all known connections on the system (see “autECLConnList Class” on page 201). Its name in the registry is PCOMM.autECLConnMgr.

Properties

This section describes the properties for the autECLConnMgr object.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>autECLConnList Object</td>
<td>autECLConnList</td>
<td>Read-only</td>
</tr>
</tbody>
</table>

autECLConnList

The autECLConnMgr object contains an autECLConnList object. See “autECLConnList Class” on page 201 for details on its methods and properties. The property has a value of autECLConnList, which is an autECLConnList dispatch object. The following example shows this property.

```vba
Dim Mgr as Object
Dim Num as Long

Set Mgr = CreateObject("PCOMM.autECLConnMgr")
Mgr.autECLConnList.Refresh
Num = Mgr.autECLConnList.Count
```

autECLConnMgr Methods

The following section describes the methods that are valid for autECLConnMgr.

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

RegisterStartEvent

This method registers an autECLConnMgr object to receive notification of start events in sessions.

Prototype

```vba
void RegisterStartEvent()
```

Parameters

None

Return Value

None

Example

See “Event Processing Example” on page 210 for an example.

UnregisterStartEvent

Ends Start Event Processing
Prototype
void UnregisterStartEvent()

Parameters
None

Return Value
None

Example
See “Event Processing Example” on page 21 for an example.

StartConnection
This member function starts a new Personal Communications emulator window. The ConfigParms string contains connection configuration information as explained under “Usage Notes.”

Prototype
void StartConnection(String ConfigParms)

Parameters
String ConfigParms Configuration string.

Return Value
None

Usage Notes
The configuration string is implementation-specific. Different implementations of the autECL objects may require different formats or information in the configuration string. The new emulator is started upon return from this call, but it may or may not be connected to the host.

For Personal Communications, the configuration string has the following format:
PROFILE="["<filename>"[" [CONNNAME=<c>] [WINSTATE=<MAX|MIN|RESTORE|HIDE>]"

Optional parameters are enclosed in square brackets []. The parameters are separated by at least one blank. Parameters may be in upper, lower, or mixed case and may appear in any order. The meaning of each parameter is as follows:

- PROFILE=<filename>: Names the Personal Communications workstation profile (.WS file), which contains the configuration information. This parameter is not optional; a profile name must be supplied. If the file name contains blanks the name must be enclosed in single quotation marks. The <filename> value may be either the profile name with no extension, the profile name with the .WS extension, or the fully qualified profile name path.
- CONNNAME=<c> specifies the short ID of the new connection. This value must be a single, alphabetic character (A-Z). If this value is not specified, the next available connection ID is assigned automatically.
- WINSTATE=<MAX|MIN|RESTORE|HIDE> specifies the initial state of the emulator window. The default if this parameter is not specified is RESTORE.

Example
The following example shows how to start a new Personal Communications emulator window.
Dim Mgr as Object
Dim Obj as Object
Dim Hand as Long

Set Mgr = CreateObject("PCOMM.autECLConnMgr")
Mgr.StartConnection("profile=coax connname=e")

StopConnection

The StopConnection method stops (terminates) the emulator window identified by the connection handle. See Usage Notes for contents of the StopParms string.

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

Parameters

Variant Connection Connection name or handle. Legal types for this variant are short, long, BSTR, short by reference, long by reference, and BSTR by reference.

String StopParms Stop parameters string. See usage notes for format of string. This parameter is optional.

Return Value
None

Usage Notes
The stop parameter string is implementation-specific. Different implementations of the autECL objects may require a different format and contents of the parameter string. For Personal Communications, the string has the following format:

[SAVEPROFILE=<YES|NO|DEFAULT>]

Optional parameters are enclosed in square brackets []. The parameters are separated by at least one blank. Parameters may be in upper, lower, or mixed case and may appear in any order. The meaning of each parameter is as follows:

- SAVEPROFILE=<YES|NO|DEFAULT> controls the saving of the current configuration back to the workstation profile (.WS file). This causes the profile to be updated with any configuration changes you may have made. If NO is specified, the connection is stopped and the profile is not updated. If YES is specified, the connection is stopped and the profile is updated with the current (possibly changed) configuration. If DEFAULT is specified, the update option is controlled by the File->Save On Exit emulator menu option. If this parameter is not specified, DEFAULT is used.

Example

The following example shows how to stop the emulator window identified by the connection handle.

Dim Mgr as Object
Dim Hand as Long

Set Mgr = CreateObject("PCOMM.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"
The following events are valid for autECLConnMgr:

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

NotifyStartEvent
A Session has started or stopped.

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

Note: Visual Basic will create this subroutine correctly.

Parameters
By Val Handle As Variant          Handle of the Session that started or stopped.
By Val Started As Boolean        True if the Session is started, False otherwise.

Example
See “Event Processing Example” for an example.

NotifyStartError
This event occurs when an error occurs in Event Processing.

Prototype
NotifyStartError(By Val ConnHandle As Variant)

Note: Visual Basic will create this subroutine correctly.

Parameters
None

Example
See “Event Processing Example” for an example.

NotifyStartStop
This event occurs when event processing stops.

Prototype
void NotifyStartStop(Long Reason)

Parameters
Long Reason                  Reason code for the stop. Currently, this will always be 0.

Event Processing Example
The following is a short example of how to implement Start Events:
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("PCOMM.autECLSession")
mCmgr.RegisterStartEvent "register for PS Updates"
'Message display in your form
Display your form or whatever here (this should be a blocking call, otherwise sub just ends
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 Class

autECLFieldList performs operations on fields in an emulator presentation space. This object does not stand on its own. It is contained by autECLPS, and can only be accessed through an autECLPS object. autECLPS can stand alone or be contained by autECLSession.

autECLFieldList contains a collection of all the fields on a given presentation space. Each element of the collection contains the elements shown in Collection Element Properties.

An autECLFieldList object provides a static snapshot of what the presentation space contained when the Refresh method was called.

Note: You should call the Refresh method in the autECLFieldList object before accessing its elements to ensure that you have current field data. Once you have called Refresh, you may begin walking through the collection.

Properties

This section describes the properties and the collection element properties for the autECLFieldList object.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long</td>
<td>Count</td>
<td>Read-only</td>
</tr>
</tbody>
</table>

Count

This property is the number of fields present in the autECLFieldList collection for the last call to the Refresh method. Count is a Long data type and is read-only. The following example shows this property.
The following properties are collection element properties and are valid for each item in the list.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long</td>
<td>StartRow</td>
<td>Read-only</td>
</tr>
<tr>
<td>Long</td>
<td>StartCol</td>
<td>Read-only</td>
</tr>
<tr>
<td>Long</td>
<td>EndRow</td>
<td>Read-only</td>
</tr>
<tr>
<td>Long</td>
<td>EndCol</td>
<td>Read-only</td>
</tr>
<tr>
<td>Long</td>
<td>Length</td>
<td>Read-only</td>
</tr>
<tr>
<td>Boolean</td>
<td>Modified</td>
<td>Read-only</td>
</tr>
<tr>
<td>Boolean</td>
<td>Protected</td>
<td>Read-only</td>
</tr>
<tr>
<td>Boolean</td>
<td>Numeric</td>
<td>Read-only</td>
</tr>
<tr>
<td>Boolean</td>
<td>HighIntensity</td>
<td>Read-only</td>
</tr>
<tr>
<td>Boolean</td>
<td>PenDetectable</td>
<td>Read-only</td>
</tr>
<tr>
<td>Boolean</td>
<td>Display</td>
<td>Read-only</td>
</tr>
</tbody>
</table>

**StartRow**

This collection element property is the row position of the first character in a given field in the autECLFieldList collection. StartRow is a Long data type and is read-only. The following example shows this property.

```vba
Dim StartRow as Long
Dim StartCol as Long
Dim autECLPSObj as Object
Dim autECLConnList as Object
Set autECLPSObj = CreateObject("PCOMM.autECLPS")
Set autECLConnList = CreateObject("PCOMM.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
End If
```

**StartCol**

This collection element property is the column position of the first character in a given field in the autECLFieldList collection. StartCol is a Long data type and is read-only. The following example shows this property.
Dim StartRow as Long
Dim StartCol as Long
Dim autECLPSObj as Object
Dim autECLConnList as Object
Set autECLPSObj = CreateObject("PCOMM.autECLPS")
Set autECLConnList = CreateObject("PCOMM.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
This collection element property is the row position of the last character in a given field in the autECLFieldList collection. EndRow is a Long data type and is read-only. The following example shows this property.

Dim EndRow as Long
Dim EndCol as Long
Dim autECLPSObj as Object
Dim autECLConnList as Object
Set autECLPSObj = CreateObject("PCOMM.autECLPS")
Set autECLConnList = CreateObject("PCOMM.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
This collection element property is the column position of the last character in a given field in the autECLFieldList collection. EndCol is a Long data type and is read-only. The following example shows this property.

Dim EndRow as Long
Dim EndCol as Long
Dim autECLPSObj as Object
Dim autECLConnList as Object
Set autECLPSObj = CreateObject("PCOMM.autECLPS")
Set autECLConnList = CreateObject("PCOMM.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
This collection element property is the length of a given field in the autECLFieldList collection. Length is a Long data type and is read-only. The following example shows this property.

```vbscript
Dim Len as Long
Dim autECLPSObj as Object
Dim autECLConnList as Object
Set autECLPSObj = CreateObject("PCOMM.autECLPS")
Set autECLConnList = CreateObject("PCOMM.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
This collection element property indicates if a given field in the autECLFieldList collection has a modified attribute. Modified is a Boolean data type and is read-only. The following example shows this property.

```vbscript
Dim autECLPSObj as Object
Dim autECLConnList as Object
Set autECLPSObj = CreateObject("PCOMM.autECLPS")
Set autECLConnList = CreateObject("PCOMM.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
```

Protected
This collection element property indicates if a given field in the autECLFieldList collection has a protected attribute. Protected is a Boolean data type and is read-only. The following example shows this property.

```vbscript
Dim autECLPSObj as Object
Dim autECLConnList as Object
Set autECLPSObj = CreateObject("PCOMM.autECLPS")
Set autECLConnList = CreateObject("PCOMM.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**
This collection element property indicates if a given field in the autECLFieldList collection has a numeric input only attribute. Numeric is a Boolean data type and is read-only. The following example shows this property.

```vbnet
dim autECLPSObj as Object
dim autECLConnList as Object
set autECLPSObj = createobject("PCOMM.autECLPS")
set autECLConnList = createobject("PCOMM.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

dim autECLPSObj as Object
set autECLConnList = createobject("PCOMM.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
```

**HighIntensity**
This collection element property indicates if a given field in the autECLFieldList collection has a high intensity attribute. HighIntensity is a Boolean data type and is read-only. The following example shows this property.

```vbnet
dim autECLPSObj as Object
dim autECLConnList as Object
set autECLPSObj = createobject("PCOMM.autECLPS")
set autECLConnList = createobject("PCOMM.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**
This collection element property indicates if a given field in the autECLFieldList collection has a pen detectable attribute. PenDetectable is a Boolean data type and is read-only. The following example shows this property.

```vbnet
dim autECLPSObj as Object
dim autECLConnList as Object
set autECLPSObj = createobject("PCOMM.autECLPS")
set autECLConnList = createobject("PCOMM.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
This collection element property indicates whether a given field in the autECLFieldList collection has a display attribute. Display is a Boolean data type and is read-only. The following example shows this property.

```vba
Dim autECLPSObj as Object
Dim autECLConnList as Object
Set autECLPSObj = CreateObject("PCOMM.autECLPS")
Set autECLConnList = CreateObject("PCOMM.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 Methods
The following section describes the methods that are valid for the autECLFieldList object.

```vba
void Refresh()
Object FindFieldByRowCol(Long Row, Long Col)
Object FindFieldByText(String text, [optional] Long Direction, [optional] Long StartRow, [optional] Long StartCol)
```

Collection Element Methods
The following collection element methods are valid for each item in the list.

```vba
String GetText()
void SetText(String Text)
```

Refresh
The Refresh method gets a snapshot of all the fields.

**Note:** You should call the Refresh method before accessing the field collection to ensure that you have current field data.

**Prototype**
```vba
void Refresh()
```

**Parameters**
None

**Return Value**
None

**Example**
The following example shows how to get a snapshot of all the fields for a given presentation space for a given plane.
Dim NumFields as long
Dim autECLPSObj as Object
Dim autECLConnList as Object
Set autECLPSObj = CreateObject("PCOMM.autECLPS")
Set autECLConnList = CreateObject("PCOMM.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
This method searches the autECLFieldList object for a field containing the given row and column coordinates. The value returned is a collection element object in the autECLFieldList collection.

Prototype
Object FindFieldByRowCol(Long Row, Long Col)

Parameters
Long Row Field row to search for.
Long Col Field column to search for.

Return Value
Object Dispatch object for the autECLFieldList collection item.

Example
The following example shows how to search the autECLFieldList object for a field containing the given row and column coordinates.

Dim autECLPSObj as Object
Dim autECLConnList as Object
Dim FieldElement as Object
Set autECLPSObj = CreateObject("PCOMM.autECLPS")
Set autECLConnList = CreateObject("PCOMM.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("IBM")

FindFieldByText
This method searches the autECLFieldList object for a field containing the string passed in as Text. The value returned is a collection element object in the autECLFieldList collection.

Prototype
Object FindFieldByText(String Text, [optional] Long Direction, [optional] Long StartRow, [optional] Long StartCol)
**autECLFieldList**

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String Text</td>
<td>The text string to search for.</td>
</tr>
<tr>
<td>Long StartRow</td>
<td>Row position in the presentation space at which to begin the search.</td>
</tr>
<tr>
<td>Long StartCol</td>
<td>Column position in the presentation space at which to begin the search.</td>
</tr>
<tr>
<td>Long Direction</td>
<td>Direction in which to search. Values are 1 for search forward, 2 for search backward</td>
</tr>
</tbody>
</table>

**Return Value**

**Object**

Dispatch object for the autECLFieldList collection item.

**Example**

The following example shows how to search the autECLFieldList object for a field containing the string passed in as text.

```vba
Dim autECLPSObj As Object
Dim autECLConnList As Object
Dim FieldElement As Object

Set autECLPSObj = CreateObject("PCOMM.autECLPS")
Set autECLConnList = CreateObject("PCOMM.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 "IBM"

' Or... search starting at row 2 col 1
set FieldElement = autECLPSObj.autECLFieldList.FindFieldByText "IBM", 2, 1
' Or... search starting at row 2 col 1 going backwards
set FieldElement = autECLPSObj.autECLFieldList.FindFieldByText "IBM", 2, 2, 1

FieldElement.SetText("Hello.")
```

**GetText**

The collection element method GetText retrieves the characters of a given field in an autECLFieldList item.

**Prototype**

String GetText()

**Parameters**

None

**Return Value**

String Field text.

**Example**

The following example shows how to use the GetText method.

```vba
Dim autECLPSObj As Object
Dim TestStr As String

' Initialize the connection
```
Set autECLPSObj = CreateObject("PCOMM.autECLPS")
autECLPSObj.SetConnectionByName("A")
autECLPSObj.autECLFieldList.Refresh()

TextStr = autECLPSObj.autECLFieldList(1).GetText()
SetText
This method populates a given field in an autECLFieldList item with the character string passed in as text. If the text exceeds the length of the field, the text is truncated.

Prototype
void SetText(String Text)

Parameters
String text String to set in field

Return Value
None

Example
The following example shows how to populate the field in an autECLFieldList item with the character string passed in as text.

Dim NumFields as Long
Dim autECLPSObj as Object
Dim autECLConnList as Object
Set autECLPSObj = CreateObject("PCOMM.autECLPS")
Set autECLConnList = CreateObject("PCOMM.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("IBM is a cool company")

autECLOIA Class

The autECLOIA object retrieves status from the Host Operator Information Area. Its name in the registry is PCOMM.autECLOIA.

You must initially set the connection for the object you create. Use SetConnectionByName or SetConnectionByHandle to initialize your object. The connection may be set only once. After the connection is set, any further calls to the set connection methods cause an exception. If you do not set the connection and try to access a property or method, an exception is also raised.

Note: The autECLOIA object in the autECLSession object is set by the autECLSession object.

The following example shows how to create and set the autECLOIA object in Visual Basic.

DIM autECLOIA as Object
Set autECLOIA = CreateObject("PCOMM.autECLOIA")
autECLOIA.SetConnectionByName("A")
Properties

This section describes the properties for the autECLOIA object.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>Alphanumeric</td>
<td>Read-only</td>
</tr>
<tr>
<td>Boolean</td>
<td>APL</td>
<td>Read-only</td>
</tr>
<tr>
<td>Boolean</td>
<td>Katakana</td>
<td>Read-only</td>
</tr>
<tr>
<td>Boolean</td>
<td>Hiragana</td>
<td>Read-only</td>
</tr>
<tr>
<td>Boolean</td>
<td>DBCS</td>
<td>Read-only</td>
</tr>
<tr>
<td>Boolean</td>
<td>UpperShift</td>
<td>Read-only</td>
</tr>
<tr>
<td>Boolean</td>
<td>Numeric</td>
<td>Read-only</td>
</tr>
<tr>
<td>Boolean</td>
<td>CapsLock</td>
<td>Read-only</td>
</tr>
<tr>
<td>Boolean</td>
<td>InsertMode</td>
<td>Read-only</td>
</tr>
<tr>
<td>Boolean</td>
<td>CommErrorReminder</td>
<td>Read-only</td>
</tr>
<tr>
<td>Boolean</td>
<td>MessageWaiting</td>
<td>Read-only</td>
</tr>
<tr>
<td>Long</td>
<td>InputInhibited</td>
<td>Read-only</td>
</tr>
<tr>
<td>String</td>
<td>Name</td>
<td>Read-only</td>
</tr>
<tr>
<td>Long</td>
<td>Handle</td>
<td>Read-only</td>
</tr>
<tr>
<td>String</td>
<td>ConnType</td>
<td>Read-only</td>
</tr>
<tr>
<td>Long</td>
<td>CodePage</td>
<td>Read-only</td>
</tr>
<tr>
<td>Boolean</td>
<td>Started</td>
<td>Read-only</td>
</tr>
<tr>
<td>Boolean</td>
<td>CommStarted</td>
<td>Read-only</td>
</tr>
<tr>
<td>Boolean</td>
<td>APIEnabled</td>
<td>Read-only</td>
</tr>
<tr>
<td>Boolean</td>
<td>Ready</td>
<td>Read-only</td>
</tr>
<tr>
<td>Boolean</td>
<td>NumLock</td>
<td>Read-only</td>
</tr>
</tbody>
</table>

Alphanumeric

This property queries the operator information area to determine whether the field at the cursor location is alphanumeric. Alphanumeric is a Boolean data type and is read-only. The following example shows this property.

```vbscript
DIM autECLOIA as Object
DIM autECLConnList as Object
Set autECLOIA = CreateObject("PCOMM.autECLOIA")
Set autECLConnList = CreateObject("PCOMM.autECLConnList")

' Initialize the connection
autECLConnList.Refresh
autECLOIA.SetConnectionByHandle(autECLConnList(1).Handle)

If autECLOIA.Alphanumeric Then...
```

APL

This property queries the operator information area to determine whether the keyboard is in APL mode. APL is a Boolean data type and is read-only. The following example shows this property.

```vbscript
DIM autECLOIA as Object
DIM autECLConnList as Object
Set autECLOIA = CreateObject("PCOMM.autECLOIA")
```
Set autECLOIA = CreateObject("PCOMM.autECLOIA")
' Initialize the connection
autECLOIA.Refresh
autECLOIA.SetConnectionByHandle(autECLOIA.Handle)
' Check if the keyboard is in APL mode
if autECLOIA.APL Then...

**Katakana**
This property queries the operator information area to determine whether Katakana characters are enabled. Katakana is a Boolean data type and is read-only. The following example shows this property.

```vbs
DIM autECLOIA as Object
DIM autECLOIA as Object
Set autECLOIA = CreateObject("PCOMM.autECLOIA")
Set autECLOIA = CreateObject("PCOMM.autECLOIA")
' Initialize the connection
autECLOIA.Refresh
autECLOIA.SetConnectionByHandle(autECLOIA.Handle)
' Check if Katakana characters are available
if autECLOIA.Katakana Then...
```

**Hiragana**
This property queries the operator information area to determine whether Hiragana characters are enabled. Hiragana is a Boolean data type and is read-only. The following example shows this property.

```vbs
DIM autECLOIA as Object
DIM autECLOIA as Object
Set autECLOIA = CreateObject("PCOMM.autECLOIA")
Set autECLOIA = CreateObject("PCOMM.autECLOIA")
' Initialize the connection
autECLOIA.Refresh
autECLOIA.SetConnectionByHandle(autECLOIA.Handle)
' Check if Hiragana characters are available
if autECLOIA.Hiragana Then...
```

**DBCS**
This property queries the operator information area to determine whether the field at the cursor location is DBCS. DBCS is a Boolean data type and is read-only. The following example shows this property.

```vbs
DIM autECLOIA as Object
DIM autECLOIA as Object
Set autECLOIA = CreateObject("PCOMM.autECLOIA")
Set autECLOIA = CreateObject("PCOMM.autECLOIA")
' Initialize the connection
autECLOIA.Refresh
autECLOIA.SetConnectionByHandle(autECLOIA.Handle)
' Check if DBCS is available
if autECLOIA.DBCS Then...
```
**UpperShift**
This property queries the operator information area to determine whether the keyboard is in uppershift mode. Uppershift is a Boolean data type and is read-only. The following example shows this property.

```vba
DIM autECLOIA as Object
DIM autECLConnList as Object

Set autECLOIA = CreateObject("PCOMM.autECLOIA")
Set autECLConnList = CreateObject("PCOMM.autECLConnList")

' Initialize the connection
autECLConnList.Refresh
autECLOIA.SetConnectionByHandle(autECLConnList(1).Handle)

' Check if the keyboard is in uppershift mode
If autECLOIA.UpperShift then...
```

**Numeric**
This property queries the operator information area to determine whether the field at the cursor location is numeric. Numeric is a Boolean data type and is read-only. The following example shows this property.

```vba
DIM autECLOIA as Object
DIM autECLConnList as Object

Set autECLOIA = CreateObject("PCOMM.autECLOIA")
Set autECLConnList = CreateObject("PCOMM.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**
This property queries the operator information area to determine if the keyboard CapsLock key is on. CapsLock is a Boolean data type and is read-only. The following example shows this property.

```vba
DIM autECLOIA as Object
DIM autECLConnList as Object

Set autECLOIA = CreateObject("PCOMM.autECLOIA")
Set autECLConnList = CreateObject("PCOMM.autECLConnList")

' Initialize the connection
autECLConnList.Refresh
autECLOIA.SetConnectionByHandle(autECLConnList(1).Handle)

' Check if the caps lock is on
If autECLOIA.CapsLock Then...
```

**InsertMode**
This property queries the operator information area to determine whether if the keyboard is in insert mode. InsertMode is a Boolean data type and is read-only. The following example shows this property.

```vba
DIM autECLOIA as Object
DIM autECLConnList as Object

Set autECLOIA = CreateObject("PCOMM.autECLOIA")
Set autECLConnList = CreateObject("PCOMM.autECLConnList")

' Initialize the connection
```
CommErrorReminder
This property queries the operator information area to determine whether a communications error reminder condition exists. CommErrorReminder is a Boolean data type and is read-only. The following example shows this property.

```
DIM autECLOIA as Object
DIM autECLConnList as Object
Set autECLOIA = CreateObject("PCOMM.autECLOIA")
Set autECLConnList = CreateObject("PCOMM.autECLConnList")
'
' Initialize the connection
autECLConnList.Refresh
autECLOIA.SetConnectionByHandle(autECLConnList(1).Handle)
'
' Check if comm error
If autECLOIA.CommErrorReminder Then...
```

MessageWaiting
This property queries the operator information area to determine whether the message waiting indicator is on. This can only occur for 5250 connections. MessageWaiting is a Boolean data type and is read-only. The following example shows this property.

```
DIM autECLOIA as Object
DIM autECLConnList as Object
Set autECLOIA = CreateObject("PCOMM.autECLOIA")
Set autECLConnList = CreateObject("PCOMM.autECLConnList")
'
' Initialize the connection
autECLConnList.Refresh
autECLOIA.SetConnectionByHandle(autECLConnList(1).Handle)
'
' Check if message waiting
If autECLOIA.MessageWaiting Then...
```

InputInhibited
This property queries the operator information area to determine whether keyboard input is inhibited. InputInhibited is a Long data type and is read-only. The following table shows valid values for InputInhibited.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not Inhibited</td>
<td>0</td>
</tr>
<tr>
<td>System Wait</td>
<td>1</td>
</tr>
<tr>
<td>Communication Check</td>
<td>2</td>
</tr>
<tr>
<td>Program Check</td>
<td>3</td>
</tr>
<tr>
<td>Machine Check</td>
<td>4</td>
</tr>
<tr>
<td>Other Inhibit</td>
<td>5</td>
</tr>
</tbody>
</table>

The following example shows this property.

```
DIM autECLOIA as Object
DIM autECLConnList as Object
Set autECLOIA = CreateObject("PCOMM.autECLOIA")
Set autECLConnList = CreateObject("PCOMM.autECLConnList")
' Initialize the connection
autECLConnList.Refresh
autECLOIA.SetConnectionByHandle(autECLConnList(1).Handle)

' Check if input inhibited
If not autECLOIA.InputInhibited = 0 Then...

Name
This property is the connection name string of the connection for which autECLOIA was set. Personal Communications only returns the short character ID (A-Z) in the string. There can be only one Personal Communications connection open with a given name. For example, there can be only one connection “A” open at a time. Name is a String data type and is read-only. The following example shows this property.

```
DIM Name as String
DIM Obj as Object
Set Obj = CreateObject("PCOMM.autECLOIA")

' Initialize the connection
Obj.SetConnectionByHandle("A")

' Save the name
Name = Obj.Name
```

Handle
This is the handle of the connection for which the autECLOIA object was set. There can be only one Personal Communications connection open with a given handle. For example, there can be only one connection “A” open at a time. Handle is a Long data type and is read-only. The following example shows this property.

```
DIM Obj as Object
Set Obj = CreateObject("PCOMM.autECLOIA")

' Initialize the connection
Obj.SetConnectionByHandle("A")

' Save the handle
Handle = Obj.Handle
```

ConnType
This is the connection type for which autECLOIA was set. This type may change over time. ConnType is a String data type and is read-only. The following example shows this property.

```
DIM Type as String
DIM Obj as Object
Set Obj = CreateObject("PCOMM.autECLOIA")

' Initialize the connection
Obj.SetConnectionByHandle("A")

' Save the type
Type = Obj.ConnType
```

Connection types for the ConnType property are:

<table>
<thead>
<tr>
<th>String Returned</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISP3270</td>
<td>3270 display</td>
</tr>
<tr>
<td>DISPS250</td>
<td>5250 display</td>
</tr>
<tr>
<td>PRNT3270</td>
<td>3270 printer</td>
</tr>
<tr>
<td>String Returned</td>
<td>Meaning</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------</td>
</tr>
<tr>
<td>PRNT5250</td>
<td>5250 printer</td>
</tr>
<tr>
<td>ASCII</td>
<td>VT emulation</td>
</tr>
</tbody>
</table>

**CodePage**

This is the code page of the connection for which autECLOIA was set. This code page may change over time. CodePage is a Long data type and is read-only. The following example shows this property.

```vba
DIM CodePage as Long
DIM Obj as Object
Set Obj = CreateObject("PCOMM.autECLOIA")

' Initialize the connection
Obj.SetConnectionByName("A")
' Save the code page
CodePage = Obj.CodePage
```

**Started**

This indicates whether the emulator window is started. The value is True if the window is open; otherwise, it is False. Started is a Boolean data type and is read-only. The following example shows this property.

```vba
DIM Hand as Long
DIM Obj as Object
Set Obj = CreateObject("PCOMM.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**

This indicates the status of the connection to the host. The value is True if the host is connected; otherwise, it is False. CommStarted is a Boolean data type and is read-only. The following example shows this property.

```vba
DIM Hand as Long
DIM Obj as Object
Set Obj = CreateObject("PCOMM.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**

This indicates whether the emulator is API-enabled. A connection may be enabled or disabled depending on the state of its API settings (in a Personal Communications window, choose **File -> API Settings**). The value is True if the emulator is enabled; otherwise, it is False. APIEnabled is a Boolean data type and is read-only. The following example shows this property.

```vba
DIM Hand as Long
DIM Obj as Object

Set Obj = CreateObject("PCOMM.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**

This indicates whether the emulator window is started, API-enabled, and connected. This property checks for all three properties. The value is True if the emulator is ready; otherwise, it is False. Ready is a Boolean data type and is read-only. The following example shows this property.

```vba
DIM Hand as Long
DIM Obj as Object

Set Obj = CreateObject("PCOMM.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**

This property queries the operator information area to determine if the keyboard NumLock key is on. NumLock is a Boolean data type and is read-only. The following example shows this property.

```vba
DIM autECLOIA as Object
DIM autECLConnList as Object

Set autECLOIA = CreateObject ("PCOMM.autECLOIA")
Set autECLConnList = CreateObject ("PCOMM.autECLConnList")

' Initialize the connection
autECLConnList.Refresh
autECLOIA.SetConnectionByFHandle (autECLConnList (1).Handle)

' Check if the num lock is on
If autECLOIA.NumLock Then ...
autECLOIA Methods

The following section describes the methods that are valid for autECLOIA.

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

This method registers an object to receive notification of all OIA events.

Prototype
void RegisterOIAEvent()

Parameters
None

Return Value
None

Example
See “Event Processing Example” on page 233 for an example.

UnregisterOIAEvent

Ends OIA event processing.

Prototype
void UnregisterOIAEvent()

Parameters
None

Return Value
None

Example
See “Event Processing Example” on page 233 for an example.

SetConnectionByName

The SetConnectionByName method uses the connection name to set the connection for a newly created autECLOIA object. In Personal Communications this connection name is the short connection ID (character A-Z). There can be only one Personal Communications connection open with a given name. For example, there can be only one connection “A” open at a time.

Note: Do not call this if using the autECLOIA object in autECLSession.
Prototype
void SetConnectionByName( String Name )

Parameters
String Name One-character string short name of the connection (A-Z).

Return Value
None

Example
The following example shows how to use the connection name to set the connection for a newly created autECLOIA object.

Set autECLOIA = CreateObject("PCOMM.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

Set Connection By Handle
The SetConnectionByHandle method uses the connection handle to set the connection for a newly created autECLOIA object. In Personal Communications this connection handle is a Long integer. There can be only one Personal Communications connection open with a given handle. For example, there can be only one connection “A” open at a time.

Note: Do not call this if using the autECLOIA object in autECLSession.

Prototype
void SetConnectionByHandle( Long Handle )

Parameters
Long Handle Long integer value of the connection to be set for the object.

Return Value
None

Example
The following example shows how to use the connection handle to set the connection for a newly created autECLOIA object.

Set autECLOIA = CreateObject("PCOMM.autECLOIA")
Set autECLConnList = CreateObject("PCOMM.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

StartCommunication

The StartCommunication collection element method connects the PCOMM emulator to the host data stream. This has the same effect as going to the PCOMM emulator Communication menu and choosing Connect.

Prototype
void StartCommunication()

Parameters
None

Return Value
None

Example
None
Dim OIAObj as Object
Dim autECLConnList as Object
Set autECLConnList = CreateObject("PCOMM.autECLConnList")
Set OIAObj = CreateObject("PCOMM.autECLOIA")

' Initialize the session
autECLConnList.Refresh
OIAObj.SetConnectionByHandle(autECLConnList(1).Handle)
OIAObj.StartCommunication()

StopCommunication

The StopCommunication collection element method disconnects the PCOMM emulator to the host data stream. This has the same effect as going to the PCOMM emulator Communication menu and choosing Disconnect.

Prototype
void StopCommunication()

Parameters
None

Return Value
None

Example
The following example shows how to connect a PCOMM emulator session to the host.
Dim OIAObj as Object
Dim autECLConnList as Object
Set autECLConnList = CreateObject("PCOMM.autECLConnList")
Set OIAObj = CreateObject("PCOMM.autECLOIA")

' Initialize the session
autECLOIA

autECLConnList.Refresh
OIAObj.SetConnectionByHandle(autECLConnList(1).Handle)

OIAObj.StopCommunication()

WaitForInputReady
The WaitForInputReady method waits until the OIA of the connection associated with the autECLOIA object indicates that the connection is able to accept keyboard input.

Prototype
Boolean WaitForInputReady([optional] Variant TimeOut)

Parameters

Variant TimeOut  The maximum length of time in milliseconds to wait, this parameter is optional. The default is Infinite.

Return Value
The method returns True if the condition is met, or False if the Timeout value is exceeded.

Example
Dim autECLOIAObj as Object
Set autECLOIAObj = CreateObject("PCOMM.autECLOIA")
autECLOIAObj.SetConnectionByName("A")
if (autECLOIAObj.WaitForInputReady(10000)) then
  msgbox "Ready for input"
else
  msgbox "Timeout Occurred"
end if

WaitForSystemAvailable
The WaitForSystemAvailable method waits until the OIA of the connection associated with the autECLOIA object indicates that the connection is connected to a host system.

Prototype
Boolean WaitForSystemAvailable([optional] Variant TimeOut)

Parameters

Variant TimeOut  The maximum length of time in milliseconds to wait, this parameter is optional. The default is Infinite.

Return Value
The method returns True if the condition is met, or False if the Timeout value is exceeded.

Example
Dim autECLOIAObj as Object
Set autECLOIAObj = CreateObject("PCOMM.autECLOIA")
autECLOIAObj.SetConnectionByName("A")
if (autECLOIAObj.WaitForSystemAvailable(10000)) then
msgbox "System Available"
else
msgbox "Timeout Occurred"
end if

**WaitForAppAvailable**

The **WaitForAppAvailable** method waits while the OIA of the connection associated with the autECLOIA object indicates that the application is being worked with.

**Prototype**

```
Boolean WaitForAppAvailable([optional] Variant TimeOut)
```

**Parameters**

- **Variant TimeOut**
  
  The maximum length of time in milliseconds to wait, this parameter is optional. The default is Infinite.

**Return Value**

The method returns True if the condition is met, or False if the Timeout value is exceeded.

**Example**

```vbnet
Dim autECLOIAObj as Object
Set autECLOIAObj = CreateObject("PCOMM.autECLOIA")
autECLOIAObj.SetConnectionByName("A")
if (autECLOIAObj.WaitForAppAvailable (10000)) then
msgbox "Application is available"
else
msgbox "Timeout Occurred"
end if
```

**WaitForTransition**

The **WaitForTransition** method waits for the OIA position specified of the connection associated with the autECLOIA object to change.

**Prototype**

```
Boolean WaitForTransition([optional] Variant Index, [optional] Variant timeout)
```

**Parameters**

- **Variant Index**
  
  The 1 byte Hex position of the OIA to monitor. This parameter is optional. The default is 3.

- **Variant TimeOut**
  
  The maximum length of time in milliseconds to wait, this parameter is optional. The default is Infinite.

**Return Value**

The method returns True if the condition is met, or False if the Timeout value is exceeded.

**Example**

```vbnet
Dim autECLOIAObj as Object
Dim Index
```
autECLOIA

Index = 03h

Set autECLOIAObj = CreateObject("PCOMM.autECLOIA")
autECLOIAObj.SetConnectionByName("A")

if (autECLOIAObj.WaitForTransition(Index,10000)) then
    msgbox "Position " " Index " " of the OIA Changed"
else
    msgbox "Timeout Occurred"
end if

CancelWaits

Cancels any currently active wait methods.

Prototype

void CancelWaits()

Parameters

None

Return Value

None

autECLOIA Events

The following events are valid for autECLOIA:

void NotifyOIAEvent()
void NotifyOIAError()
void NotifyOIAStop(Long Reason)

NotifyOIAEvent

A given OIA has occurred.

Prototype

void NotifyOIAEvent()

Parameters

None

Example

See "Event Processing Example" on page 233 for an example.

NotifyOIAError

This event occurs when an error occurs in the OIA.

Prototype

void NotifyOIAError()

Parameters

None

Example

See "Event Processing Example" on page 233 for an example.
NotifyOIAStop

This event occurs when event processing stops.

Prototype

void NotifyOIAStop(Long Reason)

Parameters

Long Reason Long Reason code for the stop. Currently, this will always be 0.

Event Processing Example

The following is a short example of how to implement OIA Events

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 Class

autECLPS performs operations on a presentation space. Its name in the registry is PCOMM.autECLPS.

You must initially set the connection for the object you create. Use
SetConnectionByName or SetConnectionByHandle to initialize your object. The
connection may be set only once. After the connection is set, any further calls to
the SetConnection methods cause an exception. If you do not set the connection
and try to access a property or method, an exception is also raised.

Notes:
1. In the presentation space, the first row coordinate is row 1 and the first column
coordinate is column 1. Therefore, the top, left position has a coordinate of row
1, column 1.
2. The autECLPS object in the autECLSession object is set by the autECLSession
object.
The following is an example of how to create and set the autECLPS object in Visual Basic.

```vba
DIM autECLPSObj as Object
DIM NumRows as Long
Set autECLPSObj = CreateObject("PCOMM.autECLPS")
' Initialize the connection
autECLPSObj.SetConnectionByName("A")
' For example, get the number of rows in the PS
NumRows = autECLPSObj.NumRows
```

### Properties

This section describes the properties of the autECLPS object.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object</td>
<td>autECLFieldList</td>
<td>Read-only</td>
</tr>
<tr>
<td>Long</td>
<td>NumRows</td>
<td>Read-only</td>
</tr>
<tr>
<td>Long</td>
<td>NumCols</td>
<td>Read-only</td>
</tr>
<tr>
<td>Long</td>
<td>CursorPosRow</td>
<td>Read-only</td>
</tr>
<tr>
<td>Long</td>
<td>CursorPosCol</td>
<td>Read-only</td>
</tr>
<tr>
<td>String</td>
<td>Name</td>
<td>Read-only</td>
</tr>
<tr>
<td>Long</td>
<td>Handle</td>
<td>Read-only</td>
</tr>
<tr>
<td>String</td>
<td>ConnType</td>
<td>Read-only</td>
</tr>
<tr>
<td>Long</td>
<td>CodePage</td>
<td>Read-only</td>
</tr>
<tr>
<td>Boolean</td>
<td>Started</td>
<td>Read-only</td>
</tr>
<tr>
<td>Boolean</td>
<td>CommStarted</td>
<td>Read-only</td>
</tr>
<tr>
<td>Boolean</td>
<td>APIEnabled</td>
<td>Read-only</td>
</tr>
<tr>
<td>Boolean</td>
<td>Ready</td>
<td>Read-only</td>
</tr>
</tbody>
</table>

### autECLFieldList

This is the field collection object for the connection associated with the autECLPS object. See "autECLFieldList Class" on page 211 for details. The following example shows this object.

```vba
Dim autECLPSObj as Object
Dim autECLConnList as Object
Set autECLPSObj = CreateObject("PCOMM.autECLPS")
Set autECLConnList = CreateObject("PCOMM.autECLConnList")

' Initialize the connection
autECLConnList.Refresh
autECLPSObj.SetConnectionByHandle(autECLConnList(1).Handle)

' Build the field list
CurPosCol = autECLPSObj.autECLFieldList.Refresh(1)
```

### NumRows

This is the number of rows in the presentation space for the connection associated with the autECLPS object. NumRows is a Long data type and is read-only. The following example shows this property.

```vba
Dim autECLPSObj as Object
Dim autECLConnList as Object
Dim Rows as Long
Set autECLPSObj = CreateObject("PCOMM.autECLPS")
Set autECLConnList = CreateObject("PCOMM.autECLConnList")
```
' Initialize the connection
autECLConnList.Refresh
autECLPSObj.SetConnectionByHandle(autECLConnList(1).Handle)
Rows = autECLPSObj.NumRows

**NumCols**
This is the number of columns in the presentation space for the connection associated with the autECLPS object. NumCols is a Long data type and is read-only. The following example shows this property.

Dim autECLPSObj as Object
Dim autECLConnList as Object
Dim Cols as Long
Set autECLPSObj = CreateObject("PCOMM.autECLPS")
Set autECLConnList = CreateObject("PCOMM.autECLConnList")

' Initialize the connection
autECLConnList.Refresh
autECLPSObj.SetConnectionByHandle(autECLConnList(1).Handle)
Cols = autECLPSObj.NumCols

**CursorPosRow**
This is the current row position of the cursor in the presentation space for the connection associated with the autECLPS object. CursorPosRow is a Long data type and is read-only. The following example shows this property.

Dim autECLPSObj as Object
Dim autECLConnList as Object
Dim CurPosRow as Long
Set autECLPSObj = CreateObject("PCOMM.autECLPS")
Set autECLConnList = CreateObject("PCOMM.autECLConnList")

' Initialize the connection
autECLConnList.Refresh
autECLPSObj.SetConnectionByHandle(autECLConnList(1).Handle)
CurPosRow = autECLPSObj.CursorPosRow

**CursorPosCol**
This is the current column position of the cursor in the presentation space for the connection associated with the autECLPS object. CursorPosCol is a Long data type and is read-only. The following example shows this property.

Dim autECLPSObj as Object
Dim autECLConnList as Object
Dim CurPosCol as Long
Set autECLPSObj = CreateObject("PCOMM.autECLPS")
Set autECLConnList = CreateObject("PCOMM.autECLConnList")

' Initialize the connection
autECLConnList.Refresh
autECLPSObj.SetConnectionByHandle(autECLConnList(1).Handle)
CurPosCol = autECLPSObj.CursorPosCol

**Name**
This is the connection name string of the connection for which autECLPS was set. Personal Communications only returns the short character ID (A-Z) in the string. There can be only one Personal Communications connection open with a given name. For example, there can be only one connection “A” open at a time. Name is a String data type and is read-only. The following example shows this property.

DIM Name as String
DIM Obj as Object
Set Obj = CreateObject("PCOMM.autECLPS")

' Initialize the connection
Obj.SetConnectionByName("A")

' Save the name
Name = Obj.Name

Handle
This is the handle of the connection for which the autECLPS object was set. There can be only one Personal Communications connection open with a given handle. For example, there can be only one connection “A” open at a time. Handle is a Long data type and is read-only. The following example shows this property.

DIM Obj as Object
Set Obj = CreateObject("PCOMM.autECLPS")

' Initialize the connection
Obj.SetConnectionByName("A")

' Save the connection handle
Hand = Obj.Handle

ConnType
This is the connection type for which autECLPS was set. This connection type may change over time. ConnType is a String data type and is read-only. The following example shows this property.

DIM Type as String
DIM Obj as Object

Set Obj = CreateObject("PCOMM.autECLPS")

' Initialize the connection
Obj.SetConnectionByName("A")

' Save the type
Type = Obj.ConnType

Connection types for the ConnType property are:

<table>
<thead>
<tr>
<th>String Returned</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISP3270</td>
<td>3270 display</td>
</tr>
<tr>
<td>DISP5250</td>
<td>5250 display</td>
</tr>
<tr>
<td>PRNT3270</td>
<td>3270 printer</td>
</tr>
<tr>
<td>PRNT5250</td>
<td>5250 printer</td>
</tr>
<tr>
<td>ASCII</td>
<td>VT emulation</td>
</tr>
</tbody>
</table>

CodePage
This is the code page of the connection for which autECLPS was set. This code page may change over time. CodePage is a Long data type and is read-only. The following example shows this property.

DIM CodePage as Long
DIM Obj as Object
Set Obj = CreateObject("PCOMM.autECLPS")

' Initialize the connection
Obj.SetConnectionByName("A")

' Save the code page
CodePage = Obj.CodePage
**Started**

This indicates if the connection emulator window is started. The value is True if the window is open; otherwise, it is False. Started is a Boolean data type and is read-only. The following example shows this property.

```vba
DIM Hand as Long
DIM Obj as Object

Set Obj = CreateObject("PCOMM.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**

This indicates the status of the connection to the host. The value is True if the host is connected; otherwise, it is False. CommStarted is a Boolean data type and is read-only. The following example shows this property.

```vba
DIM Hand as Long
DIM Obj as Object

Set Obj = CreateObject("PCOMM.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**

This indicates if the emulator is API-enabled. A connection may be enabled or disabled depending on the state of its API settings (in a Personal Communications window, choose File -> API Settings). The value is True if API is enabled; otherwise, it is False. APIEnabled is a Boolean data type and is read-only. The following example shows this property.

```vba
DIM Hand as Long
DIM Obj as Object

Set Obj = CreateObject("PCOMM.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**

This indicates whether the emulator window is started, API enabled and connected. This checks for all three properties. The value is True if the emulator is ready; otherwise, it is False. Ready is a Boolean data type and is read-only. The following example shows this property.

```vbs
DIM Hand as Long
DIM Obj as Object

Set Obj = CreateObject("PCOMM.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 Methods**

The following section describes the methods that are valid for the autECLPS object.

```plaintext
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)
void SetText(String Text, [optional] Long Row, [optional] Long Col)
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 WaitForWhileCursor(Variant Row, Variant Col, [optional]Variant TimeOut, [optional] Boolean bWaitForIr)
Boolean WaitForStringInRect(Variant WaitString, Variant sRow, Variant sCol, Variant eRow, Variant eCol, [optional] Variant nTimeOut, [optional] Boolean bWaitForIr, [optional] Boolean bCaseSens)
Boolean WaitForWhileStringInRect(Variant WaitString, Variant sRow, Variant sCol, Variant eRow, Variant eCol, [optional] Variant nTimeOut, [optional] Boolean bWaitForIr, [optional] Boolean bCaseSens)
Boolean WaitForScreen(Object screenDesc, [optional] Variant TimeOut)
Boolean WaitForWhileScreen(Object screenDesc, [optional] Variant TimeOut)
void CancelWaits()
```

**RegisterPSEvent**

This method registers an autECLPS object to receive notification of all changes to the PS of the connected session.

**Prototype**

```plaintext
void RegisterPSEvent()
```

**Parameters**

None
Return Value
None

Example
See “Event Processing Example” on page 260 for an example.

RegisterKeyEvent
Begins Keystroke Event Processing.

Prototype
void RegisterKeyEvent()

Parameters
None

Return Value
None

Example
See “Event Processing Example” on page 260 for an example.

RegisterCommEvent
This method registers an object to receive notification of all communication link connect/disconnect events.

Prototype
void RegisterCommEvent()

Parameters
None

Return Value
None

Example
See “Event Processing Example” on page 260 for an example.

UnregisterPSEvent
Ends PS Event Processing.

Prototype
void UnregisterPSEvent()

Parameters
None

Return Value
None

Example
See “Event Processing Example” on page 260 for an example.

UnregisterKeyEvent
Ends Keystroke Event Processing.
**Prototype**
void UnregisterKeyEvent()

**Parameters**
None

**Return Value**
None

**Example**
See “Event Processing Example” on page 260 for an example.

**UnregisterCommEvent**
Ends Communications Link Event Processing.

**Prototype**
void UnregisterCommEvent()

**Parameters**
None

**Return Value**
None

**SetConnectionByName**
This method uses the connection name to set the connection for a newly created autECLPS object. In Personal Communications this connection name is the short ID (character A-Z). There can be only one Personal Communications connection open with a given name. For example, there can be only one connection “A” open at a time.

**Note:** Do not call this if using the autECLPS object in autECLSession.

**Prototype**
void SetConnectionByName( String Name )

**Parameters**
String Name One-character string short name of the connection (A-Z).

**Return Value**
None

**Example**
The following example shows how to set the connection for a newly created autECLPS object using the connection name.

```
DIM autECLPSObj as Object
DIM NumRows as Long
Set autECLPSObj = CreateObject("PCOMM.autECLPS")
' Initialize the connection
autECLPSObj.SetConnectionByName("A")
' For example, get the number of rows in the PS
NumRows = autECLPSObj.NumRows
```
**SetConnectionByHandle**

This method uses the connection handle to set the connection for a newly created autECLPS object. In Personal Communications this connection handle is a Long integer. There can be only one Personal Communications connection open with a given handle. For example, there can be only one connection “A” open at a time.

**Note:** Do not call this if using the autECLPS object in autECLSession.

**Prototype**

```
void SetConnectionByHandle( Long Handle )
```

**Parameters**

- **Long Handle**
  
  Long integer value of the connection to be set for the object.

**Return Value**

None

**Example**

The following example shows how to set the connection for a newly created autECLPS object using the connection handle.

```vbs
DIM autECLPSObj as Object
DIM autECLConnList as Object
DIM NumRows as Long
Set autECLPSObj = CreateObject("PCOMM.autECLPS")
Set autECLConnList = CreateObject("PCOMM.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**

The SetCursorPos method sets the position of the cursor in the presentation space for the connection associated with the autECLPS object. The position set is in row and column units.

**Prototype**

```vbs
void SetCursorPos(Long Row, Long Col)
```

**Parameters**

- **Long Row**
  
  The row position of the cursor in the presentation space.

- **Long Col**
  
  The column position of the cursor in the presentation space.

**Return Value**

None

**Example**

The following example shows how to set the position of the cursor in the presentation space for the connection associated with the autECLPS object.

```vbs
DIM autECLPSObj as Object
DIM autECLConnList as Object
Set autECLPSObj = CreateObject("PCOMM.autECLPS")
```
Set autECLConnList = CreateObject("PCOMM.autECLConnList")
' Initialize the connection with the first in the list
autECLConnList.Refresh
autECLPSObj.SetConnectionByHandle(autECLConnList(1).Handle)
autECLPSObj.SetCursorPos 2, 1

SendKeys

The SendKeys method sends a string of keys to the presentation space for the connection associated with the autECLPS object. This method allows you to send mnemonic keystrokes to the presentation space. See Appendix A, “Sendkeys Mnemonic Keywords,” on page 397 for a list of these keystrokes.

Prototype
void SendKeys(String text, [optional] Long row, [optional] Long col)

Parameters

String text String of keys to send to the presentation space.
Long Row Row position to send keys to the presentation space. This parameter is optional. The default is the current cursor row position. If row is specified, col must also be specified.
Long Col Column position to send keys to the presentation space. This parameter is optional. The default is the current cursor column position. If col is specified, row must also be specified.

Return Value
None

Example
The following example shows how to use the SendKeys method to send a string of keys to the presentation space for the connection associated with the autECLPS object.
Dim autECLPSObj as Object
Dim autECLConnList as Object
Set autECLPSObj = CreateObject("PCOMM.autECLPS")
Set autECLConnList = CreateObject("PCOMM.autECLConnList")
' Initialize the connection
autECLConnList.Refresh
autECLPSObj.SetConnectionByHandle(autECLConnList(1).Handle)
autECLPSObj.SendKeys "IBM is a really cool company", 3, 1

SearchText

The SearchText method searches for the first occurrence of text in the presentation space for the connection associated with the autECLPS object. The search is case-sensitive. If text is found, the method returns a TRUE value. It returns a FALSE value if no text is found. If the optional row and column parameters are used, row and col are also returned, indicating the position of the text if it was found.

Prototype
boolean SearchText(String text, [optional] Long Dir, [optional] Long Row, [optional] Long Col)
Parameters

String text
- String to search for.

Long Dir
- Direction in which to search. Must either be 1 for search forward or 2 for search backward. This parameter is optional. The default is 1 for Forward.

Long Row
- Row position at which to start the search in the presentation space. The row of found text is returned if the search is successful. This parameter is optional. If row is specified, col must also be specified.

Long Col
- Column position at which to start the search in the presentation space. The column of found text is returned if the search is successful. This parameter is optional. If col is specified, row must also be specified.

Return Value
- TRUE if text is found, FALSE if text is not found.

Example
- The following example shows how to search for text in the presentation space for the connection associated with the autECLPS object.

```vba
Dim autECLPSObj as Object
Dim autECLConnList as Object
Dim Row as Long
Dim Col as Long
Set autECLPSObj = CreateObject("PCOMM.autECLPS")
Set autECLConnList = CreateObject("PCOMM.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 "IBM", 1, row, col) Then
    Call FoundFunc (row, col)
Else
    Call NotFoundFunc
Endif
```

GetText
- The GetText method retrieves characters from the presentation space for the connection associated with the autECLPS object.

Prototype

Parameters
- Long Row
  - Row position at which to start the retrieval in the presentation space. This parameter is optional.
Long Col

Column position at which to start the retrieval in the presentation space. This parameter is optional.

Long LenToGet

Number of characters to retrieve from the presentation space. This parameter is optional. The default is the length of the array passed in as BuffLen.

Return Value

String

Text from the PS.

Example

The following example shows how to retrieve a string from the presentation space for the connection associated with the autECLPS object.

```vba
Dim autECLPSObj as Object
Dim PSText as String

' Initialize the connection
Set autECLPSObj = CreateObject("PCOMM.autECLPS")
autECLPSObj.SetConnectionByName("A")

PSText = autECLPSObj.GetText(2,1,50)
```

SetText

The SetText method sends a string to the presentation space for the connection associated with the autECLPS object. Although this method is similar to the SendKeys method, this method does not send mnemonic keystrokes (for example, [enter] or [pf1]).

Prototype

```vba
void SetText(String Text, [optional] Long Row, [optional] Long Col)
```

Parameters

String Text

Character array to send.

Long Row

The row at which to begin the retrieval from the presentation space. This parameter is optional. The default is the current cursor row position.

Long Col

The column position at which to begin the retrieval from the presentation space. This parameter is optional. The default is the current cursor column position.

Return Value

None

Example

The following example shows how to search for text in the presentation space for the connection associated with the autECLPS object.

```vba
Dim autECLPSObj as Object

'Initialize the connection
Set autECLPSObj = CreateObject("PCOMM.autECLPS")
autECLPSObj.SetConnectionByName("A")
autECLPSObj.SetText"IBM is great", 2, 1
```
GetTextRect
The GetTextRect method retrieves characters from a rectangular area in the presentation space for the connection associated with the autECLPS object. No wrapping takes place in the text retrieval; only the rectangular area is retrieved.

Prototype
String GetTextRect(Long StartRow, Long StartCol, Long EndRow, Long EndCol)

Parameters
Long StartRow  Row at which to begin the retrieval in the presentation space.
Long StartCol  Column at which to begin the retrieval in the presentation space.
Long EndRow    Row at which to end the retrieval in the presentation space.
Long EndCol    Column at which to end the retrieval in the presentation space.

Return Value
String PS Text.

Example
The following example shows how to retrieve characters from a rectangular area in the presentation space for the connection associated with the autECLPS object.
Dim autECLPSObj as Object
Dim PSText String
' Initialize the connection
Set autECLPSObj = CreateObject("PCOMM.autECLPS")
autECLPSObj.SetConnectionByName("A")
PSText = GetTextRect(1,1,2,80)

StartCommunication
The StartCommunication collection element method connects the PCOMM emulator to the host data stream. This has the same effect as going to the PCOMM emulator Communication menu and choosing Connect.

Prototype
void StartCommunication()
' Initialize the session
autECLConnList.Refresh
PSObj.SetConnectionByHandle(autECLConnList(1).Handle)

PSObj.StartCommunication()

StopCommunication

The StopCommunication collection element method disconnects the PCOMM emulator to the host data stream. This has the same effect as going to the PCOMM emulator Communication menu and choosing Disconnect.

Prototype
void StopCommunication()

Parameters
None

Return Value
None

Example
The following example shows how to connect a PCOMM emulator session to the host.
Dim PSObj as Object
Dim autECLConnList as Object

Set autECLConnList = CreateObject("PCOMM.autECLConnList")
Set PSObj = CreateObject("PCOMM.autECLPS")

' Initialize the session
autECLConnList.Refresh
PSObj.SetConnectionByHandle(autECLConnList(1).Handle)

PSObj.StopCommunication()

StartMacro

The StartMacro method runs the Personal Communications macro file indicated by the MacroName parameter.

Prototype
void StartMacro(String MacroName)

Parameters

String MacroName Name of macro file located in the Personal Communications user-class application data directory (specified at installation), without the file extension. This method does not support long file names.

Return Value
None

Usage Notes
You must use the short file name for the macro name. This method does not support long file names.
**Example**
The following example shows how to start a macro.

```vba
Dim PS as Object

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

**Wait**
The Wait method waits for the number of milliseconds specified by the milliseconds parameter.

**Prototype**

```vba
void Wait(milliseconds as Long)
```

**Parameters**

- **Long milliseconds**
  - The number of milliseconds to wait.

**Return Value**

None

**Example**

```vba
Dim autECLPSObj as Object

Set autECLPSObj = CreateObject("PCOMM.autECLPS")
autECLPSObj.SetConnectionByName("A")
' Wait for 10 seconds
autECLPSObj.Wait(10000)
```

**WaitForCursor**
The WaitForCursor method waits for the cursor in the presentation space of the connection associated with the autECLPS object to be located at a specified position.

**Prototype**

```vba
Boolean WaitForCursor(Variant Row, Variant Col, [optional]Variant TimeOut, [optional] Boolean bWaitForIr)
```

**Parameters**

- **Variant Row**
  - Row position of the cursor.
- **Variant Col**
  - Column position of the cursor.
- **Variant TimeOut**
  - The maximum length of time in milliseconds to wait, this parameter is optional. The default is Infinite.
- **Boolean bWaitForIr**
  - If this value is true, after meeting the wait condition the function will wait until the OIA is ready to accept input. This parameter is optional. The default is False.

**Return Value**
The method returns True if the condition is met, or False if the Timeout value is exceeded.
Example
Dim autECLPSObj as Object
Dim Row, Col
Set autECLPSObj = CreateObject("PCOMM.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
The WaitWhileCursor method waits while the cursor in the presentation space of the connection associated with the autECLPS object is located at a specified position.

Prototype
Boolean WaitWhileCursor(Variant Row, Variant Col, [optional]Variant TimeOut, [optional] Boolean bWaitForIr)

Parameters
Variant Row
Row position of the cursor.

Variant Col
Column position of the cursor.

Variant TimeOut
The maximum length of time in milliseconds to wait, this parameter is optional. The default is Infinite.

Boolean bWaitForIr
If this value is true, after meeting the wait condition the function will wait until the OIA is ready to accept input. This parameter is optional. The default is False.

Return Value
The method returns True if the condition is met, or False if the Timeout value is exceeded.

Example
Dim autECLPSObj as Object
Dim Row, Col
Set autECLPSObj = CreateObject("PCOMM.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**

The WaitForString method waits for the specified string to appear in the presentation space of the connection associated with the autECLPS object. If the optional Row and Column parameters are used, the string must begin at the specified position. If 0,0 are passed for Row,Col the method searches the entire PS.

**Prototype**

```plaintext
Boolean WaitForString(Variant WaitString, [optional] Variant Row,
                        [optional] Variant Col, [optional] Variant TimeOut, [optional] Boolean bWaitForIr,
                        [optional] Boolean bCaseSens)
```

**Parameters**

- **Variant WaitString**
  - The string for which to wait.

- **Variant Row**
  - Row position that the string will begin. This parameter is optional. The default is 0.

- **Variant Col**
  - Column position that the string will begin. This parameter is optional. The default is 0.

- **Variant TimeOut**
  - The maximum length of time in milliseconds to wait, this parameter is optional. The default is Infinite.

- **Boolean bWaitForIr**
  - If this value is true, after meeting the wait condition the function will wait until the OIA is ready to accept input. This parameter is optional. The default is False.

- **Boolean bCaseSens**
  - If this value is True, the wait condition is verified as case-sensitive. This parameter is optional. The default is False.

**Return Value**

The method returns True if the condition is met, or False if the Timeout value is exceeded.

**Example**

```plaintext
Dim autECLPSObj as Object
Dim Row, Col, WaitString

Set autECLPSObj = CreateObject("PCOMM.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**

The WaitWhileString method waits while the specified string appears in the presentation space of the connection associated with the autECLPS object. If the
optional Row and Column parameters are used, the string must begin at the specified position. If 0,0 are passed for Row,Col the method searches the entire PS.

**Prototype**


**Parameters**

- **Variant WaitString**: The method waits while this string value exists.
- **Variant Row**: Row position that the string will begin. This parameter is optional. The default is 0.
- **Variant Col**: Column position that the string will begin. This parameter is optional. The default is 0.
- **Variant TimeOut**: The maximum length of time in milliseconds to wait, this parameter is optional. The default is Infinite.
- **Boolean bWaitForIr**: If this value is true, after meeting the wait condition the function will wait until the OIA is ready to accept input. This parameter is optional. The default is False.
- **Boolean bCaseSens**: If this value is True, the wait condition is verified as case-sensitive. This parameter is optional. The default is False.

**Return Value**

The method returns True if the condition is met, or False if the Timeout value is exceeded.

**Example**

Dim autECLPSObj as Object
Dim Row, Col, WaitString

Set autECLPSObj = CreateObject("PCOMM.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**

The WaitForStringInRect method waits for the specified string to appear in the presentation space of the connection associated with the autECLPS object in the specified rectangle.
Prototype
Boolean WaitForStringInRect(Variant WaitString, Variant sRow, Variant sCol,
Variant eRow, Variant eCol, [optional] Variant nTimeOut,
[optional] Boolean bWaitForIr, [optional] Boolean bCaseSens)

Parameters
Variant WaitString        The string for which to wait.
Variant sRow             Starting row position of the search rectangle.
Variant sCol             Starting column position of the search rectangle.
Variant eRow             Ending row position of the search rectangle.
Variant eCol             Ending column position of the search rectangle
Variant nTimeOut         The maximum length of time in milliseconds to wait, this parameter is optional. The default is Infinite.
Boolean bWaitForIr       If this value is true, after meeting the wait condition the function will wait until the OIA is ready to accept input. This parameter is optional. The default is False.
Boolean bCaseSens        If this value is True, the wait condition is verified as case-sensitive. This parameter is optional. The default is False.

Return Value
The method returns True if the condition is met, or False if the Timeout value is exceeded.

Example
Dim autECLPSObj as Object
Dim sRow, sCol, eRow, eCol, WaitString

Set autECLPSObj = CreateObject("PCOMM.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
   messagebox WaitString " " found in rectangle"
else
   messagebox "Timeout Occurred"
end if

WaitWhileStringInRect

The WaitWhileStringInRect method waits while the specified string appears in the presentation space of the connection associated with the autECLPS object in the specified rectangle.
Prototype
Boolean WaitWhileStringInRect(Variant WaitString, Variant sRow, Variant sCol,
Variant eRow, Variant eCol, [optional] Variant nTimeOut,
[optional] Boolean bWaitForIr, [optional] Boolean bCaseSens)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variant WaitString</td>
<td>The method waits while this string value exists.</td>
</tr>
<tr>
<td>Variant sRow</td>
<td>Starting row position of the search rectangle.</td>
</tr>
<tr>
<td>Variant sCol</td>
<td>Starting column position of the search rectangle.</td>
</tr>
<tr>
<td>Variant eRow</td>
<td>Ending row position of the search rectangle.</td>
</tr>
<tr>
<td>Variant eCol</td>
<td>Ending column position of the search rectangle.</td>
</tr>
<tr>
<td>Variant nTimeOut</td>
<td>The maximum length of time in milliseconds to wait, this parameter is optional. The default is Infinite.</td>
</tr>
<tr>
<td>Boolean bWaitForIr</td>
<td>If this value is true, after meeting the wait condition the function will wait until the OIA is ready to accept input. This parameter is optional. The default is False.</td>
</tr>
<tr>
<td>Boolean bCaseSens</td>
<td>If this value is True, the wait condition is verified as case-sensitive. This parameter is optional. The default is False.</td>
</tr>
</tbody>
</table>

Return Value
The method returns True if the condition is met, or False if the Timeout value is exceeded.

Example
Dim autECLPSObj as Object
Dim sRow, sCol, eRow, eCol, WaitString

Set autECLPSObj = CreateObject("PCOMM.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
The WaitForAttrib method will wait until the specified Attribute value appears in the presentation space of the connection associated with the autECLPS object at the specified Row/Column position. The optional MaskData parameter can be used to control which values of the attribute you are looking for. The optional plane parameter allows you to select any of the four PS planes.
Prototype

Boolean WaitForAttrib(Variant Row, Variant Col, Variant WaitData,
[optional] Variant MaskData, [optional] Variant plane, [optional] Variant TimeOut,
[optional] Boolean bWaitForIr)

Parameters

Variant Row
Row position of the attribute.

Variant Col
Column position of the attribute.

Variant WaitData
The 1-byte HEX value of the attribute to wait for.

Variant MaskData
The 1-byte HEX value to use as a mask with the attribute. This parameter is optional. The default value is 0xFF.

Variant plane
The plane of the attribute to get. The plane can have the following values:
1    Text Plane
2    Color Plane
3    Field Plane (default)
4    Extended Field Plane

Variant TimeOut
The maximum length of time in milliseconds to wait, this parameter is optional. The default is Infinite.

Boolean bWaitForIr
If this value is true, after meeting the wait condition the function will wait until the OIA is ready to accept input. This parameter is optional. The default is False.

Return Value
The method returns True if the condition is met, or False if the Timeout value is exceeded.

Example

Dim autECLPSObj as Object
Dim Row, Col, WaitData, MaskData, plane

Set autECLPSObj = CreateObject("PCOMM.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

The WaitWhileAttrib method waits while the specified Attribute value appears in the presentation space of the connection associated with the autECLPS object at the specified Row/Column position. The optional MaskData parameter can be used to control which values of the attribute you are looking for. The optional plane parameter allows you to select any of the four PS planes.

Prototype


Parameters

- Variant Row: Row position of the attribute.
- Variant Col: Column position of the attribute.
- Variant WaitData: The 1 byte HEX value of the attribute to wait for.
- Variant MaskData: The 1 byte HEX value to use as a mask with the attribute. This parameter is optional. The default value is 0xFF.
- Variant plane: The plane of the attribute to get. The plane can have the following values:
  - 1: Text Plane
  - 2: Color Plane
  - 3: Field Plane (default)
  - 4: Extended Field Plane
- Variant TimeOut: The maximum length of time in milliseconds to wait, this parameter is optional. The default is Infinite.
- Boolean bWaitForIr: If this value is true, after meeting the wait condition the function will wait until the OIA is ready to accept input. This parameter is optional. The default is False.

Return Value

The method returns True if the condition is met, or False if the Timeout value is exceeded.

Example

Dim autECLPSObj as Object
Dim Row, Col, WaitData, MaskData, plane

Set autECLPSObj = CreateObject("PCOMM.autECLPS")
autECLPSObj.SetConnectionByName("A")

Row = 20
Col = 16
WaitData = E8h
MaskData = FFh
plane = 3

if (autECLPSObj.WaitWhileAttrib(Row, Col, WaitData, MaskData, plane, 10000)) then
WaitForScreen

Synchronously waits for the screen described by the autECLScreenDesc parameter to appear in the Presentation Space.

**Note**: The wait for OIA input flag is set on the autECLScreenDesc object, it is not passed as a parameter to the wait method.

**Prototype**

Boolean WaitForScreen(Object screenDesc, [optional] Variant TimeOut)

**Parameters**

- **Object screenDesc**: autECLScreenDesc object that describes the screen (see “autECLScreenDesc Class” on page 261).
- **Variant TimeOut**: The maximum length of time in milliseconds to wait, this parameter is optional. The default is Infinite.

**Return Value**

The method returns True if the condition is met, or False if the Timeout value is exceeded.

**Example**

```vba
Dim autECLPSObj as Object
Dim autECLScreenDescObj as Object

Set autECLScreenDescObj = CreateObject("PCOMM.autECLScreenDesc")
Set autECLPSObj = CreateObject("PCOMM.autECLPS")
autECLPSObj.SetConnectionByName("A")

autECLScreenDescObj.AddCursorPos 23, 1

if (autECLPSObj.WaitForScreen(autECLScreenDescObj, 10000)) then
    msgbox "Screen found"
else
    msgbox "Timeout Occurred"
end if
```

WaitWhileScreen

Synchronously waits until the screen described by the autECLScreenDesc parameter is no longer in the Presentation Space.

**Note**: The wait for OIA input flag is set on the autECLScreenDesc object, it is not passed as a parameter to the wait method.

**Prototype**

Boolean WaitWhileScreen(Object screenDesc, [optional] Variant TimeOut)

**Parameters**

- **Object ScreenDesc**: autECLScreenDesc object that describes the screen (see “autECLScreenDesc Class” on page 261).
**Variant TimeOut**

The maximum length of time in milliseconds to wait, this parameter is optional. The default is Infinite.

**Return Value**

The method returns True if the condition is met, or False if the Timeout value is exceeded.

**Example**

```vba
Dim autECLPSObj as Object
Dim autECLScreenDescObj as Object

Set autECLScreenDescObj = CreateObject("PCOMM.autECLScreenDesc")
Set autECLPSObj = CreateObject("PCOMM.autECLPS")
autECLPSObj.SetConnectionByName("A")
autECLScreenDesObj.AddCursorPos 23, 1

if (autECLPSObj.WaitWhileScreen(autECLScreenDesObj, 10000)) then
    msgbox "Screen exited"
else
    msgbox "Timeout Occurred"
end if
```

**CancelWaits**

Cancels any currently active wait methods.

**Prototype**

```vba
void CancelWaits()
```

**Parameters**

None

**Return Value**

None

---

**autECLPS Events**

The following events are valid for autECLPS:

```vba
void NotifyPSEvent()
void NotifyKeyEvent(string KeyType, string KeyString, PassItOn as Boolean)
void NotifyCommEvent(boolean bConnected)
void NotifyPSError()
void NotifyKeyError()
void NotifyCommError()
void NotifyPSSstop(Long Reason)
void NotifyKeyStop(Long Reason)
void NotifyCommStop(Long Reason)
```

**NotifyPSEvent**

A given PS has been updated.

**Prototype**

```vba
void NotifyPSEvent()
```
### NotifyKeyEvent
A keystroke event has occurred and the key information has been supplied. This function can be used to intercept keystrokes to a given PS. The Key information is passed to the event handler and can be passed on, or another action can be performed.

**Note:** Only one object can have keystroke event handling registered to a given PS at a time.

**Prototype**
void NotifyKeyEvent(string KeyType, string KeyString, PassItOn as Boolean)

**Parameters**
- **String KeyType**
  Type of key intercepted.
  - M Mnemonic keystroke
  - A ASCII
- **String KeyString**
  Intercepted keystroke
- **Boolean PassItOn**
  Flag to indicate if the keystroke should be echoed to the PS.
  - TRUE Allows the keystroke to be passed on to the PS.
  - FALSE Prevents the keystroke from being passed to the PS.

**Example**
See “Event Processing Example” on page 260 for an example.

### NotifyCommEvent
A given communications link as been connected or disconnected.

**Prototype**
void NotifyCommEvent(boolean bConnected)

**Parameters**
- **boolean bConnected**
  True if Communications Link is currently Connected, False otherwise.

**Example**
See “Event Processing Example” on page 260 for an example.

### NotifyPSError
This event occurs when an error occurs in event processing.
**Prototype**

```c
void NotifyPSError()
```

**Parameters**

None

**Example**

See "Event Processing Example" on page 260 for an example.

---

**NotifyKeyError**

This event occurs when an error occurs in event processing.

**Prototype**

```c
void NotifyKeyError()
```

**Parameters**

None

**Example**

See "Event Processing Example" on page 260 for an example.

---

**NotifyCommError**

This event occurs when an error occurs in event processing.

**Prototype**

```c
void NotifyCommError()
```

**Parameters**

None

**Example**

See "Event Processing Example" on page 260 for an example.

---

**NotifyPSStop**

This event occurs when event processing stops.

**Prototype**

```c
void NotifyPSStop(Long Reason)
```

**Parameters**

- **Long Reason**: Reason code for the stop. Currently this will always be 0.

**Example**

See "Event Processing Example" on page 260 for an example.

---

**NotifyKeyStop**

This event occurs when event processing stops.

**Prototype**

```c
void NotifyKeyStop(Long Reason)
```
Parameters
Long Reason
Reason code for the stop. Currently this will always be 0.

Example
See “Event Processing Example” for an example.

NotifyCommStop
This event occurs when event processing stops.

Prototype
void NotifyCommStop(Long reason)

Parameters
Long Reason
Reason code for the stop. Currently this will always be 0.

Event Processing Example
The following is a short example of how to implement PS Events
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

'SThis event occurs if an error happens in PS event processing
Private Sub mPS_NotifyPSError()
'Do any error processing here
End Sub

'SThis event occurs when PS Event handling ends
Private Sub mPS_NotifyPSStop(Reason As Long)
'Do any stop processing here
End Sub

'SThis event occurs if an error happens in Keystroke processing
Private Sub mkey_NotifyKeyError()
'Do any error processing here
End Sub

'SThis event occurs when key stroke event handling ends
Private Sub mkey_NotifyKeyStop(Reason As Long)
'Do any stop processing here
End Sub

'SThis sub will get called when the Communication Link Status of the registered connection changes
Private Sub mPS_NotifyCommEvent()
' do your processing here
End Sub

'SThis event occurs if an error happens in Communications Link event processing
Private Sub mPS_NotifyCommError()
'Do any error processing here
End Sub

'SThis event occurs when Communications Status Notification ends
Private Sub mPS_NotifyCommStop()
'Do any stop processing here
End Sub

autECLScreenDesc Class

autECLScreenDesc is the class that is used to describe a screen for IBM's Host Access Class Library Screen Recognition Technology. It uses all four major planes of the presentation space to describe it (text, field, extended field, and color planes), as well as the cursor position.

Using the methods provided on this object, the programmer can set up a detailed description of what a given screen looks like in a host side application. Once an autECLScreenDesc object is created and set, it may be passed to either the synchronous WaitFor... methods provided on autECLPS, or it may be passed to autECLScreenReco, which fires an asynchronous event if the screen matching the autECLScreenDesc object appears in the PS.
autECLScreenDesc

autECLScreenDesc Methods

The following section describes the methods that are valid for autECLScreenDesc.

```plaintext
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

Adds an attribute value at the given position to the screen description.

**Prototype**

```plaintext
void AddAttrib(Variant attrib, Variant row, Variant col, Variant plane)
```

**Parameters**

- **Variant attrib**
  
The 1 byte HEX value of the attribute.
- **Variant row**
  
  Row position.
- **Variant col**
  
  Column position.
- **Variant plane**
  
  The plane of the attribute to get. The plane can have the following values:
  - 0. All Planes
  - 1. Text Plane
  - 2. Color Plane
  - 3. Field Plane
  - 4. Extended Field Plane
  - 5. DBCS Character Plane
  - 6. DBCS Grid Line Plane

**Return Value**

None

**Example**

```plaintext
Dim autECLPSObj as Object
Dim autECLScreenDescObj as Object

Set autECLScreenDescObj = CreateObject("PCOMM.autECLScreenDesc")
Set autECLPSObj = CreateObject("PCOMM.autECLPS")
autECLPSObj.SetConnectionByName "A"

autECLScreenDescObj.AddCursorPos 23, 1
autECLScreenDescObj.AddAttrib EBh, 1, 1, 2
autECLScreenDescObj.AddCursorPos 23,1
autECLScreenDescObj.AddNumFields 45
autECLScreenDescObj.AddNumInputFields 17
autECLScreenDescObj.AddOIAInhibitStatus 1
autECLScreenDescObj.AddString "LOGON", 23, 11, True
autECLScreenDescObj.AddStringInRect "PASSWORD", 23, 1, 24, 80, False
```
if (autECLPSObj.WaitForScreen(autECLScreenDesObj, 10000)) then
    msgbox "Screen reached"
else
    msgbox "Timeout Occurred"
end if

AddCursorPos
Sets the cursor position for the screen description to the given position.

Prototype
void AddCursorPos(Variant row, Variant col)

Parameters
Variant row           Row position.
Variant col           Column position.

Return Value
None

Example
Dim autECLPSObj as Object
Dim autECLScreenDescObj as Object

Set autECLScreenDescObj = CreateObject("PCOMM.autECLScreenDesc")
Set autECLPSObj = CreateObject("PCOMM.autECLPS")
autECLPSObj.SetConnectionByName "A"

autECLScreenDescObj.AddCursorPos 23, 1
autECLScreenDescObj.AddAttrib E8h, 1, 1, 2
autECLScreenDescObj.AddCursorPos 23,1
autECLScreenDescObj.AddNumFields 45
autECLScreenDescObj.AddNumInputFields 17
autECLScreenDescObj.AddOIAInhibitStatus 1
autECLScreenDescObj.AddString "LOGON", 23, 11, True
autECLScreenDescObj.AddStringInRect "PASSWORD", 23, 1, 24, 80, False

if (autECLPSObj.WaitForScreen(autECLScreenDescObj, 10000)) then
    msgbox "Screen reached"
else
    msgbox "Timeout Occurred"
end if

AddNumFields
Adds the number of fields to the screen description.

Prototype
void AddNumFields(Variant num)

Parameters
Variant num           Number of fields.

Return Value
None
Example
Dim autECLPSObj as Object
Dim autECLScreenDescObj as Object

Set autECLScreenDescObj = CreateObject("PCOMM.autECLScreenDesc")
Set autECLPSObj = CreateObject("PCOMM.autECLPS")
autECLPSObj.SetConnectionByName "A"

autECLScreenDescObj.AddCursorPos 23, 1
autECLScreenDescObj.AddAttrib E8h, 1, 1, 2
autECLScreenDescObj.AddCursorPos 23, 1
autECLScreenDescObj.AddNumFields 45
autECLScreenDescObj.AddNumInputFields 17
autECLScreenDescObj.AddOIAInhibitStatus 1
autECLScreenDescObj.AddString "LOGON", 23, 11, True
autECLScreenDescObj.AddStringInRect "PASSWORD", 23, 1, 24, 80, False

if (autECLPSObj.WaitForScreen(autECLScreenDescObj, 10000)) then
    msgbox "Screen reached"
else
    msgbox "Timeout Occurred"
end if

AddNumInputFields
Adds the number of fields to the screen description.

Prototype
void AddNumInputFields(Variant num)

Parameters
Variant num, Number of input fields.

Return Value
None

Example
Dim autECLPSObj as Object
Dim autECLScreenDescObj as Object

Set autECLScreenDescObj = CreateObject("PCOMM.autECLScreenDesc")
Set autECLPSObj = CreateObject("PCOMM.autECLPS")
autECLPSObj.SetConnectionByName "A"

autECLScreenDescObj.AddCursorPos 23, 1
autECLScreenDescObj.AddAttrib E8h, 1, 1, 2
autECLScreenDescObj.AddCursorPos 23, 1
autECLScreenDescObj.AddNumFields 45
autECLScreenDescObj.AddNumInputFields 17
autECLScreenDescObj.AddOIAInhibitStatus 1
autECLScreenDescObj.AddString "LOGON", 23, 11, True
autECLScreenDescObj.AddStringInRect "PASSWORD", 23, 1, 24, 80, False

if (autECLPSObj.WaitForScreen(autECLScreenDescObj, 10000)) then
    msgbox "Screen reached"
else
    msgbox "Timeout Occurred"
end if
AddOIAInhibitStatus

Sets the type of OIA monitoring for the screen description.

**Prototype**

```plaintext
void AddOIAInhibitStatus(Variant type)
```

**Parameters**

- **Variant type**: Type of OIA status. Valid values are as follows:
  - 0. Don't Care
  - 1. Not Inhibited

**Return Value**

None

**Example**

```plaintext
Dim autECLPSObj as Object
Dim autECLScreenDescObj as Object

Set autECLScreenDescObj = CreateObject("PCOMM.autECLScreenDesc")
Set autECLPSObj = CreateObject("PCOMM.autECLPS")
autECLPSObj.SetConnectionByName "A"

autECLScreenDescObj.AddCursorPos 23, 1
autECLScreenDescObj.AddAttrib EBh, 1, 1, 2
autECLScreenDescObj.AddCursorPos 23,1
autECLScreenDescObj.AddNumFields 45
autECLScreenDescObj.AddNumInputFields 17
autECLScreenDescObj.AddOIAInhibitStatus 1
autECLScreenDescObj.AddString "LOGON", 23, 11, True
autECLScreenDescObj.AddStringInRect "PASSWORD", 23, 1, 24, 80, False

if (autECLPSObj.WaitForScreen(autECLScreenDescObj, 10000)) then
    msgbox "Screen reached"
else
    msgbox "Timeout Occurred"
end if
```

AddString

Adds a string at the given location to the screen description.

** Prototype**

```plaintext
void AddString(String str, Variant row, Variant col, [optional] Boolean caseSense)
```

**Parameters**

- **String str**: String to add.
- **Variant row**: Row position.
- **Variant col**: Column position.
- **Boolean caseSense**: If this value is True, the strings are added as case-sensitive. This parameter is optional. The default is True.

**Return Value**

None
Example

Dim autECLPSObj as Object
Dim autECLScreenDescObj as Object

Set autECLScreenDescObj = CreateObject("PCOMM.autECLScreenDesc")
Set autECLPSObj = CreateObject("PCOMM.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

Adds a string in the given rectangle to the screen description.

Prototype

void AddStringInRect(String str, Variant sRow, Variant sCol,
            Variant eRow, Variant eCol, [optional] Variant caseSense)

Parameters

String str               String to add
Variant sRow             Upper left row position.
Variant sCol             Upper left column position.
Variant eRow             Lower right row position.
Variant eCol             Lower right column position.
Variant caseSense        If this value is True, the strings are added as case-sensitive. This parameter is optional. The default is True.

Return Value

None

Example

Dim autECLPSObj as Object
Dim autECLScreenDescObj as Object

Set autECLScreenDescObj = CreateObject("PCOMM.autECLScreenDesc")
Set autECLPSObj = CreateObject("PCOMM.autECLPS")
autECLPSObj.SetConnectionByName "A"

autECLScreenDesObj.AddCursorPos 23, 1
autECLScreenDesObj.AddAttrib E8h, 1, 1, 2
autECLScreenDesObj.AddCursorPos 23,1
autECLScreenDesObj.AddNumFields 45
Clear

Removes all description elements from the screen description.

Prototype

doctor Clear()

Parameters

None

Return Value

None

Example

Dim autECLPSObj as Object
Dim autECLScreenDescObj as Object
Set autECLScreenDescObj = CreateObject("PCOMM.autECLScreenDesc")
Set autECLPSObj = CreateObject("PCOMM.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 Class

The autECLScreenReco class is the engine for the Host Access Class Library screen recognition system. It contains the methods for adding and removing descriptions of screens. It also contains the logic for recognizing those screens and for asynchronously calling back to your event handler code for those screens.

Think of an object of the autECLScreenReco class as a unique recognition set. The object can have multiple autECLPS objects that it watches for screens, and multiple
screens to look for, and when it sees a registered screen in any of the added autECLPS objects it will fire event handling code defined in your application. All you need to do is set up your autECLScreenReco objects at the start of your application, and when any screen appears in any autECLPS that you want to monitor, your event code will get called by autECLScreenReco. You do absolutely no legwork in monitoring screens.

See “Event Processing Example” on page 271 for an example.

**autECLScreenReco Methods**

The following section describes the methods that are valid for autECLScreenReco.

```java
void AddPS(autECLPS ps)
Boolean IsMatch(autECLPS ps, AutECLScreenDesc sd)
void RegisterScreen(AutECLScreenDesc sd)
void RemovePS(autECLPS ps)
void UnregisterScreen(AutECLScreenDesc sd)
```

**AddPS**

Adds an autECLPS object to monitor to the autECLScreenReco Object.

**Prototype**

```java
void AddPS(autECLPS ps)
```

**Parameters**

- `autECLPS ps`: PS object to monitor.

**Return Value**

None

**Example**

See “Event Processing Example” on page 271 for an example.

**IsMatch**

Allows for passing an autECLPS object and an AutECLScreenDesc object and determining if the screen description matches the current state of the PS. The screen recognition engine uses this logic, but is provided so any routine can call it.

**Prototype**

```java
Boolean IsMatch(autECLPS ps, AutECLScreenDesc sd)
```

**Parameters**

- `autECLPS ps`: autPS object to compare.
- `AutECLScreenDesc sd`: autECLScreenDesc object to compare.

**Return Value**

True if the AutECLScreenDesc object matches the current screen in the PS, False otherwise.

**Example**

```java
Dim autPSObj as Object
Dim autECLScreenDescObj as Object
```
Set autECLScreenDescObj = CreateObject("PCOMM.autECLScreenDesc")
Set autPSObj = CreateObject("PCOMM.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

Begins monitoring all autECLPS objects added to the screen recognition object for the given screen description. If the screen appears in the PS, a NotifyRecoEvent will occur.

Prototype
void RegisterScreen(AutECLScreenDesc sd)

Parameters
AutECLScreenDesc sd Screen description object to register.

Return Value
None

Example
See "Event Processing Example" on page 271 for an example.

RemovePS

Removes the autECLPS object from screen recognition monitoring.

Prototype
void RemovePS(autECLPS ps)

Parameters
autECLPS ps autECLPS object to remove.

Return Value
None

Example
See "Event Processing Example" on page 271 for an example.

UnregisterScreen

Removes the screen description from screen recognition monitoring.

Prototype
void UnregisterScreen(AutECLScreenDesc sd)
### autECLScreenReco

**Parameters**

| AutECLScreenDesc sd | Screen description object to remove. |

**Return Value**

None

**Example**

See “Event Processing Example” on page 271 for an example.

### autECLScreenReco Events

The following events are valid for autECLScreenReco:

- void NotifyRecoEvent(AutECLScreenDesc sd, autECLPS ps)
- void NotifyRecoError()
- void NotifyRecoStop(Long Reason)

#### NotifyRecoEvent

This event occurs when a Registered Screen Description appears in a PS that was added to the autECLScreenReco object.

**Prototype**

void NotifyRecoEvent(AutECLScreenDesc sd, autECLPS ps)

**Parameters**

| AutECLScreenDesc sd | Screen Description object that had its criteria met. |
| autECLPS ps | PS object in which the match occurred. |

**Example**

See “Event Processing Example” on page 271 for an example.

#### NotifyRecoError

This event occurs when an error occurs in Event Processing.

**Prototype**

void NotifyRecoError()

**Parameters**

None

**Example**

See “Event Processing Example” on page 271 for an example.

#### NotifyRecoStop

This event occurs when event processing stops.

**Prototype**

void NotifyRecoStop(Long Reason)

**Parameters**

| Long Reason | Reason code for the stop. Currently this will always be 0. |
**Event Processing Example**

The following is a short example of how to implement Screen Recognition Events:

```vba
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("PCOMM.autECLScreenDesc")
    Set myPS = CreateObject("PCOMM.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

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 Class**

The autECLSession object provides general emulator related services and contains pointers to other key objects in the Host Access Class Library. Its name in the registry is PCOMM.autECLSession.

Although the objects that autECLSession contains are capable of standing on their own, pointers to them exist in the autECLSession class. When an autECLSession object is created, autECLPS, autECLOIA, autECLXfer, autECLWindowMetrics, autECLPageSettings, and autECLPrinterSettings objects are also created. Refer to them as you would any other property.
Notes:
1. The current version of this object is 1.2. There are two versions of this object; their ProgIDs in the registry are PCOMM.autECLSession.1 and PCOMM.autECLSession.2. The version-independent ProgID is PCOMM.autECLSession. The PCOMM.autECLSession.1 object does not support the properties autECLPageSettings and autECLPrinterSettings.
2. You must initially set the connection for the object you create. Use SetConnectionByName or SetConnectionByHandle to initialize your object. The connection can be set only once. After the connection is set, any further calls to the SetConnection methods cause an exception. If you do not set the connection and try to access an autECLSession property or method, an exception is also raised.

The following example shows how to create and set the autECLSession object in Visual Basic.

```vbscript
DIM SessObj as Object
Set SessObj = CreateObject("PCOMM.autECLSession")

' Initialize the session
SessObj.SetConnectionByName("A")
' For example, set the host window to minimized
SessObj.autECLWinMetrics.Minimized = True
```

Properties
This section describes the properties for the autECLSession object.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>Name</td>
<td>Read-only</td>
</tr>
<tr>
<td>Long</td>
<td>Handle</td>
<td>Read-only</td>
</tr>
<tr>
<td>String</td>
<td>ConnType</td>
<td>Read-only</td>
</tr>
<tr>
<td>Long</td>
<td>CodePage</td>
<td>Read-only</td>
</tr>
<tr>
<td>Boolean</td>
<td>Started</td>
<td>Read-only</td>
</tr>
<tr>
<td>Boolean</td>
<td>CommStarted</td>
<td>Read-only</td>
</tr>
<tr>
<td>Boolean</td>
<td>APIEnabled</td>
<td>Read-only</td>
</tr>
<tr>
<td>Boolean</td>
<td>Ready</td>
<td>Read-only</td>
</tr>
<tr>
<td>Object</td>
<td>autECLPS</td>
<td>Read-only</td>
</tr>
<tr>
<td>Object</td>
<td>autECLOIA</td>
<td>Read-only</td>
</tr>
<tr>
<td>Object</td>
<td>autECLXfer</td>
<td>Read-only</td>
</tr>
<tr>
<td>Object</td>
<td>autECLWinMetrics</td>
<td>Read-only</td>
</tr>
<tr>
<td>Object</td>
<td>autECLPageSettings</td>
<td>Read-only</td>
</tr>
<tr>
<td>Object</td>
<td>autECLPrinterSettings</td>
<td>Read-only</td>
</tr>
</tbody>
</table>

Name
This property is the connection name string of the connection for which autECLSession was set. Personal Communications only returns the short character ID (A-Z) in the string. There can be only one Personal Communications connection open with a given name. For example, there can be only one connection “A” open at a time. Name is a String data type and is read-only. The following example shows this property.
DIM Name as String
DIM SessObj as Object
Set SessObj = CreateObject("PCOMM.autECLSession")

' Initialize the session
SessObj.SetConnectionByName("A")

' Save the name
Name = SessObj.Name

Handle
This is the handle of the connection for which the autECLSession object was set. There can be only one Personal Communications connection open with a given handle. For example, there can be only one connection “A” open at a time. Handle is a Long data type and is read-only. The following example shows this property.

DIM SessObj as Object
Set SessObj = CreateObject("PCOMM.autECLSession")

' Initialize the session
SessObj.SetConnectionByName("A")

' Save the session handle
Hand = SessObj.Handle

ConnType
This is the connection type for which autECLXfer was set. This type may change over time. ConnType is a String data type and is read-only. The following example shows this property.

DIM Type as String
DIM SessObj as Object
Set SessObj = CreateObject("PCOMM.autECLSession")

' Initialize the session
SessObj.SetConnectionByName("A")

' Save the type
Type = SessObj.ConnType

Connection types for the ConnType property are:

<table>
<thead>
<tr>
<th>String Returned</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISP3270</td>
<td>3270 display</td>
</tr>
<tr>
<td>DISP5250</td>
<td>5250 display</td>
</tr>
<tr>
<td>PRNT3270</td>
<td>3270 printer</td>
</tr>
<tr>
<td>PRNT5250</td>
<td>5250 printer</td>
</tr>
<tr>
<td>ASCII</td>
<td>VT emulation</td>
</tr>
</tbody>
</table>

CodePage
This is the code page of the connection for which autECLXfer was set. This code page may change over time. CodePage is a Long data type and is read-only. The following example shows this property.

DIM CodePage as Long
DIM SessObj as Object
Set SessObj = CreateObject("PCOMM.autECLSession")
Initialize the session
SessObj.SetConnectionByName("A")
' Save the code page

Started
This indicates whether the emulator window is started. The value is True if the window is open; otherwise, it is False. Started is a Boolean data type and is read-only. The following example shows this property.

CommStarted
This indicates the status of the connection to the host. The value is True if the host is connected; otherwise, it is False. CommStarted is a Boolean data type and is read-only. The following example shows this property.

APIEnabled
This indicates whether the emulator is API-enabled. A connection may be enabled or disabled depending on the state of its API settings (in a Personal Communications window, choose **File -> API Settings**). The value is True if the emulator is enabled; otherwise, it is False. APIEnabled is a Boolean data type and is read-only. The following example shows this property.
Result.Text = "No"
Else
    Result.Text = "Yes"
End If

**Ready**
This indicates whether the emulator window is started, API-enabled, and connected. This property checks for all three properties. The value is True if the emulator is ready; otherwise, it is False. Ready is a Boolean data type and is read-only. The following example shows this property.

```vba
DIM Hand as Long
DIM SessObj as Object

Set SessObj = CreateObject("PCOMM.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 object**
The autECLPS object allows you to access the methods contained in the PCOMM.autECLPS class. See "autECLPS Class" on page 233 for more information. The following example shows this object.

```vba
DIM SessObj as Object
DIM PSSize as Long
Set SessObj = CreateObject("PCOMM.autECLSession")

' Initialize the session
SessObj.SetConnectionByName("A")
' For example, get the PS size
PSSize = SessObj.autECLPS.GetSize()
```

**autECLOIA object**
The autECLOIA object allows you to access the methods contained in the PCOMM.autECLOIA class. See "autECLOIA Class" on page 219 for more information. The following example shows this object.

```vba
DIM SessObj as Object
Set SessObj = CreateObject("PCOMM.autECLSession")

' Initialize the session
SessObj.SetConnectionByName("A")
' For example, set the host window to minimized
If (SessObj.autECLOIA.Katakana) Then
    ' whatever
Endif
```

**autECLXfer object**
The autECLXfer object allows you to access the methods contained in the PCOMM.autECLXfer class. See "autECLXfer Class" on page 294 for more information. The following example shows this object.

```vba
DIM SessObj as Object
Set SessObj = CreateObject("PCOMM.autECLSession")

' Initialize the session
SessObj.SetConnectionByName("A")
```
' For example
SessObj.Xfer.Sendfile "c:\temp\filename.txt",
   "filename text a0",
   "CRLF ASCII"

autECLWinMetrics object
The autECLWinMetrics object allows you to access the methods contained in the
PCOMM.autECLWinMetrics class. See “autECLWinMetrics Class” on page 281
for more information. The following example shows this object.

DIM SessObj as Object
Set SessObj = CreateObject("PCOMM.autECLSession")

' Initialize the session
SessObj.SetConnectionByName("A")
' For example, set the host window to minimized
SessObj.autECLWinMetrics.Minimized = True

autECLPageSettings object
The autECLPageSettings object enables you to access the methods contained in the
PCOMM.autECLPageSettings class. See “autECLPageSettings Class” on page 304
for more information.

The following example shows the autECLPageSettings object.

DIM SessObj as Object
Set SessObj = CreateObject("PCOMM.autECLSession")
' Initialize the session
SessObj.SetConnectionByName("A")

' For example, set the FaceName
SessObj.autECLPageSettings.FaceName = "Courier New"

The autECLPageSettings object is also supported in VBSCRIPT. The following
example shows how to use VBSCRIPT.

sub test()  
   autECLSession.SetConnectionByName(ThisSessionName)  
   autECLSession.autECLPageSettings.FaceName="Courier"
end sub

autECLPrinterSettings object
The autECLPrinterSettings object enables you to access the methods contained in
the PCOMM.autECLPrinterSettings class. See “autECLPageSettings Class” on page 304
for more information.

The following example shows the autECLPageSettings object.

DIM SessObj as Object
Set SessObj = CreateObject("PCOMM.autECLSession")
' Initialize the session
SessObj.SetConnectionByName("A")

' For example, set the Windows default printer
SessObj.autECLPrinterSettings.SetWinDefaultPrinter

The autECLPrinterSettings object is also supported in VBSCRIPT. The following
example shows how to use VBSCRIPT.

sub test()  
   autECLSession.SetConnectionByName(ThisSessionName)  
   autECLSession.autECLPrinterSettings.SetWinDefaultPrinter
end sub
autECLSession Methods

The following section describes the methods that are valid for the autECLSession object.

```java
void RegisterSessionEvent(Long updateType)
void RegisterCommEvent()
void UnregisterSessionEvent()
void UnregisterCommEvent()
void SetConnectionByName (String Name)
void SetConnectionByHandle (Long Handle)
void StartCommunication()
void StopCommunication()
```

**RegisterSessionEvent**

This method registers an autECLSession object to receive notification of specified Session events.

**Prototype**

```java
void RegisterSessionEvent(Long updateType)
```

**Parameters**

- **Long updateType**
  - Type of update to monitor for:
    1. PS Update
    2. OIA Update
    3. PS or OIA Update

**Return Value**

None

**Example**

See “Event Processing Example” on page 281 for an example.

**RegisterCommEvent**

This method registers an object to receive notification of all communication link connect/disconnect events.

**Prototype**

```java
void RegisterCommEvent()
```

**Parameters**

None

**Return Value**

None

**Example**

See “Event Processing Example” on page 281 for an example.

**UnregisterSessionEvent**

Ends Session Event processing.

**Prototype**

```java
void UnregisterSessionEvent()
```
UnregisterCommEvent

Ends Communications Link Event processing.

Prototype
void UnregisterCommEvent()

Parameters
None

Return Value
None

Example
See “Event Processing Example” on page 281 for an example.

SetConnectionByName

This method uses the connection name to set the connection for a newly created autECLSession object. In Personal Communications this connection name is the short ID (character A-Z). There can be only one Personal Communications connection open with a given name. For example, there can be only one connection “A” open at a time.

Prototype
void SetConnectionByName( String Name )

Parameters
String Name: One-character string short name of the connection (A-Z).

Return Value
None

Example
The following example shows how to use the connection name to set the connection for a newly created autECLSession object.

DIM SessObj as Object
Set SessObj = CreateObject("PCOMM.autECLSession")

' Initialize the session
SessObj.SetConnectionByName("A")
' For example, set the host window to minimized
SessObj.autECLWinMetrics.Minimized = True

SetConnectionByHandle

This method uses the connection handle to set the connection for a newly created autECLSession object. In Personal Communications this connection handle is a
long integer. There can be only one Personal Communications connection open with a given handle. For example, there can be only one connection “A” open at a time.

**Prototype**

void SetConnectionByHandle( Long Handle )

**Parameters**

Long Handle Long integer value of the connection to be set for the object.

**Return Value**

None

**Example**

The following example shows how to use the connection handle to set the connection for a newly created autECLSession object.

```vba
Dim SessObj as Object
Dim autECLConnList as Object

Set autECLConnList = CreateObject("PCOMM.autECLConnList")
Set SessObj = CreateObject("PCOMM.autECLSession")

' Initialize the session
autECLConnList.Refresh
SessObj.SetConnectionByHandle(autECLConnList(1).Handle)
```

**StartCommunication**

The StartCommunication collection element method connects the PCOMM emulator to the host data stream. This has the same effect as going to the PCOMM emulator Communication menu and choosing Connect.

**Prototype**

void StartCommunication()

**Parameters**

None

**Return Value**

None

**Example**

The following example shows how to connect a PCOMM emulator session to the host.

```vba
Dim SessObj as Object
Dim autECLConnList as Object

Set autECLConnList = CreateObject("PCOMM.autECLConnList")
Set SessObj = CreateObject("PCOMM.autECLSession")

' Initialize the session
autECLConnList.Refresh
SessObj.SetConnectionByHandle(autECLConnList(1).Handle)

SessObj.StartCommunication()
```
StopCommunication

The StopCommunication collection element method disconnects the PCOMM emulator to the host data stream. This has the same effect as going to the PCOMM emulator Communication menu and choosing Disconnect.

Prototype
void StopCommunication()

Parameters
None

Return Value
None

Example
The following example shows how to connect a PCOMM emulator session to the host.

```vba
Dim SessObj as Object
Dim autECLConnList as Object

Set autECLConnList = CreateObject("PCOMM.autECLConnList")
Set SessObj = CreateObject("PCOMM.autECLSession")

' Initialize the session
autECLConnList.Refresh
SessObj.SetConnectionByHandle(autECLConnList(1).Handle)
SessObj.StopCommunication()
```

autECLSession Events

The following events are valid for autECLSession:

- void NotifyCommEvent(boolean bConnected)
- void NotifyCommError()
- void NotifyCommStop(Long Reason)

NotifyCommEvent

A given communications link has been connected or disconnected.

Prototype
void NotifyCommEvent(boolean bConnected)

Parameters

boolean bConnected

Example
See "Event Processing Example" on page 281 for an example.

NotifyCommError

This event occurs when an error occurs in event processing.

Prototype
void NotifyCommError()
Parameters
None

Example
See "Event Processing Example" for an example.

NotifyCommStop
This event occurs when event processing stops.

Prototype
void NotifyCommStop(Long Reason)

Parameters
Long Reason
Reason code for the stop. Currently, this will always be 0.

Event Processing Example
The following is a short example of how to implement Session Events
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
collection 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 Class
The autECLWinMetrics object performs operations on an emulator window. It allows you to perform window rectangle and position manipulation (for example, SetWindowRect, Ypos and Width), as well as window state manipulation (for example, Visible or Restored). Its name in the registry is PCOMM.autECLWinMetrics.

You must initially set the connection for the object you create. Use SetConnectionByName or SetConnectionByHandle to initialize your object. The connection may be set only once. After the connection is set, any further calls to
the set connection methods cause an exception. If you do not set the connection and try to access a property or method, an exception is also raised.

**Note:** The autECLSession object in the autECL object is set by the autECL object.

The following example shows how to create and set the autECLWinMetrics object in Visual Basic.

```vbnet
dim autECLWinObj as object
set autECLWinObj = createobject("PCOMM.autECLWinMetrics")

' Initialize the connection
autECLWinObj.SetConnectionByName("A")

' For example, set the host window to minimized
autECLWinObj.Minimized = true
```

### Properties

This section describes the properties for the autECLWinMetrics object.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>WindowTitle</td>
<td>Read/Write</td>
</tr>
<tr>
<td>Long</td>
<td>Xpos</td>
<td>Read/Write</td>
</tr>
<tr>
<td>Long</td>
<td>Ypos</td>
<td>Read/Write</td>
</tr>
<tr>
<td>Long</td>
<td>Width</td>
<td>Read/Write</td>
</tr>
<tr>
<td>Long</td>
<td>Height</td>
<td>Read/Write</td>
</tr>
<tr>
<td>Boolean</td>
<td>Visible</td>
<td>Read/Write</td>
</tr>
<tr>
<td>Boolean</td>
<td>Active</td>
<td>Read/Write</td>
</tr>
<tr>
<td>Boolean</td>
<td>Minimized</td>
<td>Read/Write</td>
</tr>
<tr>
<td>Boolean</td>
<td>Maximized</td>
<td>Read/Write</td>
</tr>
<tr>
<td>Boolean</td>
<td>Restored</td>
<td>Read/Write</td>
</tr>
<tr>
<td>String</td>
<td>Name</td>
<td>Read-only</td>
</tr>
<tr>
<td>Long</td>
<td>Handle</td>
<td>Read-only</td>
</tr>
<tr>
<td>String</td>
<td>ConnType</td>
<td>Read-only</td>
</tr>
<tr>
<td>Long</td>
<td>CodePage</td>
<td>Read-only</td>
</tr>
<tr>
<td>Boolean</td>
<td>Started</td>
<td>Read-only</td>
</tr>
<tr>
<td>Boolean</td>
<td>CommStarted</td>
<td>Read-only</td>
</tr>
<tr>
<td>Boolean</td>
<td>APIEnabled</td>
<td>Read-only</td>
</tr>
<tr>
<td>Boolean</td>
<td>Ready</td>
<td>Read-only</td>
</tr>
</tbody>
</table>

### WindowTitle

This is the title that is currently in the title bar for the connection associated with the autECLWinMetrics object. This property may be both changed and retrieved. WindowTitle is a String data type and is read/write enabled. The following example shows this process. The following example shows this property.

```vbnet
dim autECLWinObj as object
dim ConnList as object
dim WinTitle as string
set autECLWinObj = createobject("PCOMM.autECLWinMetrics")
set ConnList = createobject("PCOMM.autECLConnList")

' Initialize the connection
ConnList.Refresh
```
autECLWinObj.SetConnectionByHandle(ConnList(1).Handle)

WindowTitle = autECLWinObj.WindowTitle 'get the window title

' or...

autECLWinObj.WindowTitle = "Flibberdeejibbet" 'set the window title

Usage Notes: If WindowTitle is set to blank, the window title of the connection is restored to its original setting.

Xpos
This is the x position of the upper left point of the emulator window rectangle. This property may be both changed and retrieved. Xpos is a Long data type and is read/write enabled. However, if the connection you are attached to is an inplace, embedded object, this property is read-only. The following example shows this property.

Dim autECLWinObj as Object
Dim ConnList as Object
Dim x as Long
Set autECLWinObj = CreateObject("PCOMM.autECLWinMetrics")
Set ConnList = CreateObject("PCOMM.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
This is the y position of the upper left point of the emulator window rectangle. This property may be both changed and retrieved. Ypos is a Long data type and is read/write enabled. However, if the connection you are attached to is an inplace, embedded object, this property is read-only. The following example shows this property.

Dim autECLWinObj as Object
Dim ConnList as Object
Dim y as Long
Set autECLWinObj = CreateObject("PCOMM.autECLWinMetrics")
Set ConnList = CreateObject("PCOMM.autECLConnList")

' Initialize the connection
ConnList.Refresh
autECLWinObj.SetConnectionByHandle(ConnList(1).Handle)

y = autECLWinObj.Ypos 'get the y position

' or...

autECLWinObj.Ypos = 6081 'set the y position

Width
This is the width of the emulator window rectangle. This property may be both changed and retrieved. Width is a Long data type and is read/write enabled. However, if the connection you are attached to is an inplace, embedded object, this property is read-only. The following example shows this property.
Dim autECLWinObj as Object
Dim ConnList as Object
Dim cx as Long
Set autECLWinObj = CreateObject("PCOMM.autECLWinMetrics")
Set ConnList = CreateObject("PCOMM.autECLConnList")

' Initialize the connection
ConnList.Refresh
autECLWinObj.SetConnectionByHandle(ConnList(1).Handle)

cx = autECLWinObj.Width 'get the width

' or...
autECLWinObj.Width = 6081 'set the width

Height
This is the height of the emulator window rectangle. This property may be both changed and retrieved. Height is a Long data type and is read/write enabled. However, if the connection you are attached to is an inplace, embedded object, this property is read-only. The following example shows this property.

Dim autECLWinObj as Object
Dim ConnList as Object
Dim cy as Long
Set autECLWinObj = CreateObject("PCOMM.autECLWinMetrics")
Set ConnList = CreateObject("PCOMM.autECLConnList")

' Initialize the connection
ConnList.Refresh
autECLWinObj.SetConnectionByHandle(ConnList(1).Handle)

cy = autECLWinObj.Height 'get the height

' or...
autECLWinObj.Height = 6081 'set the height

Visible
This is the visibility state of the emulator window. This property may be both changed and retrieved. Visible is a Boolean data type and is read/write enabled. However, if the connection you are attached to is an inplace, embedded object, this property is read-only. The following example shows this property.

Dim autECLWinObj as Object
Dim ConnList as Object
Set autECLWinObj = CreateObject("PCOMM.autECLWinMetrics")
Set ConnList = CreateObject("PCOMM.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

Active
This is the focus state of the emulator window. This property may be both changed and retrieved. Active is a Boolean data type and is read/write enabled. However, if the connection you are attached to is an inplace, embedded object, this property is read-only. The following example shows this property.
Dim autECLWinObj as Object
Dim ConnList as Object
Set autECLWinObj = CreateObject("PCOMM.autECLWinMetrics")
Set ConnList = CreateObject("PCOMM.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
This is the minimize state of the emulator window. This property may be both changed and retrieved. Minimized is a Boolean data type and is read/write enabled. However, if the connection you are attached to is an inplace, embedded object, this property is read-only. The following example shows this property.

Dim autECLWinObj as Object
Dim ConnList as Object
Set autECLWinObj = CreateObject("PCOMM.autECLWinMetrics")
Set ConnList = CreateObject("PCOMM.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
This is the maximize state of the emulator window. This property may be both changed and retrieved. Maximized is a Boolean data type and is read/write enabled. However, if the connection you are attached to is an inplace, embedded object, this property is read-only. The following example shows this property.

Dim autECLWinObj as Object
Dim ConnList as Object
Set autECLWinObj = CreateObject("PCOMM.autECLWinMetrics")
Set ConnList = CreateObject("PCOMM.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
This is the restore state of the emulator window. Restored is a Boolean data type and is read/write enabled. However, if the connection you are attached to is an inplace, embedded object, this property is read-only. The following example shows this property.
autECLWinMetrics

Dim autECLWinObj as Object
Dim SessList as Object
Set autECLWinObj = CreateObject("PCOMM.autECLWinMetrics")
Set SessList = CreateObject("PCOMM.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

Name
This property is the connection name string of the connection for which autECLWinMetrics was set. Currently, Personal Communications only returns the short character ID (A-Z) in the string. There can be only one Personal Communications connection open with a given name. For example, there can be only one connection “A” open at a time. Name is a String data type and is read-only. The following example shows this property.

DIM Name as String
DIM Obj as Object
Set Obj = CreateObject("PCOMM.autECLWinMetrics")

' Initialize the connection
Obj.SetConnectionByName("A")

' Save the name
Name = Obj.Name

Handle
This is the handle of the connection for which the autECLWinMetrics object was set. There can be only one Personal Communications connection open with a given handle. For example, there can be only one connection “A” open at a time. Handle is a Long data type and is read-only. The following example shows this property.

DIM Obj as Object
Set Obj = CreateObject("PCOMM.autECLWinMetrics")

' Initialize the connection
Obj.SetConnectionByName("A")

' Save the handle
Hand = Obj.Handle

ConnType
This is the connection type for which autECLWinMetrics was set. This type may change over time. ConnType is a String data type and is read-only. The following example shows this property.

DIM Type as String
DIM Obj as Object
Set Obj = CreateObject("PCOMM.autECLWinMetrics")

' Initialize the connection
Obj.SetConnectionByName("A")

' Save the type
Type = Obj.ConnType
Connection types for the ConnType property are:

<table>
<thead>
<tr>
<th>String Returned</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISP3270</td>
<td>3270 display</td>
</tr>
<tr>
<td>DISP5250</td>
<td>5250 display</td>
</tr>
<tr>
<td>PRNT3270</td>
<td>3270 printer</td>
</tr>
<tr>
<td>PRNT5250</td>
<td>5250 printer</td>
</tr>
<tr>
<td>ASCII</td>
<td>VT emulation</td>
</tr>
</tbody>
</table>

**CodePage**
This is the code page of the connection for which autECLWinMetrics was set. This code page may change over time. CodePage is a Long data type and is read-only. The following example shows this property.

1. Initialize the connection
2. Save the code page

```vbscript
DIM CodePage as Long
DIM Obj as Object
Set Obj = CreateObject("PCOMM.autECLWinMetrics")

' Initialize the connection
Obj.SetConnectionByName("A")
' Save the code page
CodePage = Obj.CodePage
```

**Started**
This indicates whether the emulator window is started. The value is True if the window is open; otherwise, it is False. Started is a Boolean data type and is read-only. The following example shows this property.

```vbscript
DIM Hand as Long
DIM Obj as Object
Set Obj = CreateObject("PCOMM.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**
This indicates the status of the connection to the host. The value is True if the host is connected; otherwise, it is False. CommStarted is a Boolean data type and is read-only. The following example shows this property.

```vbscript
DIM Hand as Long
DIM Obj as Object
Set Obj = CreateObject("PCOMM.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
```
APIEnabled
This indicates whether the emulator is API-enabled. A connection may be enabled or disabled depending on the state of its API settings (in a Personal Communications window, choose File ->API Settings). The value is True if the emulator is enabled; otherwise, it is False. APIEnabled is a Boolean data type and is read-only. The following example shows this property.

```vba
DIM Hand as Long
DIM Obj as Object

Set Obj = CreateObject("PCOMM.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
This indicates whether the emulator window is started, API enabled, and connected. This property checks for all three properties. The value is True if the emulator is ready; otherwise, it is False. Ready is a Boolean data type and is read-only. The following example shows this property.

```vba
DIM Hand as Long
DIM Obj as Object

Set Obj = CreateObject("PCOMM.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 Methods**

The following section describes the methods that are valid for the autECLWinMetrics object.

```vba
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

This method registers an object to receive notification of all communication link connect/disconnect events.

Prototype
void RegisterCommEvent()

Parameters
None

Return Value
None

Example
See "Event Processing Example" on page 293 for an example.

UnregisterCommEvent

Ends Communications Link Event Processing.

Prototype
void UnregisterCommEvent()

Parameters
None

Return Value
None

SetConnectionByName

This method uses the connection name to set the connection for a newly created autECLWinMetrics object. In Personal Communications this connection name is the short ID (character A-Z). There can be only one Personal Communications connection open with a given name. For example, there can be only one connection “A” open at a time.

Note: Do not call this if using the autECLWinMetrics object in autECLSession.

Prototype
void SetConnectionByName( String Name )

Parameters
String Name
One-character string short name of the connection (A-Z).

Return Value
None

Example
The following example shows how to use the connection name to set the connection for a newly created autECLWinMetrics object.
DIM autECLWinObj as Object
Set autECLWinObj = CreateObject("PCOMM.autECLWinMetrics")
SetConnectionByHandle

This method uses the connection handle to set the connection for a newly created autECLWinMetrics object. In Personal Communications this connection handle is a long integer. There can be only one Personal Communications connection open with a given handle. For example, there can be only one connection “A” open at a time.

Note: Do not call this if using the autECLWinMetrics object in autECLSession.

Prototype

void SetConnectionByHandle( Long Handle )

Parameters

Long Handle

Long integer value of the connection to be set for the object.

Return Value

None

Example

The following example shows how to use the connection handle to set the connection for a newly created autECLWinMetrics object.

DIM autECLWinObj as Object
DIM ConnList as Object
Set autECLWinObj = CreateObject("PCOMM.autECLWinMetrics")
Set ConnList = CreateObject("PCOMM.autECLConnList")

' Initialize the connection
ConnList.Refresh
autECLWinObj.SetConnectionByHandle(ConnList(1).Handle)
' For example, set the host window to minimized
autECLWinObj.Minimized = True

GetWindowRect

The GetWindowRect method returns the bounding points of the emulator window rectangle.

Prototype

void GetWindowRect(Variant Left, Variant Top, Variant Right, Variant Bottom)

Parameters

Variant Left, Top, Right, Bottom

Bounding points of the emulator window.

Return Value

None

Example

The following example shows how to return the bounding points of the emulator window rectangle.
SetWindowRect
The SetWindowRect method sets the bounding points of the emulator window rectangle.

Prototype
void SetWindowRect(Long Left, Long Top, Long Right, Long Bottom)

Parameters
Long Left, Top, Right, Bottom
Bounding points of the emulator window.

Return Value
None

Example
The following example shows how to set the bounding points of the emulator window rectangle.
Dim autECLWinObj as Object
Dim ConnList as Object
Set autECLWinObj = CreateObject("PCOMM.autECLWinMetrics")
Set ConnList = CreateObject("PCOMM.autECLConnList")
' Initialize the connection
ConnList.Refresh
autECLWinObj.SetConnectionByHandle(ConnList(1).Handle)
autECLWinObj.SetWindowRect 0, 0, 6081, 6081

StartCommunication
The StartCommunication collection element method connects the PCOMM emulator to the host data stream. This has the same effect as going to the PCOMM emulator Communication menu and choosing Connect.

Prototype
void StartCommunication()

Parameters
None

Return Value
None

Example
The following example shows how to connect a PCOMM emulator session to the host.
Dim WinObj as Object
Dim autECLConnList as Object

Set autECLConnList = CreateObject("PCOMM.autECLConnList")
Set WinObj = CreateObject("PCOMM.autECLWinMetrics")

' Initialize the session
autECLConnList.Refresh
WinObj.SetConnectionByHandle(autECLConnList(1).Handle)
WinObj.StartCommunication()

StopCommunication
The StopCommunication collection element method disconnects the PCOMM emulator to the host data stream. This has the same effect as going to the PCOMM emulator Communication menu and choosing Disconnect.

Prototype
void StopCommunication()

Parameters
None

Return Value
None

Example
The following example shows how to connect a PCOMM emulator session to the host.
Dim WinObj as Object
Dim autECLConnList as Object

Set autECLConnList = CreateObject("PCOMM.autECLConnList")
Set WinObj = CreateObject("PCOMM.autECLWinMetrics")

' Initialize the session
autECLConnList.Refresh
WinObj.SetConnectionByHandle(autECLConnList(1).Handle)
WinObj.StopCommunication()

autECLWinMetrics Events
The following events are valid for autECL WinMetrics:

void NotifyCommEvent(boolean bConnected)
NotifyCommError()
void NotifyCommStop(Long Reason)

NotifyCommEvent
A given communications link as been connected or disconnected.

Prototype
void NotifyCommEvent(boolean bConnected)

Parameters
boolean bConnected True if Communications Link is currently Connected, False otherwise.
Example
See "Event Processing Example" for an example.

NotifyCommError
This event occurs when an error occurs in Event Processing.

 Prototype
NotifyCommError()

 Parameters
None

 Example
See "Event Processing Example" for an example.

NotifyCommStop
This event occurs when event processing stops.

 Prototype
void NotifyCommStop(Long Reason)

 Parameters
Long Reason Reason code for the stop. Currently this will always be 0.

Event Processing Example
The following is a short example of how to implement WinMetrics Events.

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
c '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 Class**

The autECLXfer object provides file transfer services. Its name in the registry is PCOMM.autECLXfer.

You must initially set the connection for the object you create. Use SetConnectionByName or SetConnectionByHandle to initialize your object. The connection may be set only once. After the connection is set, any further calls to the SetConnection methods cause an exception. If you do not set the connection and try to access an autECLXfer property or method, an exception is also raised. The following shows how to create and set the autECLXfer object in Visual Basic.

```vbnet
DIM XferObj as Object
Set XferObj = CreateObject("PCOMM.autECLXfer")
' Initialize the connection
XferObj.SetConnectionByName("A")
```

### Properties

This section describes the properties for the autECLXfer object.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>Name</td>
<td>Read-only</td>
</tr>
<tr>
<td>Long</td>
<td>Handle</td>
<td>Read-only</td>
</tr>
<tr>
<td>String</td>
<td>ConnType</td>
<td>Read-only</td>
</tr>
<tr>
<td>Long</td>
<td>CodePage</td>
<td>Read-only</td>
</tr>
<tr>
<td>Boolean</td>
<td>Started</td>
<td>Read-only</td>
</tr>
<tr>
<td>Boolean</td>
<td>CommStarted</td>
<td>Read-only</td>
</tr>
<tr>
<td>Boolean</td>
<td>APIEnabled</td>
<td>Read-only</td>
</tr>
<tr>
<td>Boolean</td>
<td>Ready</td>
<td>Read-only</td>
</tr>
</tbody>
</table>

#### Name

This property is the connection name string of the connection for which autECLXfer was set. Personal Communications only returns the short character ID (A-Z) in the string. There can be only one Personal Communications connection open with a given name. For example, there can be only one connection “A” open at a time. Name is a String data type and is read-only. The following example shows this property.

```vbnet
DIM Name as String
DIM Obj as Object
Set Obj = CreateObject("PCOMM.autECLXfer")

' Initialize the connection
Obj.SetConnectionByName("A")

' Save the name
Name = Obj.Name
```

#### Handle

This is the handle of the connection for which the autECLXfer object was set. There can be only one Personal Communications connection open with a given handle. For example, there can be only one connection “A” open at a time. Handle is a Long data type and is read-only. The following example shows this property.
DIM Obj as Object
Set Obj = CreateObject("PCOMM.autECLXfer")

' Initialize the connection
Obj.SetConnectionByName("A")

' Save the handle
Hand = Obj.Handle

**ConnType**
This is the connection type for which autECLXfer was set. This type may change over time. ConnType is a String data type and is read-only. The following example shows this property.

```
DIM Type as String
DIM Obj as Object
Set Obj = CreateObject("PCOMM.autECLXfer")

' Initialize the connection
Obj.SetConnectionByName("A")
' Save the type
Type = Obj.ConnType
```

Connection types for the ConnType property are:

<table>
<thead>
<tr>
<th>String Returned</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISP3270</td>
<td>3270 display</td>
</tr>
<tr>
<td>DISP5250</td>
<td>5250 display</td>
</tr>
<tr>
<td>PRNT3270</td>
<td>3270 printer</td>
</tr>
<tr>
<td>PRNT5250</td>
<td>5250 printer</td>
</tr>
<tr>
<td>ASCII</td>
<td>VT emulation</td>
</tr>
</tbody>
</table>

**CodePage**
This is the code page of the connection for which autECLXfer was set. This code page may change over time. CodePage is a Long data type and is read-only. The following example shows this property.

```
DIM CodePage as Long
DIM Obj as Object
Set Obj = CreateObject("PCOMM.autECLXfer")

' Initialize the connection
Obj.SetConnectionByName("A")
' Save the code page
CodePage = Obj.CodePage
```

**Started**
This indicates whether the emulator window is started. The value is True if the window is open; otherwise, it is False. Started is a Boolean data type and is read-only. The following example shows this property.

```
DIM Hand as Long
DIM Obj as Object
Set Obj = CreateObject("PCOMM.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**
This indicates the status of the connection to the host. The value is True if the host is connected; otherwise, it is False. CommStarted is a Boolean data type and is read-only. The following example shows this property.

```vba
DIM Hand as Long
DIM Obj as Object

Set Obj = CreateObject("PCOMM.autECLXfer")

' Initialize the connection
Obj.SetConnectionByIndex("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**
This indicates whether the emulator is API-enabled. A connection may be enabled or disabled depending on the state of its API settings (in a Personal Communications window, choose File -> API Settings). The value is True if the emulator is enabled; otherwise, it is False. APIEnabled is a Boolean data type and is read-only. The following example shows this property.

```vba
DIM Hand as Long
DIM Obj as Object

Set Obj = CreateObject("PCOMM.autECLXfer")

' Initialize the connection
Obj.SetConnectionByIndex("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**
This indicates whether the emulator window is started, API enabled, and connected. This property checks for all three properties. The value is True if the emulator is ready; otherwise, it is False. Ready is a Boolean data type and is read-only. The following example shows this property.

```vba
DIM Hand as Long
DIM Obj as Object

Set Obj = CreateObject("PCOMM.autECLXfer")

' Initialize the connection
Obj.SetConnectionByIndex("A")

' This code segment checks to see if a is ready.
autECLXfer

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 Methods**

The following section describes the methods that are valid for the autECLXfer object.

```plaintext
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**

This method registers an object to receive notification of all communication link connect/disconnect events.

**Prototype**

```plaintext
void RegisterCommEvent()
```

**Parameters**

None

**Return Value**

None

**Example**

See “Event Processing Example” on page 302 for an example.

**UnregisterCommEvent**

Ends Communications Link Event Processing.

**Prototype**

```plaintext
void UnregisterCommEvent()
```

**Parameters**

None

**Return Value**

None

**SetConnectionByName**

The SetConnectionByName method uses the connection name to set the connection for a newly created autECLXfer object. In Personal Communications this connection name is the short ID (character A-Z). There can be only one Personal Communications connection open with a given name. For example, there can be only one connection “A” open at a time.
**Note:** Do not call this if using the autECLXfer object in autECLSession.

**Prototype**

```plaintext
tvoid SetConnectionByName( String Name )
```

**Parameters**

- **String Name**
  
  One-character string short name of the connection (A-Z).

**Return Value**

None

**Example**

The following example shows how to use the connection name to set the connection for a newly created autECLXfer object.

```plaintext
DIM XferObj as Object

Set XferObj = CreateObject("PCOMM.autECLXfer")

' Initialize the connection
XferObj.SetConnectionByName("A")
```

**SetConnectionByHandle**

The SetConnectionByHandle method uses the connection handle to set the connection for a newly created autECLXfer object. In Personal Communications this connection handle is a Long integer. There can be only one Personal Communications connection open with a given handle. For example, there can be only one connection “A” open at a time.

**Note:** Do not call this if using the autECLXfer object in autECLSession.

**Prototype**

```plaintext
tvoid SetConnectionByHandle( Long Handle )
```

**Parameters**

- **Long Handle**
  
  Long integer value of the connection to be set for the object.

**Return Value**

None

**Example**

The following example shows how to use the connection handle to set the connection for a newly created autECLXfer object.

```plaintext
DIM XferObj as Object
DIM autECLConnList as Object

Set XferObj = CreateObject("PCOMM.autECLXfer")
Set autECLConnList = CreateObject("PCOMM.autECLConnList")

' Initialize the connection with the first connection in the list
autECLConnList.Refresh
XferObj.SetConnectionByHandle(autECLConnList(1).Handle)
```

**SendFile**

The SendFile method sends a file from the workstation to the host for the connection associated with the autECLXfer object.
Prototype
void SendFile( String PCFile, String HostFile, String Options )

Parameters
String PCFile Name of the file on the workstation.
String HostFile Name of the file on the host.
String Options Host-dependent transfer options. See "Usage Notes" for more information.

Return Value
None

Usage Notes
File transfer options are host-dependent. The following is a list of some of the valid host options for a VM/CMS host.

- ASCII
- CRLF
- APPEND
- LRECL
- RECFM
- CLEAR/NOCLEAR
- PROGRESS
- QUIET

Refer to Emulator Programming for the list of supported hosts and associated file transfer options.

Example
The following example shows how to send a file from the workstation to the host for the connection associated with the autECLXfer object.

```vba
DIM XferObj as Object
DIM autECLConnList as Object
DIM NumRows as Long

Set XferObj = CreateObject("PCOMM.autECLXfer")
Set autECLConnList = CreateObject("PCOMM.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

The ReceiveFile method receives a file from the host to the workstation for the connection associated with the autECLXfer object.

Prototype
void ReceiveFile( String PCFile, String HostFile, String Options )

Parameters
String PCFile Name of the file on the workstation.
String HostFile Name of the file on the host.
String Options

Host-dependent transfer options. See “Usage Notes” for more information.

Return Value

None

Usage Notes

File transfer options are host-dependent. The following is a list of some of the valid host options for a VM/CMS host:

- ASCII
- CRLF
- APPEND
- LRECL
- RECFM
- CLEAR/NOCLEAR
- PROGRESS
- QUIET

Refer to Emulator Programming manual for the list of supported hosts and associated file transfer options.

Example

The following example shows how to receive a file from the host and send it to the workstation for the connection associated with the autECLXfer object.

```vbscript
DIM XferObj as Object
DIM autECLConnList as Object
DIM NumRows as Long

Set XferObj = CreateObject("PCOMM.autECLXfer")
Set autECLConnList = CreateObject("PCOMM.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

The StartCommunication collection element method connects the PCOMM emulator to the host data stream. This has the same effect as going to the PCOMM emulator Communication menu and choosing Connect.

Prototype

```vbscript
void StartCommunication()
```

Parameters

None

Return Value

None

Example

The following example shows how to connect a PCOMM emulator session to the host.
Dim XObj as Object
Dim autECLConnList as Object

Set autECLConnList = CreateObject("PCOMM.autECLConnList")
Set XObj = CreateObject("PCOMM.autECLXfer")

' Initialize the session
autECLConnList.Refresh
XObj.SetConnectionByHandle(autECLConnList(1).Handle)
XObj.StartCommunication()

StopCommunication

The StopCommunication collection element method disconnects the PCOMM emulator to the host data stream. This has the same effect as going to the PCOMM emulator Communication menu and choosing Disconnect.

Prototype
void StopCommunication()

Parameters
None

Return Value
None

Example
The following example shows how to connect a PCOMM emulator session to the host.

Dim XObj as Object
Dim autECLConnList as Object

Set autECLConnList = CreateObject("PCOMM.autECLConnList")
Set XObj = CreateObject("PCOMM.autECLXfer")

' Initialize the session
autECLConnList.Refresh
XObj.SetConnectionByHandle(autECLConnList(1).Handle)

SessObj.StopCommunication()

autECLXfer Events

The following events are valid for autECLXfer:

void NotifyCommEvent(boolean bConnected)
NotifyCommError()
void NotifyCommStop(Long Reason)

NotifyCommEvent

A given communications link as been connected or disconnected.

Prototype
void NotifyCommEvent(boolean bConnected)

Parameters

boolean bConnected True if Communications Link is currently Connected, False otherwise.
Example
See "Event Processing Example" for an example.

NotifyCommError
This event occurs when an error occurs in event processing.

Prototype
NotifyCommError()

Parameters
None

Example
See "Event Processing Example" for an example.

NotifyCommStop
This event occurs when event processing stops.

Prototype
void NotifyCommStop(Long Reason)

Parameters
Long Reason Reason code for the stop. Currently this will always be 0.

Event Processing Example
The following is a short example of how to implement Xfer Events
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 Class

The autSystem class is used to perform utility operations that are not present in some programming languages.

autSystem Methods

The following section describes the methods that are valid for the autSystem object.

Long Shell(VARIANT ExeName, VARIANT Parameters, VARIANT WindowStyle)
String Inputnd()

Shell

The shell function runs any executable file.

Prototype
Long Shell(VARIANT ExeName, VARIANT Parameters, VARIANT WindowStyle)

Parameters
VARIANT ExeName Full path and file name of the executable file.
VARIANT Parameters Any parameters to pass to the executable file. This parameter is optional.
VARIANT WindowStyle The initial window style to show as executable. This parameter is optional and can have the following values:
1. Normal with focus (default)
2. Minimized with focus
3. Maximized
4. Normal without focus
5. Minimized without focus

Return Value
The method returns the Process ID if it is successful, or zero if it fails.

Example
Example autSystem - Shell()

'This example starts notepad with the file c:\test.txt loaded
dim ProcessID
dim SysObj as object

set SysObj = CreateObject("PCOMM.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

The Inputnd method displays a popup input box to the user with a no-display text box so that when the user types in data only asterisks(*) are displayed.
**autSystem**

**Prototype**
String Inputnd()

**Parameters**
None

**Return Value**
The characters typed into the input box, or "" if nothing was typed in.

**Example**

```vbnet
DIM strPassWord
dim SysObj as Object
dim PSObj as Object

set SysObj = CreateObject("PCOMM.autSystem")
set PSObj = CreateObject("PCOMM.autPS")

PSObj.SetConnectionByName("A")
' Prompt user for password
strPassWord = SysObj.Inputnd()
PSObj.SetText(strPassWord)
DIM XferObj as Object

Set XferObj = CreateObject("PCOMM.autECLXfer")
' Initialize the connection
XferObj.SetConnectionByName("A")
```

**autECLPageSettings Class**

The autECLPageSettings object controls the page settings of a Personal Communications connection. Its name in the registry is Personal Communications.autECLPageSettings. This automation object can also be used in VB scripts.

The read-only property autECLPageSettings has been added to the autECLSession object. See "[autECLSession Class](#) on page 271" for information about how to use this property.

**Note:** The autECLPageSettings object in the autECLSession object is set by the autECLSession object.

The following example shows how to create and set the autECLPageSettings object in Visual Basic.

```vbnet
DIM PgSet as Object
Set PgSet = CreateObject("PCOMM.autECLPageSettings")
PgSet.SetConnectionByName("A")
```

**Usage Notes**

This object is not supported for DBCS or bidirectional sessions.

You must initially set the connection for the object you create. Use SetConnectionByName or SetConnectionByHandle to initialize your object. The connection can be set only once. After the connection is set, any further calls to the set connection methods cause an exception. If you do not set the connection and try to access a property or method, an exception is raised.
The properties CPI, LPI, and FontSize are dependent on the property FaceName. Therefore, if CPI, LPI, or FontSize are set before the FaceName is set, and if they are not valid for the new FaceName, different CPI, LPI, or FontSize values might be reconfigured in the connection. You should set the FaceName before setting the CPI, LPI, or FontSize. Otherwise, every time you set FaceName, query CPI, LPI, and FontSize and make sure that they have the desired values.

Restrictions

The connection associated with each method must be in a particular state for the method to succeed. If the restrictions are not met, an appropriate exception is raised.

The following restrictions must be satisfied while any property or method of the autECLPageSettings object is invoked.

- The host session should not be printing when this API is invoked.
- The File → Page Setup and File → Printer Setup dialogs must not be in use.
- The associated connection must not be in PDT mode.

Additional restrictions might apply for each specific property or method.

Connection types

The following connection types are valid for the methods in the autECLPageSettings class:

- 3270 display
- 3270 printer
- 5250 display
- VT (ASCII)

If a property or method is accessed or called on an unsupported connection, an exception is raised. Use the ConnType property to determine the connection type.

Properties

This section describes the properties for the autECLPageSettings object.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long</td>
<td>CPI</td>
<td>Read/Write</td>
</tr>
<tr>
<td>Boolean</td>
<td>FontCPI</td>
<td>Read-only</td>
</tr>
<tr>
<td>Long</td>
<td>LPI</td>
<td>Read/Write</td>
</tr>
<tr>
<td>Boolean</td>
<td>FontLPI</td>
<td>Read-only</td>
</tr>
<tr>
<td>String</td>
<td>FaceName</td>
<td>Read/Write</td>
</tr>
<tr>
<td>Long</td>
<td>FontSize</td>
<td>Read/Write</td>
</tr>
<tr>
<td>Long</td>
<td>MaxLinesPerPage</td>
<td>Read/Write</td>
</tr>
<tr>
<td>Long</td>
<td>MaxCharsPerLine</td>
<td>Read/Write</td>
</tr>
<tr>
<td>String</td>
<td>Name</td>
<td>Read-only</td>
</tr>
<tr>
<td>Long</td>
<td>Handle</td>
<td>Read-only</td>
</tr>
<tr>
<td>String</td>
<td>ConnType</td>
<td>Read-only</td>
</tr>
<tr>
<td>Long</td>
<td>CodePage</td>
<td>Read-only</td>
</tr>
<tr>
<td>Boolean</td>
<td>Started</td>
<td>Read-only</td>
</tr>
<tr>
<td>Type</td>
<td>Name</td>
<td>Attributes</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------</td>
<td>--------------</td>
</tr>
<tr>
<td>Boolean</td>
<td>CommStarted</td>
<td>Read-only</td>
</tr>
<tr>
<td>Boolean</td>
<td>APIEnabled</td>
<td>Read-only</td>
</tr>
<tr>
<td>Boolean</td>
<td>Ready</td>
<td>Read-only</td>
</tr>
</tbody>
</table>

**CPI**
This property determines the number of characters printed per inch. This is a Long data type and is read/write enabled.

Set this property to the predefined constant pcFontCPI to select the Font CPI in Page Settings or set it to some specific CPI value. If this property is queried when FontCPI is configured in the connection, the actual CPI value is returned and the constant pcFontCPI is not returned.

To determine whether FontCPI is set in the connection, use the property FontCPI.

**Example:**
```vba
Dim PgSet as Object
Dim ConnList as Object
Dim CPI as Long

Set PgSet = CreateObject("PCOMM.autECLPageSettings")
Set ConnList = CreateObject("PCOMM.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**
This determines whether Font CPI is set in the connection. FontCPI is a Boolean data type and is read-only.

**Example:**
```vba
Dim PgSet as Object
Dim ConnList as Object

Set PgSet = CreateObject("PCOMM.autECLPageSettings")
Set ConnList = CreateObject("PCOMM.autECLConnList")
' Initialize the connection
ConnList.Refresh
PgSet.SetConnectionByHandle(ConnList(1).Handle)

'check if Font CPI is set
If PgSet.FontCPI Then
  ...
```

**LPI**
This property determines the number of lines printed per inch. This is a Long data type and is read/write enabled. Set it to the predefined constant pcFontLPI to select the Font LPI in Page Settings or set it to some specific LPI value. If this property is queried when FontLPI is configured in the connection, the actual LPI value is returned and the constant pcFontLPI is not returned. To determine whether FontLPI is set in the connection, use the property FontLPI.

**Example:**
Dim PgSet as Object
Dim ConnList as Object
Dim LPI as Long

Set PgSet = CreateObject("PCOMM.autECLPageSettings")
Set ConnList = CreateObject("PCOMM.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**
This property determines whether Font LPI is set in the connection. FontLPI is a Boolean data type and is read-only.

**Example:**
Dim PgSet as Object
Dim ConnList as Object

Set PgSet = CreateObject("PCOMM.autECLPageSettings")
Set ConnList = CreateObject("PCOMM.autECLConnList")
' Initialize the connection
ConnList.Refresh
PgSet.SetConnectionByHandle(ConnList(1).Handle)

'check if Font LPI is set
If PgSet.FontLPI Then
...

**FaceName**
This is the Font Face Name of the Page Settings of the connection. FaceName is a String data type and is read/write enabled.

**Example:**
Dim PgSet as Object
Dim ConnList as Object
Dim FaceName as String

Set PgSet = CreateObject("PCOMM.autECLPageSettings")
Set ConnList = CreateObject("PCOMM.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

**FontSize**
This is the size of the font in the Page Settings of the connection. FontSize is a Long data type and is read/write enabled.

**Notes:**
1. This property is for sessions that use a DBCS code page. Connections that use an SBCS host code page will raise an exception.
2. Because FontSize is dependent on the FaceName of the connection, FaceName should be set before FontSize is set.

**Example:**
Dim PgSet as Object
Dim ConnList as Object
Dim FontSize as Long
Set PgSet = CreateObject("PCOMM.autECLPageSettings")
Set ConnList = CreateObject("PCOMM.autECLConnList")
' Initialize the connection
ConnList.Refresh
PgSet.SetConnectionByHandle(ConnList(1).Handle)
FontSize = PgSet.FontSize ' get the FontSize
' or...
PgSet.FontSize = 14 ' set the FontSize

**MaxLinesPerPage**
This property is the maximum number of lines that can be printed per page. This is also called *maximum print lines* or MPL. Valid values are in the range 1–255. This is a Long data type and is read/write enabled.

**Example:**
Dim PgSet as Object
Dim ConnList as Object
Dim MPL as Long
Set PgSet = CreateObject("PCOMM.autECLPageSettings")
Set ConnList = CreateObject("PCOMM.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**
This property is the maximum number of characters that can be printed per line. This is also called *maximum print position* or MPP. Valid values are in the range 1–255. This is a Long data type and is read/write enabled.

**Example:**
Dim PgSet as Object
Dim ConnList as Object
Dim MPP as Long
Set PgSet = CreateObject("PCOMM.autECLPageSettings")
Set ConnList = CreateObject("PCOMM.autECLConnList")
' Initialize the connection
ConnList.Refresh
PgSet.SetConnectionByHandle(ConnList(1).Handle)
MPP = PgSet.MaxCharsPerLine ' get the MaxCharsPerLine
' or...
PgSet.MaxCharsPerLine = 80 ' set the MaxCharsPerLine

**Name**
This property is the connection name string of the connection for which autECLPageSettings was set. Personal Communications returns only the short character ID (a single alphabetical character from A to Z) in the string. There can be only one Personal Communications connection open with a given name. For example, there can be only one connection A open at a time. Name is a String data type and is read-only.

**Example:**
Dim PgSet as Object
Dim ConnList as Object
DIM Name as String

Set PgSet = CreateObject("PCOMM.autECLPageSettings")
Set ConnList = CreateObject("PCOMM.autECLConnList")
' Initialize the connection
ConnList.Refresh
PgSet.SetConnectionByHandle(ConnList(1).Handle)

Name = PgSet.Name ' Save the name

**Handle**

This property is the handle of the connection for which the autECLPageSettings object was set. There can be only one Personal Communications connection open with a given handle. For example, there can be only one connection A open at a time. Handle is a Long data type and is read-only.

**Example:**

Dim PgSet as Object
Dim ConnList as Object
Dim Hand as Long

Set PgSet = CreateObject("PCOMM.autECLPageSettings")
Set ConnList = CreateObject("PCOMM.autECLConnList")
' Initialize the connection
ConnList.Refresh
PgSet.SetConnectionByHandle(ConnList(1).Handle)

Hand = PgSet.Handle ' Save the handle

**ConnType**

This property is the connection type for which autECLPageSettings was set. This type might change over time. ConnType is a String data type and is read-only.

<table>
<thead>
<tr>
<th>String Value</th>
<th>Connection Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISP3270</td>
<td>3270 display</td>
</tr>
<tr>
<td>DISP5250</td>
<td>5250 display</td>
</tr>
<tr>
<td>PRNT3270</td>
<td>3270 printer</td>
</tr>
<tr>
<td>PRNT5250</td>
<td>5250 printer</td>
</tr>
<tr>
<td>ASCII</td>
<td>VT emulation</td>
</tr>
</tbody>
</table>

**Example:**

Dim PgSet as Object
Dim ConnList as Object
Dim Type as String

Set PgSet = CreateObject("PCOMM.autECLPageSettings")
Set ConnList = CreateObject("PCOMM.autECLConnList")
' Initialize the connection
ConnList.Refresh
PgSet.SetConnectionByHandle(ConnList(1).Handle)

Type = PgSet.ConnType ' Save the type

**CodePage**

This property is the connection type for which autECLPageSettings was set. This type might change over time. ConnType is a String data type and is read-only.
autECLPageSettings

Example:
Dim PgSet as Object
Dim ConnList as Object
Dim CodePage as Long

Set PgSet = CreateObject("PCOMM.autECLPageSettings")
Set ConnList = CreateObject("PCOMM.autECLConnList")
' Initialize the connection
ConnList.Refresh
PgSet.SetConnectionByHandle(ConnList(1).Handle)

CodePage = PgSet.CodePage ' save the codepage

Started
This property indicates whether the emulator window is started. The value is TRUE if the window is open; otherwise, it is FALSE. Started is a Boolean data type and is read-only.

Example:
Dim PgSet as Object
Dim ConnList as Object

Set PgSet = CreateObject("PCOMM.autECLPageSettings")
Set ConnList = CreateObject("PCOMM.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
This property indicates the status of the connection to the host. The value is TRUE if the host is connected; otherwise, it is FALSE. CommStarted is a Boolean data type and is read-only.

Example:
Dim PgSet as Object
Dim ConnList as Object

Set PgSet = CreateObject("PCOMM.autECLPageSettings")
Set ConnList = CreateObject("PCOMM.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
This property indicates whether the emulator is API-enabled. A connection can be API-enabled or disabled depending on the state of its API settings (in a Personal
Communications window, click **Edit → Preferences → API**. The value is TRUE if the emulator is API-enabled; otherwise, it is FALSE. APIEnabled is a Boolean data type and is read-only.

**Example:**

```vba
Dim PgSet as Object
Dim ConnList as Object

Set PgSet = CreateObject("PCOMM.autECLPageSettings")
Set ConnList = CreateObject("PCOMM.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**

This property indicates whether the emulator window is started, API-enabled, and connected. This property checks for all three properties. The value is TRUE if the emulator is ready; otherwise, it is FALSE. Ready is a Boolean data type and is read-only.

**Example:**

```vba
Dim PgSet as Object
Dim ConnList as Object

Set PgSet = CreateObject("PCOMM.autECLPageSettings")
Set ConnList = CreateObject("PCOMM.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 Methods**

The following section describes the methods that are valid for the autECLPageSettings object.

- `void RestoreTextDefaults()`
- `void SetConnectionByName (String Name)`
- `void SetConnectionByHandle (Long Handle)`

**RestoreTextDefaults**

The RestoreTextDefaults method restores the system default values of the **Text** property page in the Page Setup dialog of the connection. This is equivalent to pressing the **Default** button on the **Text** property page of the Page Setup Dialog of the connection.
autECLPageSettings

**Prototype**
void RestoreTextDefaults()

**Parameters**
None

**Return Value**
None

**Example**
The following example shows the RestoreTextDefaults method.
```
    Dim PgSet as Object
    Dim ConnList as Object

    Set PgSet = CreateObject("PCOMM.autECLPageSettings")
    Set ConnList = CreateObject("PCOMM.autECLConnList")
    ' Initialize the connection
    ConnList.Refresh
    PgSet.SetConnectionByHandle(ConnList(1).Handle)

    PgSet.RestoreTextDefaults 'Restores Text Default Settings
```

**SetConnectionByName**
The SetConnectionByName method uses the connection name to set the connection for a newly created autECLPageSettings object. This connection name is the short connection ID (a single alphabetical character from A to Z). There can be only one Personal Communications connection open with a given name. For example, there can be only one connection A open at a time.

**Note:** Do not call this method if you are using the autECLPageSettings object contained in the autECLSession object.

**Prototype**
void SetConnectionByName( String Name )

**Parameters**
String Name
   One-character string short name of the connection. Valid values are A–Z.

**Return Value**
None

**Example**
The following example shows how to use the connection name to set the connection for a newly created autECLPageSettings object.
```
    Dim PgSet as Object
    Set PgSet = CreateObject("PCOMM.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**
The SetConnectionByHandle method uses the connection handle to set the connection for a newly created autECLPageSettings object. In Personal
Communications, this connection handle is a Long integer. There can be only one Personal Communications connection open with a given handle. For example, there can be only one connection A open at a time.

**Note:** Do not call this method if you are using the autECLPageSettings object contained in the autECLSession object.

**Prototype**

```c
void SetConnectionByHandle( Long Handle )
```

**Parameters**

- `Long Handle`
  - Long integer value of the connection to be set for the object.

**Return Value**

None

**Example**

The following example shows how to use the connection handle to set the connection for a newly created autECLPageSettings object.

```vbnet
Dim PgSet as Object
Dim ConnList as Object

Set PgSet = CreateObject("PCOMM.autECLPageSettings")
Set ConnList = CreateObject("PCOMM.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 Class

The autECLPrinterSettings object controls the Printer Settings of a Personal Communications connection. Its name in the registry is `PCOMM.autECLPrinterSettings`. This automation object can also be used in VB scripts.

The read-only property `autECLPrinterSettings` has been added to the autECLSession object. See ["autECLSession Class" on page 271](#) for information about how to use this property.

**Note:** The autECLPrinterSettings object in the autECLSession object is set by the autECLSession object.

The following example shows how to create and set the autECLPrinterSettings object in Visual Basic.

```vbnet
DIM PrSet as Object
Set PrSet = CreateObject("PCOMM.autECLPrinterSettings")
PrSet.SetConnectionByName("A")
```

**Usage Notes**

This object is not supported for DBCS or bidirectional sessions.
You must initially set the connection for the object you create. Use SetConnectionByName or SetConnectionByHandle to initialize your object. The connection can be set only once. After the connection is set, any further calls to the set connection methods cause an exception. If you do not set the connection and try to access a property or method, an exception is raised.

The properties CPI, LPI, and FontSize are dependent on the property FaceName. Therefore, if CPI, LPI, or FontSize are set before the FaceName is set, and if they are not valid for the new FaceName, different CPI, LPI, or FontSize values might be reconfigured in the connection. You should set the FaceName before setting the CPI, LPI, or FontSize. Otherwise, every time you set FaceName, query CPI, LPI, and FontSize and make sure that they have the desired values.

Restrictions

The connection associated with each method must be in a particular state for the method to succeed. If the restrictions are not met, an appropriate exception is raised.

The following restrictions must be satisfied while any property or method of the autECLPageSettings object is invoked.
- The host session should not be printing when this API is invoked.
- The File → Page Setup and File → Printer Setup dialogs should not be in use.

Additional restrictions might apply for each specific property or method.

Properties

This section describes the properties for the autECLPrinterSettings object.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>PDTMode</td>
<td>Read-only</td>
</tr>
<tr>
<td>String</td>
<td>PDTFile</td>
<td>Read-only</td>
</tr>
<tr>
<td>Long</td>
<td>PrintMode</td>
<td>Read-only</td>
</tr>
<tr>
<td>String</td>
<td>Printer</td>
<td>Read-only</td>
</tr>
<tr>
<td>String</td>
<td>PrtToDskAppendFile</td>
<td>Read-only</td>
</tr>
<tr>
<td>String</td>
<td>PrtToDskSeparateFile</td>
<td>Read-only</td>
</tr>
<tr>
<td>Boolean</td>
<td>PrompDialogOption</td>
<td>Read/Write</td>
</tr>
<tr>
<td>String</td>
<td>Name</td>
<td>Read-only</td>
</tr>
<tr>
<td>Long</td>
<td>Handle</td>
<td>Read-only</td>
</tr>
<tr>
<td>String</td>
<td>ConnType</td>
<td>Read-only</td>
</tr>
<tr>
<td>Boolean</td>
<td>CodePage</td>
<td>Read-only</td>
</tr>
<tr>
<td>Boolean</td>
<td>Started</td>
<td>Read-only</td>
</tr>
<tr>
<td>Boolean</td>
<td>CommStarted</td>
<td>Read-only</td>
</tr>
<tr>
<td>Boolean</td>
<td>APIEnabled</td>
<td>Read-only</td>
</tr>
<tr>
<td>Boolean</td>
<td>Ready</td>
<td>Read-only</td>
</tr>
</tbody>
</table>

PDTMode

This property determines whether the connection is in PDT mode or not. PDTMode is a Boolean data type and is read/write enabled.
Example:
Dim PrSet as Object
Dim ConnList as Object

Set PrSet = CreateObject("PCOMM.autECLPrinterSettings")
Set ConnList = CreateObject("PCOMM.autECLConnList")
' Initialize the connection
ConnList.Refresh
PrSet.SetConnectionByHandle(ConnList(1).Handle)

' check if in PDT mode.
If PrSet.PDTMode Then
    ...
Else
    ...

**PDTFile**
This property is the PDT file configured in the connection. This property gives a null string if the PDT file is not configured in the connection. Otherwise, this property gives the fully qualified path name of the PDT file. PDTFile is a String data type and is read-only.

Example:
Dim PrSet as Object
Dim ConnList as Object

Set PrSet = CreateObject("PCOMM.autECLPrinterSettings")
Set ConnList = CreateObject("PCOMM.autECLConnList")
' Initialize the connection
ConnList.Refresh
PrSet.SetConnectionByHandle(ConnList(1).Handle)

If PrSet.PDTFile = vbNullString Then ' get the
    ...
Else
    ...

**PrintMode**
This property indicates the print mode of the connection. PrintMode is a Long data type and is read-only. This property returns one of the following four enumerated values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Name of the enum constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>pcPrtToDskAppend</td>
<td>Print to Disk-Append mode. This means the Print to Disk + Append option is selected in the Printer listbox in the Printer Setup dialog of the connection.</td>
</tr>
<tr>
<td>2</td>
<td>pcPrtToDskSeparate</td>
<td>Print to Disk-Separate mode. This means the Print to Disk + Separate option is selected in the Printer listbox in the Printer Setup dialog of the connection.</td>
</tr>
<tr>
<td>3</td>
<td>pcSpecificPrinter</td>
<td>Specific Printer mode. This means one of the printers is selected in the Printer listbox in the Printer Setup dialog of the connection and the Use Windows Default Printer checkbox is clear.</td>
</tr>
<tr>
<td>4</td>
<td>pcWinDefaultPrinter</td>
<td>Windows Default Printer mode. This means that the Use Windows Default Printer checkbox is selected.</td>
</tr>
</tbody>
</table>

Example:
Dim PrSet as Object
Dim ConnList as Object
```vba
Set PrSet = CreateObject("PCOMM.autECLPrinterSettings")
Set ConnList = CreateObject("PCOMM.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
...

Printer
This property is the name of the printer. It contains one of the following:
• The name of the specific printer if the PrintMode of the connection is pcSpecificPrinter.
• The name of the Windows default printer if the PrintMode of the connection is pcWinDefaultPrinter.
• A null string if a printer is not configured in the connection or if the PrintMode of the connection is pcPrtToDskAppend or pcPrtToDskSeparate.

Printer is a String data type and is read-only.

The value must have the following format:
<Printer name> on <Port Name>

For example:
• IBM InfoPrint 40 PS on Network Port
• HP LaserJet 4050 Series PCL 6 on LPT1

Example:
Dim PrSet as Object
Dim ConnList as Object
Dim Printer as String

Set PrSet = CreateObject("PCOMM.autECLPrinterSettings")
Set ConnList = CreateObject("PCOMM.autECLConnList")
' Initialize the connection
ConnList.Refresh
PrSet.SetConnectionByHandle(ConnList(1).Handle)

Printer = PrSet.Printer ' get the Printer Name

PrtToDskAppendFile
This property is the name of the file set for the Print to Disk-Append mode. This file is called the Print to Disk-Append file. This property contains one of the following:
• The fully qualified path name of the Print to Disk-Append file of the connection.
• A null string if the Print to Disk-Append file is not configured in the connection.

PrtToDskAppendFile is a String data type and is read-only.

Example:
Dim PrSet as Object
Dim ConnList as Object
Dim DskAppFile as String

Set PrSet = CreateObject("PCOMM.autECLPrinterSettings")
Set ConnList = CreateObject("PCOMM.autECLConnList")
' Initialize the connection
ConnList.Refresh
PrSet.SetConnectionByHandle(ConnList(1).Handle)

DskAppFile = PrSet.PrtToDskAppendFile ' get the Disk append file.

**PrtToDskSeparateFile**

This property is the name of the file set for the **Print to Disk-Separate** mode. This file is called the **Print to Disk-Separate** file. This property contains one of the following:

- The fully qualified path name of the **Print to Disk-Separate** file of the connection.
- A null string if the **Print to Disk-Separate** file is not configured in the connection.

PrtToDskSeparateFile is a String data type and is read-only.

**Example:**

Dim PrSet as Object
Dim ConnList as Object
Dim DskSepFile as String

Set PrSet = CreateObject("PCOMM.autECLPrinterSettings")
Set ConnList = CreateObject("PCOMM.autECLConnList")
' Initialize the connection
ConnList.Refresh
PrSet.SetConnectionByHandle(ConnList(1).Handle)

DskSepFile = PrSet.PrtToDskSeparateFile ' get the Disk separate file.

**PromptDialogOption**

This property indicates whether the option to show the Printer Setup dialog before printing is set or not. PromptDialogOption is a Boolean data type and is read-only.

**Example:**

Dim PrSet as Object
Dim ConnList as Object
Dim PromptDialog as Boolean

Set PrSet = CreateObject("PCOMM.autECLPrinterSettings")
Set ConnList = CreateObject("PCOMM.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

**Name**

This property is the connection name string of the connection for which autECLPrinterSettings was set. Personal Communications returns only the short character ID (a single alphabetical character from A to Z) in the string. There can
be only one Personal Communications connection open with a given name. For example, there can be only one connection open at a time. Name is a String data type and is read-only.

Example:
Dim PrSet as Object
Dim ConnList as Object
DIM Name as String
Set PrSet = CreateObject("PCOMM.autECLPrinterSettings")
Set ConnList = CreateObject("PCOMM.autECLConnList")
' Initialize the connection
ConnList.Refresh
PrSet.SetConnectionByHandle(ConnList(1).Handle)
Name = PrSet.Name ' Save the name

Handle
This property is the handle of the connection for which the autECLPrinterSettings object was set. There can be only one Personal Communications connection open with a given handle. For example, there can be only one connection open at a time. Handle is a Long data type and is read-only.

Example:
Dim PrSet as Object
Dim ConnList as Object
Dim Hand as Long
Set PrSet = CreateObject("PCOMM.autECLPrinterSettings")
Set ConnList = CreateObject("PCOMM.autECLConnList")
' Initialize the connection
ConnList.Refresh
PrSet.SetConnectionByHandle(ConnList(1).Handle)
Hand = PrSet.Handle ' Save the handle

ConnType
This property is the connection type for which autECLPrinterSettings was set. This type might change over time. ConnType is a String data type and is read-only.

<table>
<thead>
<tr>
<th>String Value</th>
<th>Connection Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISP3270</td>
<td>3270 display</td>
</tr>
<tr>
<td>DISP5250</td>
<td>5250 display</td>
</tr>
<tr>
<td>PRNT3270</td>
<td>3270 printer</td>
</tr>
<tr>
<td>PRNT5250</td>
<td>5250 printer</td>
</tr>
<tr>
<td>ASCII</td>
<td>VT emulation</td>
</tr>
</tbody>
</table>

Example:
Dim PrSet as Object
Dim ConnList as Object
Dim Type as String
Set PrSet = CreateObject("PCOMM.autECLPrinterSettings")
Set ConnList = CreateObject("PCOMM.autECLConnList")
' Initialize the connection
ConnList.Refresh
PrSet.SetConnectionByHandle(ConnList(1).Handle)
Type = PrSet.ConnType ' Save the type
CodePage
This property is the code page of the connection for which autECLPrinterSettings was set. This code page might change over time. CodePage is a Long data type and is read-only.

Example:
Dim PrSet as Object
Dim ConnList as Object
Dim CodePage as Long

Set PrSet = CreateObject("PCOMM.autECLPrinterSettings")
Set ConnList = CreateObject("PCOMM.autECLConnList")
' Initialize the connection
ConnList.Refresh
PrSet.SetConnectionByHandle(ConnList(1).Handle)

CodePage = PrSet.CodePage ' save the codepage

Started
This property indicates whether the emulator window is started. The value is TRUE if the window is open; otherwise, it is FALSE. Started is a Boolean data type and is read-only.

Example:
Dim PrSet as Object
Dim ConnList as Object

Set PrSet = CreateObject("PCOMM.autECLPrinterSettings")
Set ConnList = CreateObject("PCOMM.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
This property indicates the status of the connection to the host. The value is TRUE if the host is connected; otherwise, it is FALSE. CommStarted is a Boolean data type and is read-only.

Example:
Dim PrSet as Object
Dim ConnList as Object

Set PrSet = CreateObject("PCOMM.autECLPrinterSettings")
Set ConnList = CreateObject("PCOMM.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
APIEnabled
This property indicates whether the emulator is API-enabled. A connection is API-enabled or disabled depending on the state of its API settings (in a Personal Communications window, click Edit → Preferences → API).

The value is TRUE if the emulator is API-enabled; otherwise, it is FALSE. APIEnabled is a Boolean data type and is read-only.

Example:
Dim PrSet as Object
Dim ConnList as Object

Set PrSet = CreateObject("PCOMM.autECLPrinterSettings")
Set ConnList = CreateObject("PCOMM.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
This property indicates whether the emulator window is started, API-enabled, and connected. This property checks for all three properties. The value is TRUE if the emulator is ready; otherwise, it is FALSE. Ready is a Boolean data type and is read-only.

Example:
Dim PrSet as Object
Dim ConnList as Object

Set PrSet = CreateObject("PCOMM.autECLPrinterSettings")
Set ConnList = CreateObject("PCOMM.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
The following sections describe the methods that are valid for the autECLPrinterSettings object.

```plaintext
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

The SetPDTMode method sets the connection in PDT mode with the given PDT file or sets the connection in non-PDT mode (also called GDI mode).

#### Restriction

If this method is called with bPDTMode set to FALSE, PrintMode of the associated connection must already be set to SpecificPrinter or WinDefaultPrinter.

#### Prototype

```plaintext
void SetPDTMode(Boolean bPDTMode, [optional] String PDTFile)
```

#### Parameters

**Boolean bPDTMode**  
Possible values are as follows:  
• TRUE to set the connection to PDT mode.  
• FALSE to set the connection to non-PDT mode (GDI mode).

**String PDTFile**  
This optional parameter contains the PDT file name.  
This parameter is used only if bPDTMode is TRUE. If this parameter is not specified and bPDTMode is set to TRUE, the PDT file configured in the connection is used. If there is no PDT file already configured in the connection, this method fails with an exception.  
This parameter ignored if bPDTMode is FALSE.  
Possible values are as follows:  
• File name without path
  
PDTFile in the PDFPDT subfolder in the Personal Communications installation path is used.  
• Fully qualified path name of the file  
If PDTFile does not exist, this method fails with an exception.

#### Return Value

None

#### Example

```plaintext
Dim PrSet as Object
Dim ConnList as Object

Set PrSet = CreateObject("PCOMM.autECLPrinterSettings")
Set ConnList = CreateObject("PCOMM.autECLConnList")
' Initialize the connection
```
SetPrtToDskAppend

This method sets the PrintMode of the connection to Print to Disk-Append mode. It also sets the appropriate file for this mode.

Notes:
1. The folder where this file is to be set must have write access. Otherwise, this method fails with an exception.
2. The associated connection must be in PDT mode.

Prototype
void SetPrtToDskAppend([optional] String FileName)

Parameters
String FileName
   This optional parameter contains the name of the Print to Disk-Append file.
   If the file exists, it is used. Otherwise, it is created when printing is complete.
   Possible values are as follows:
      • File name, without the path
         The user-class application data directory path will be used to locate the file.
      • Fully qualified path name of the file
         The directory must exist in the path, or the method will fail with an exception. It is not necessary that the file exist in the path.
   If this parameter is not specified, the file configured for this PrintMode in the connection is used. If there is no file already configured in the connection, this method fails with an exception.

Return Value
None

Example
Dim PrSet as Object
Dim ConnList as Object

Set PrSet = CreateObject("PCOMM.autECLPrinterSettings")
Set ConnList = CreateObject("PCOMM.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

This method sets the PrintMode of the connection to Print to Disk-Separate mode. It also sets the appropriate file for this mode.
Notes:
1. The folder where this file is to be set must have write access. Otherwise, this method fails with an exception.
2. The associated connection must be in PDT mode.

Prototype
void SetPrtToDskSeparate([optional] String FileName)

Parameters

String FileName
This optional parameter contains the name of the Print to Disk-Separate file.

If this parameter is not specified, the file configured for this PrintMode in the connection is used.

Possible values are:
• NULL (default)
  The file that is currently configured for this PrintMode in the connection is used. If there is no file already configured in the connection, the method fails with an exception.
• File name, without the path
  The user-class application data directory path will be used to locate the file.
• Fully qualified path name of the file
  The directory must exist in the path, or the method will fail with an exception. It is not necessary that the file exist in the path.

Note: The file name must not contain an extension. If it contains an extension, the method fails with an exception.

Return Value
None

Example
Dim PrSet as Object
Dim ConnList as Object

Set PrSet = CreateObject("PCOMM.autECLPrinterSettings")
Set ConnList = CreateObject("PCOMM.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

This method sets the PrintMode of the connection to Specific Printer mode with the printer specified by the Printer parameter.

Prototype
void SetSpecificPrinter(String Printer)
Parameters

String Printer
Contains the name of the printer. If the printer does not exist, this method fails with an exception.

The value must have the following format:
<Printer name> on <Port Name>

For example:
- IBM InfoPrint 40 PS on Network Port
- HP LaserJet 4050 Series PCL 6 on LPT1

Return Value
None

Example
Dim PrSet as Object
Dim ConnList as Object

Set PrSet = CreateObject("PCOMM.autECLPrinterSettings")
Set ConnList = CreateObject("PCOMM.autECLConnList")
' Initialize the connection
ConnList.Refresh
PrSet.SetConnectionByHandle(ConnList(1).Handle)

' Set PrintMode to pcSpecificPrinter
PrSet.SetSpecificPrinter("IBM InfoPrint 40 PS on Network Port")

SetWinDefaultPrinter
This method sets the PrintMode of the connection to Windows Default Printer mode (the connection will use the Windows default printer). If no Windows default printer is configured, this method fails with an exception.

Prototype
void SetWinDefaultPrinter()

Parameters
None

Return Value
None

Example
Dim PrSet as Object
Dim ConnList as Object

Set PrSet = CreateObject("PCOMM.autECLPrinterSettings")
Set ConnList = CreateObject("PCOMM.autECLConnList")
' Initialize the connection
ConnList.Refresh
PrSet.SetConnectionByHandle(ConnList(1).Handle)

' Set PrintMode to pcWinDefaultPrinter
PrSet.SetWinDefaultPrinter

SetConnectionByName
The SetConnectionByName method uses the connection name to set the connection for a newly created autECLPrinterSettings object. In Personal Communications, this
connection name is the short connection ID (a single alphabetical character from A to Z). There can be only one Personal Communications connection open with a given name. For example, there can be only one connection A open at a time.

**Note:** Do not call this if you are using the autECLPrinterSettings object contained in the autECLSession object.

### Prototype

```csharp
void SetConnectionByName( String Name )
```

### Parameters

- **String Name**
  - One-character string short name of the connection. Valid values are A–Z.

### Return Value

None

### Example

The following example shows how to use the connection name to set the connection for a newly created autECLPrinterSettings object.

```csharp
Dim PrSet as Object
Set PrSet = CreateObject("PCOMM.autECLPrinterSettings")
' Initialize the connection
PrSet.SetConnectionByName("A")
' For example, see if PDTMode
If PrSet.PDTMode Then
    'your logic here...
End If
```

### SetConnectionByHandle

The SetConnectionByHandle method uses the connection handle to set the connection for a newly created autECLPrinterSettings object. In Personal Communications, this connection handle is a Long integer.

There can be only one Personal Communications connection open with a given handle. For example, there can be only one connection A open at a time.

**Note:** Do not call this method if you are using the autECLPrinterSettings object contained in the autECLSession object.

### Prototype

```csharp
void SetConnectionByHandle( Long Handle )
```

### Parameters

- **Long Handle**
  - The Long integer value of the connection to set for the object.

### Return Value

None

### Example

The following example shows how to use the connection handle to set the connection for a newly created autECLPrinterSettings object.

```csharp
Dim PrSet as Object
Dim ConnList as Object
Set PrSet = CreateObject("PCOMM.autECLPrinterSettings")
```
autECLPrinterSettings

Set ConnList = CreateObject("PCOM.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

Support For Primary Interop Assemblies for Automation Objects

Automation objects exposed by IBM Personal Communications can be used by applications written in any language that targets the .NET framework. Managed .NET applications can program Personal Communications by using the Primary Interop Assemblies (PIA) that wrap the automation objects. Interop Assemblies are the mechanism with which managed (.NET) applications use COM-compliant objects. Interop Assemblies contain binding and metadata information, which enables the .NET framework (CLR) to load or marshall COM objects and wrap them for .NET applications. The PIA contains the official description of the COM types as defined by the publisher of those COM types. The PIA is always digitally signed by the publisher of the original COM type.

There are two ways a .NET application can reference an assembly.
- If it is a simple application or the only application that uses the assembly, Microsoft recommends that the assembly be copied in the same directory as the application.
- If multiple applications are referencing the assembly, you can install them in the Global Assembly Cache (GAC) and have all the solutions reference the assembly in the GAC.

The model for programming the types exposed by Interop Assemblies is very similar to COM. The methods, properties, and events exposed by the COM object can be accessed by any .NET language, using the syntax of the language. A sample application (ECLSamps.net) written in C# is provided in the \samples directory in the Personal Communications installation image. The sample demonstrates the simple usage of various Interop Assembly types.

For Visual Basic 6.0, projects that use Personal Communications automation objects and have been migrated to Visual Basic .NET using the conversion assistant wizard, you only need to replace the references that the conversion assistant wizard implicitly generates with the corresponding Personal Communications Interop references (from the \Interops directory) and recompile. The way to replace the references is to delete all the references generated by the conversion assistant and use Visual Studio .NET to add the .NET interop references. If you have registered the assemblies in the GAC and want to use them, add the references and set the Copy Local property for the Personal Communications Interop references to False.

The PIAs for the Personal Communications emulator automation objects are installed in the \Interops directory in the Personal Communications installation image. If the Personal Communications product installer detects that the .NET framework is present, it gives you the additional option to register the types in the GAC. While installing the assemblies in the GAC, the PIAs will also be placed in the registry, under the registry key of the corresponding type library.
Table 2 lists the PIAs supplied for the Personal Communications automation objects.

**Table 2. Primary Interop Assemblies for Personal Communications Automation Objects**

<table>
<thead>
<tr>
<th>Automation Object</th>
<th>Interop Assembly Dependency</th>
</tr>
</thead>
<tbody>
<tr>
<td>autECLConnList</td>
<td>Interop.AutConnListTypeLibrary.dll</td>
</tr>
<tr>
<td>autECLConnMgr</td>
<td>Interop.AutConnMgrTypeLibrary.dll</td>
</tr>
<tr>
<td>autECLConnList</td>
<td>Interop.AutPSTypeLibrary.dll</td>
</tr>
<tr>
<td>autECLOIA</td>
<td>Interop.AutOIATypeLibrary.dll</td>
</tr>
<tr>
<td>autECLPS</td>
<td>Interop.AutPSTypeLibrary.dll</td>
</tr>
<tr>
<td>autECLScreenDesc</td>
<td>Interop.AutScreenDescTypeLibrary.dll</td>
</tr>
<tr>
<td>autECLScreenReco</td>
<td>Interop.AutScreenRecoTypeLibrary.dll</td>
</tr>
<tr>
<td>autECLSession</td>
<td>Interop.AutSessTypeLibrary.dll</td>
</tr>
<tr>
<td>autECLPageSettings</td>
<td>Interop.AutSettingsTypeLibrary.dll</td>
</tr>
<tr>
<td>autECLPrinterSettings</td>
<td>Interop.AutSettingsTypeLibrary.dll</td>
</tr>
<tr>
<td>autECLWinMetrics</td>
<td>Interop.AutWinMetricsTypeLibrary.dll</td>
</tr>
<tr>
<td>autECLXfer</td>
<td>Interop.AutXferTypeLibrary.dll</td>
</tr>
<tr>
<td>autSystem</td>
<td>Interop.AutSystemTypeLibrary.dll</td>
</tr>
</tbody>
</table>
Primary Interop Assemblies
Chapter 4. Host Access Class Library LotusScript Extension

The Host Access Class Library LotusScript Extension (ECLLSX) allows you to write LotusScript programs that can query and control Personal Communications connections. The ECLLSX contains several new LotusScript classes that can be used inside LotusScript programs. By running methods on objects created from the new classes, you can access Personal Communications connection information and control the objects that make up a Personal Communications connection.

For example, if you want to automate the task of entering a line of text in a Personal Communications connection you can write a LotusScript program that uses the IsxECLPS class to create an lsxECLPS object associated with the presentation space of a Personal Communications connection. You can then run the SendKeys method on this lsxECLPS object to send a series of keystrokes to the presentation space and the effect is similar to a user typing the keystrokes in that presentation space. The following code fragment shows how this would be done using the ECLLSX classes:

```
'Create an lsxECLPS object associated with Personal
'Communications connection A
dim myPSobj as new lsxECLPS("A")

'Send some keystrokes to the presentation space of
'connection A
myPSobj.Sendkeys("[clear]QUERY FILES[ENTER]")
```

The ECLLSX classes are similar to the ECL C++ classes. Each ECLLSX class begins with lsxECL, for LotusScript Host Access Class Library. The classes are as follows:

- `IsxECLConnection`, Connection Information, on page 330 provides information about the Personal Communications connection associated with this lsxECLConnection object. In addition to being included in an lsxECLConnList object, an lsxECLConnection object can be created on its own if you only want to query information on a specific Personal Communications connection.
- `IsxECLConnList`, Connection List, on page 334 provides a list of Personal Communications connections on a system. Each element in an lsxECLConnList is an lsxECLConnection object.
- `IsxECLConnMgr`, Connection Manager, on page 337 manages Personal Communications connections on a system. Each lsxECLConnMgr object contains an lsxECLConnList object.
- `IsxECLField`, Field Information, on page 339 provides information on a field in the presentation space of the Personal Communications connection associated with this lsxECLField object.
- `IsxECLFieldList`, Field List, on page 344 provides a list of the fields in the presentation space of the Personal Communications connection associated with this lsxECLFieldList object. Each element in the list is an lsxECLField object.
- `IsxECLIOA`, Operator Information Area, on page 346 provides methods to query and manipulate the Operator Information Area of the associated Personal Communications connection. In addition to being contained in an lsxECLSession object, an lsxECLIOA object can be created on its own if you only want to perform OIA related tasks.
- `IsxECLPS`, Presentation Space, on page 355 provides methods to query and manipulate the Presentation Space of the associated Personal Communications connection. An lsxECLPS object contains an lsxECLFieldList object. In addition

© Copyright IBM Corp. 1997, 2004
to being contained in an lsxECLSession object, an lsxECLPS object can be created on its own if you only want to perform presentation space related tasks.

- lsxECLScreenDesc, Screen Description, on page 373 provides methods and properties to describe a screen. This may be used to wait for screens on the autECLPS object or the autECLScreenReco object.
- lsxECLScreenReco, Screen Recognition, on page 372 provides the engine of the HACL screen recognition system.
- lsxECLSession, Session, on page 378 provides Personal Communications connection related functionality and information. For convenience, an lsxECLSession object contains lsxECLPS, lsxECLXfer, lsxECLWinMetrics and lsxECLIOA objects for the Personal Communications connection associated with the lsxECLSession object.
- lsxECLWinMetrics, Window Metrics, on page 382 provides methods to query the window metrics of the Personal Communications connection associated with this lsxECLWinMetrics object. In addition to being contained in an lsxECLSession object, an lsxECLWinMetrics object can be created on its own if you only want to perform window metrics related queries.
- lsxECLXfer, File Transfer, on page 389 provides methods to transfer files between the host and the workstation over the Personal Communications connection associated with this file transfer object. In addition to being contained in an lsxECLSession object, an lsxECLXfer object can be created on its own if you only want to perform file transfer related tasks.

In order to use the ECL LotusScript Extension classes in a LotusScript program, you must load the ECL LotusScript Extension. This can be done using the following LotusScript statement:

USELX "*pcs1sx"

This statement loads the ECL LotusScript Extension and allows you to access the ECL LotusScript Extension classes.

This chapter describes each class’ methods and properties in detail.

**lsxECLConnection Class**

The lsxECLConnection class provides information about a Personal Communications connection.

An lsxECLConnection object is associated with a Personal Communications connection when the lsxECLConnection object is created. You cannot change the connection associated with an lsxECLConnection object. If you want to query information about a different connection, you must create a new lsxECLConnection object associated with that connection.

There are two ways to create an lsxECLConnection object:

- Create a new lsxECLConnection object by passing a Personal Communications connection name as a parameter on the new statement. A Personal Communications connection name is a single, alphabetic character from A-Z. The following is an example of creating an lsxECLConnection object that is associated with Personal Communications connection A:

  ```lotuscript
  Create an lsxECLConnection object associated with PCOMM connection A
  dim myConnObj as new lsxECLConnection("A")
  ```

- Create a new lsxECLConnection object by passing a Personal Communications connection handle as a parameter on the new statement. A Personal
Communications connection handle is a Long integer. The following is another example of creating an IsxECLConnection object that is associated with Personal Communications connection A:

' Create an IsxECLConnection object using a connection handle
dim myPSObj as new IsxECPS("A")

' Now use the connection handle from the PS object to build a connection object
dim myConnObj as new IsxECLConnection(myPSObj.Handle)

Properties

This section describes the properties for the IsxECLConnection class.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>Name</td>
<td>Read-only</td>
</tr>
<tr>
<td>Long</td>
<td>Handle</td>
<td>Read-only</td>
</tr>
<tr>
<td>String</td>
<td>ConnType</td>
<td>Read-only</td>
</tr>
<tr>
<td>Long</td>
<td>CodePage</td>
<td>Read-only</td>
</tr>
<tr>
<td>Integer</td>
<td>Started</td>
<td>Read-only</td>
</tr>
<tr>
<td>Integer</td>
<td>CommStarted</td>
<td>Read-only</td>
</tr>
<tr>
<td>Integer</td>
<td>APIEnabled</td>
<td>Read-only</td>
</tr>
<tr>
<td>Integer</td>
<td>Ready</td>
<td>Read-only</td>
</tr>
</tbody>
</table>

Name

Name is the connection name of the Personal Communications connection associated with this IsxECLConnection object. The Name property is a String data type and is read-only. Personal Communications connection names are one character in length and from the character set A-Z. The following example shows this property.

' Create an IsxECLConnMgr object to get the list of
' connections on the system.
dim myCMgrObj as new IsxECLConnMgr
dim myName as String

' Get the connection name for the first connection
' in the connection list.
myName = myCMgrObj.ConnList(1).Name

Handle

Handle is the connection handle of the Personal Communications connection associated with this IsxECLConnection object. The Handle property is a Long data type and is read-only. The following example shows this property.

' Create a new IsxECLConnection object associated with connection A
dim myConnObj as new IsxECLConnection("A")
dim myHandle as Long

' Get the connection handle for connection A
myHandle = myConnObj.Handle

ConnType

ConnType is the connection type of the connection that is associated with this IsxECLConnection object. The ConnType property is a String data type and is read-only. The following example shows this property.
Create a new lsxECLConnection object associated with connection A

dim myConnObj as new lsxECLConnection("A")
dim myConnType as String

' Get the Connection type for connection A
myConnType = myConnObj.ConnType

Connection types for the ConnType property are:

<table>
<thead>
<tr>
<th>String Returned</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISP3270</td>
<td>3270 display</td>
</tr>
<tr>
<td>DISP5250</td>
<td>5250 display</td>
</tr>
<tr>
<td>PRNT3270</td>
<td>3270 printer</td>
</tr>
<tr>
<td>PRNT5250</td>
<td>5250 printer</td>
</tr>
<tr>
<td>ASCII</td>
<td>VT emulation</td>
</tr>
<tr>
<td>UNKNOWN</td>
<td>Unknown</td>
</tr>
</tbody>
</table>

**CodePage**

CodePage is the code page of the connection associated with this lsxECLConnection object. The CodePage property is a Long data type, is read-only and cannot be changed through this LotusScript interface. However, the code page of a connection may change if the Personal Communications connection is restarted with a new configuration (see "lsxECLConnMgr Class" on page 337 for information about starting a connection). The following example shows this property.

Create a new lsxECLConnection object associated with connection A

dim myConnObj as new lsxECLConnection("A")
dim myCodePage as Long

' Get the CodePage for connection A
myCodePage = myConnObj.CodePage

**Started**

Started is a Boolean flag that indicates whether the connection associated with this lsxECLConnection object is started. The Started property is an integer and is read-only. Started is 1 if the Personal Communications connection has been started; otherwise, it is 0. The following example shows this property.

Create a new lsxECLConnection object associated with connection A

' See if connection is started
if myConnObj.Started then
    call connection_started

**CommStarted**

CommStarted is a Boolean flag that indicates whether the connection associated with this lsxECLConnection object is connected to the host data stream. The CommStarted property is an integer and is read-only. CommStarted is 1 if there is communication with the host; otherwise, it is 0. The following example shows this property.

Create a new lsxECLConnection object associated with connection A

' See if we are communicating with the host
if myConnObj.CommStarted then
    call connection_connected
**APIEnabled**

APIEnabled is a Boolean flag that indicates whether the HLLAPI API has been enabled for the Personal Communications connection associated with this lsxECLConnection object. The APIEnabled property is an integer and is read-only. APIEnabled is 1 if the HLLAPI API is available; otherwise, it is 0. The following example shows this property.

```
' Create a new lsxECLConnection object associated with connection A
dim myConnObj as new lsxECLConnection("A")

' See if the HLLAPI API is enabled on this connection
if myConnObj.APIEnabled then
call hllapi_available
```

**Ready**

Ready is a Boolean flag that indicates whether the Personal Communications connection associated with this lsxECLConnection object is ready. The Ready property is a combination of the Started, CommStarted and APIEnabled properties. It is an integer and is read-only. Ready is 1 if the Started, CommStarted and APIEnabled properties are 1; otherwise, it is 0. The following example shows this property.

```
' Create a new lsxECLConnection object associated with connection A
dim myConnObj as new lsxECLConnection("A")

' See if the connection is ready
if myConnObj.Ready then
call conn_ready
```

**lsxECLConnection Methods**

The following section describes the methods that are valid for the lsxECLConnection class.

StartCommunication()
StopCommunication()

**StartCommunication**

This method connects the ECL Connection to the host data stream. The effect is the same as using the **Connect** option on the Personal Communications emulator **Communication** menu.

**Prototype**

StartCommunication()

**Parameters**

None

**Return Value**

None

**Example**

The following example shows how to connect the ECL Connection to the host data stream.

```
' Create a new lsxECLConnection object for ECL Connection A
dim myConnObj as new lsxECLConnection("A")

' Make sure we have communications with the host
if myConnObj.CommStarted = 0 then
   myConnObj.StartCommunication
   ```
StopCommunication
This method disconnects the ECL Connection from the host data stream. The effect is the same as using the Disconnect option on the Personal Communications emulator Communication menu.

Prototype
StopCommunication()

Parameters
None

Return Value
None

Example
The following example shows how to disconnect the ECL Connection from the host data stream.
' Create a new lsxECLConnection object for ECL Connection A
dim myConnObj as new lsxECLConnection("A")

' Stop communications with the host on this connection
if myConnObj.CommStarted = 1 then
    myConnObj.StopCommunication

lsxECLConnList Class
The lsxECLConnList class manages the Personal Communications connections on a system. An lsxECLConnList object contains a list of all the connections that are currently available on the system. Each element of the connection list is an lsxECLConnection object. lsxECLConnection objects can be queried to determine the state of the associated connection. See "lsxECLConnection Class" on page 330 for details on its methods and properties.

An lsxECLConnList object provides a snapshot of the current connections on a system. The Refresh method provides a way to take a new snapshot of the connections on a system. The order of the connections in the lsxECLConnList is undefined and could change as a result of calling the Refresh method.

There are two ways to create an lsxECLConnList object:

• Create a new lsxECLConnList object by using the new statement. There are no parameters used when creating the lsxECLConnList object. The following is an example of creating an lsxECLConnList object:
  ' Create an lsxECLConnList object
dim myListObj as new lsxECLConnList

• Create an lsxECLConnMgr object and an lsxECLConnList object is automatically created. Access the lsxECLConnList attribute of the lsxECLConnMgr object to get to the lsxECLConnList object contained in the lsxECLConnMgr object. The following is an example of accessing the lsxECLConnList object contained in an lsxECLConnMgr object:
  dim myMgrObj as new lsxECLConnMgr
  dim myListObj as lsxECLConnList

  ' Get the lsxECLConnList object from inside the lsxECLConnMgr
  set myListObj = myMgrObj.lsxECLConnList
Properties

This section describes the properties of the lsxECLConnList class.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long</td>
<td>Count</td>
<td>Number of connections in the connection list</td>
</tr>
</tbody>
</table>

Count

Count is the number of connections present in the lsxECLConnList. The Count property is a Long data type and is read-only. The following example shows this property.

```vbnet
Dim myCMgrObj as New lsxECLConnMgr
Dim myCListObj as lsxECLConnList
Set myCListObj = myCMgrObj.lsxECLConnList

Dim numConns as Long

' Get a current snapshot of connections on the system
myCListObj.Refresh

' Get number of connections
numConns = myCListObj.Count
```

lsxECLConnList Methods

The following section describes the methods that are valid for the lsxECLConnList class.

Refresh()

FindConnectionByHandle(Long Handle)

FindConnectionByName(String Name)

Refresh

This method gets a list of the connections available on a system.

Prototype

Refresh()

Parameters

None

Return Value

None

Example

The following example shows how to use the Refresh method to get a current list of connections.

```
'Create a new lsxECLConnMgr
Dim myCMgrObj as New lsxECLConnMgr

'Get the lsxECLConnList contained in the lsxECLConnMgr
Dim myCListObj as lsxECLConnList
Set myCListObj = myCMgrObj.lsxECLConnList

Later...

'Refresh the list of connections found in lsxECLConnList
myCListObj.Refresh
```
**FindConnectionByHandle**

This method finds the connection identified by the **Handle** parameter in the **lsxECLConnList** list of connections.

**Prototype**

```plaintext```
FindConnectionByHandle( Long Handle )
```

**Parameters**

- **Long Handle**: The connection handle of the target connection.

**Return Value**

- **lsxECLConnection**: The **lsxECLConnection** object corresponding to the target connection.

**Example**

The following example shows how to find the connection identified by the **Handle** parameter.

```plaintext```
dim myConnObj as lsxECLConnection

'Create a new lsxECLConnList object
dim myCListObj as new lsxECLConnList

'Create a new lsxECLPS associated with connection A
dim myPSObj as new lsxECLPS("A")

'Get the lsxECLConnection object for connection A
set myConnObj = myCListObj.FindConnectionByHandle(myPSObj.Handle)
```

**FindConnectionByName**

This method finds a connection identified by the **Name** parameter in the **lsxECLConnList** list of connections.

**Prototype**

```plaintext```
FindConnectionByName(String Name)
```

**Parameters**

- **String Name**: The connection name of the target connection.

**Return Value**

- **Long Handle**: The connection handle of the target connection.

**Example**

The following example shows how to find a connection identified by the **Name** parameter.

```plaintext```
dim myConnObj as lsxECLConnection

'Create a new lsxECLConnList object
dim myCListObj as new lsxECLConnList

'Create a new lsxECLPS associated with connection A
dim myPSObj as new lsxECLPS("A")

'Get the lsxECLConnection object for connection A
set myConnObj = myCListObj.FindConnectionByName("A")
```
lsxECLConnMgr Class

The lsxECLConnMgr class manages Personal Communications connections on a system. It contains methods relating to the management of connections such as starting, stopping and querying connections. It also contains an lsxECLConnList object that is a static list of the connections available when the list was created (see "lsxECLConnList Class" on page 334 for more details on the lsxECLConnList class).

To create an lsxECLConnMgr object, use the new statement. There are no parameters used when creating the lsxECLConnMgr object. The following is an example of creating an lsxECLConnMgr object:

```
'Create an lsxECLConnMgr object
dim myCMgrObj as new lsxECLConnMgr
```

Properties

This section describes the properties of the lsxECLConnMgr class.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>lsxECLConnList</td>
<td>lsxECLConnList</td>
<td>Read-only</td>
</tr>
</tbody>
</table>

lsxECLConnList

The lsxECLConnMgr object contains an lsxECLConnList object. See "lsxECLConnList Class" on page 334 for details on the lsxECLConnList methods and properties. The following example shows this object:

```
' Create a new Connection manager
dim myCMgrObj as new lsxECLConnMgr

dim NumConns as Long

' Get the number of connections currently available on the system
NumConns = myCMgrObj.lsxECLConnList.Count
```

lsxECLConnMgr Methods

The following section explains the methods that are valid for the lsxECLConnMgr class.

StartConnection(String ConfigParms)
StopConnection(Long Handle, [optional], StringStopParms
StopConnection(String Name, [optional], StringStopParms)

StartConnection

This method starts a new Personal Communications emulator connection. The ConfigParms parameter contains Personal Communications connection startup information (see Usage Notes for an explanation of the startup information).

Prototype

StartConnection(String ConfigParms)

Parameters

String ConfigParms Personal Communications connection startup information.
Return Value
None

Example
The following example shows how to start a new Personal Communications emulator connection.

' Create a connection manager
dim myCMgrObj as new lsxECLConnMgr

' Start a new PCOMM connection
myCMgrObj.StartConnection("profile=coax Name=e")

Usage Notes
The connection configuration string is implementation-specific. Different implementations of the lsxECLConnMgr class may require different formats or information in the configuration string. The new connection is started upon return from this call, but it may or may not be connected to the host.

For Personal Communications, the configuration string has the following format:
PROFILE='[']<filename>' [NAME=<c>] [WINSTATE=<MAX|MIN|RESTORE|HIDE>]

Optional parameters are enclosed in square brackets []. The parameters are separated by at least one blank. Parameters may be in upper, lower, or mixed case and may appear in any order. The meaning of each parameter is as follows:

- PROFILE=<filename>: Names the Personal Communications workstation profile (.WS file), which contains the configuration information. This parameter is not optional; a profile name must be supplied. If the file name contains blanks the name must be enclosed in single quotation marks. The <filename> value may be either the profile name with no extension, the profile name with the .WS extension, or the fully qualified profile name path.
- NAME=<c> specifies the short ID of the new connection. This value must be a single, alphabetic character (A-Z). If this value is not specified, the next available connection ID is assigned automatically. If a connection already exists with the specified ID a connection not Open error is thrown.
- WINSTATE=<MAX|MIN|RESTORE|HIDE> specifies the initial state of the emulator window. The default if this parameter is not specified is RESTORE.

StopConnection
This method stops the Personal Communications connection identified by the Handle parameter. The StopParms parameters are additional Personal Communications stop connection parameters. See Usage Notes for an explanation of the valid values of StopParms.

Prototype
StopConnection(Long Handle, [optional], StringStopParms)
StopConnection(String Name, [optional], StringStopParms)

Parameters
Long Handle Connection handle of the connection to be stopped.

String Name One-character string short name of the connection (A-Z)

String StopParms Personal Communications connection stop parameters. This parameter is optional.
Return Value
None

Example
The following example shows how to stop the Personal Communications connection identified by the Handle parameter.

' Create a new connection manager
dim myCMgrObj as new lsxECLConnMgr

' Stop the first connection found in the list
myCMgrObj.StopConnection(myCMgrObj.lsxECLConnList(1).Handle, "saveprofile=no")

Usage Notes
The connection stop parameter string is implementation-specific. Different implementations of the lsxECLConnMgr class may require a different format and contents of the parameter string. For Personal Communications the string has the following format:

[SAVEPROFILE=<YES|NO|DEFAULT>]

Optional parameters are enclosed in square brackets []. The parameters are separated by at least one blank. Parameters may be in upper, lower, or mixed case and may appear in any order. The meaning of each parameter is as follows:

- SAVEPROFILE=<YES|NO|DEFAULT> controls the saving of the current connection configuration back to the workstation profile (.WS file). This causes the profile to be updated with any configuration changes you may have made during the connection. If NO is specified, the connection is stopped and the profile is not updated. If YES is specified, the connection is stopped and the profile is updated with the current (possibly changed) configuration. If DEFAULT is specified, the update option is controlled by the File->Save On Exit emulator menu option. If this parameter is not specified, DEFAULT is used.

lsxECLField Class
lsxECLField contains information for a given field from an lsxECLFieldList object residing in an lsxECLPS object. The only way to obtain an lsxECLField object is to access it through the lsxECLFieldList object.

Properties
This section describes the properties for the lsxECLField class.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long</td>
<td>StartRow</td>
<td>Read-only</td>
</tr>
<tr>
<td>Long</td>
<td>StartCol</td>
<td>Read-only</td>
</tr>
<tr>
<td>Long</td>
<td>EndRow</td>
<td>Read-only</td>
</tr>
<tr>
<td>Long</td>
<td>EndCol</td>
<td>Read-only</td>
</tr>
<tr>
<td>Long</td>
<td>Length</td>
<td>Read-only</td>
</tr>
<tr>
<td>Integer</td>
<td>Modified</td>
<td>Read-only</td>
</tr>
<tr>
<td>Integer</td>
<td>Protected</td>
<td>Read-only</td>
</tr>
<tr>
<td>Integer</td>
<td>Numeric</td>
<td>Read-only</td>
</tr>
<tr>
<td>Integer</td>
<td>HighIntensity</td>
<td>Read-only</td>
</tr>
<tr>
<td>Integer</td>
<td>PenDetectable</td>
<td>Read-only</td>
</tr>
</tbody>
</table>
### lsxECLField

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>Display</td>
<td>Read-only</td>
</tr>
</tbody>
</table>

**StartRow**

StartRow is the row of the first character of the field. The StartRow property is a Long data type and is read-only. The following example shows this property.

```vba
' Create a new PS object associated with connection A
dim myPSObj as new lsxECLPS("A")

dim StartRow as Long

' Refresh the list of fields
myPSObj.lsxECLFieldList.Refresh
If (myPSObj.lsxECLFieldList.Count) Then
' Get the starting row of the first field in the list
  StartRow = myPSObj.lsxECLFieldList(1).StartRow
Endif
```

**StartCol**

StartCol is the column of the first character of the field. The StartCol property is a Long data type and is read-only. The following example shows this property.

```vba
' Create a new PS object associated with connection A
dim myPSObj as new lsxECLPS("A")

dim StartCol as Long

' Refresh the list of fields
myPSObj.lsxECLFieldList.Refresh
If (myPSObj.lsxECLFieldList.Count) Then
' Get the starting column of the first field in the list
  StartCol = myPSObj.lsxECLFieldList(1).StartCol
Endif
```

**EndRow**

EndRow is the row of the last character of the field. The EndRow property is a Long data type and is read-only. The following example shows this property.

```vba
' Create a new PS object associated with connection A
dim myPSObj as new lsxECLPS("A")

dim EndRow as Long

' Refresh the list of fields
myPSObj.lsxECLFieldList.Refresh
If (myPSObj.lsxECLFieldList.Count) Then
' Get the ending row of the first field in the list
  EndRow = myPSObj.lsxECLFieldList(1).EndRow
Endif
```

**EndCol**

EndCol is the column of the last character of the field. The EndCol property is a Long data type and is read-only. The following example shows this property.

```vba
' Create a new PS object associated with connection A
dim myPSObj as new lsxECLPS("A")

dim EndCol as Long

' Refresh the list of fields
myPSObj.lsxECLFieldList.Refresh
```
If (myPSObj.lsxECLFieldList.Count) Then
' Get the ending column of the first field in the list
   EndCol = myPSObj.lsxECLFieldList(1).EndCol
Endif

**Length**

Length is the length of the field. The Length property is a Long data type and is 
read-only. The following example shows this property.

' Create a new PS object associated with connection A
dim myPSObj as new lsxECLPS("A")

dim length as Long

' Refresh the list of fields
myPSObj.lsxECLFieldList.Refresh
If (myPSObj.lsxECLFieldList.Count) Then
' Get the length of the first field in the list
   length = myPSObj.lsxECLFieldList(1).Length
Endif

**Modified**

Modified is a Boolean flag that indicates whether this field has been modified. A 
value of 1 means the field has been modified; otherwise, the value is 0. This 
property is read-only. The following example shows this property.

' Create a new PS object associated with connection A
dim myPSObj as new lsxECLPS("A")

' Refresh the list of fields
myPSObj.lsxECLFieldList.Refresh
if (myPSObj.lsxECLFieldList.Count) then
' Check if the first field in the list has been modified
   if (myPSObj.lsxECLFieldList(1).Modified) then
      call field_modified
   endif
endif

**Protected**

This is a Boolean flag that indicates whether the field has a protected attribute. A 
value of 1 means the field has the protected attribute; otherwise, the value is 0. 
This property is read-only. The following example shows this property.

' Create a new PS object associated with connection A
dim myPSObj as new lsxECLPS("A")

' Refresh the list of fields
myPSObj.lsxECLFieldList.Refresh
if (myPSObj.lsxECLFieldList.Count) then
' Check if the first field in the list is protected
   if (myPSObj.lsxECLFieldList(1).Protected) then
      call field_protected
   endif
endif

**Numeric**

This is a Boolean flag that indicates whether the field has the numeric-only input 
attribute. A value of 1 means the field has the numeric-only attribute; otherwise, 
the value is 0. This property is read-only. The following example shows this 
property.

' Create a new PS object associated with connection A
dim myPSObj as new lsxECLPS("A")

' Refresh the list of fields
myPSObj.lsxECLFieldList.Refresh
if (myPSObj.lsxECLFieldList.Count) then
    ' Check if the first field has the numeric only attribute
    if (myPSObj.lsxECLFieldList(1).Numeric) then
        call numeric_field
    endif
endif
endif

HighIntensity
This is a Boolean flag that indicates whether the field has the high intensity attribute. A value of 1 means the field has the high intensity attribute; otherwise, the value is 0. This property is read-only. The following example shows this property.
    ' Create a new PS object associated with connection A
    dim myPSObj as new lsxECLPS("A")
    ' Refresh the list of fields
    myPSObj.lsxECLFieldList.Refresh
    If (myPSObj.lsxECLFieldList.Count) Then
        ' Check if the first field has the high intensity attribute
        if (myPSObj.lsxECLFieldList(1).HighIntensity) then
            call high_intensity_field
        endif
    Endif

PenDetectable
This is a Boolean flag that indicates whether this field has the pen detectable attribute. A value of 1 means the field does have the pen detectable attribute; otherwise, the value is 0. This property is read-only. The following example shows this property.
    ' Create a new PS object associated with connection A
    dim myPSObj as new lsxECLPS("A")
    ' Refresh the list of fields
    myPSObj.lsxECLFieldList.Refresh
    If (myPSObj.lsxECLFieldList.Count) Then
        ' Check if the first field is pen detectable
        if (myPSObj.lsxECLFieldList(1).PenDetectable) then
            call field_pen_detectable
        endif
    Endif

Display
This is a Boolean flag that indicates whether this field has the display attribute. A value of 1 means that the field has the display attribute; otherwise, the value is 0. This property is read-only. The following example shows this property.
    ' Create a new PS object associated with connection A
    dim myPSObj as new lsxECLPS("A")
    ' Refresh the list of fields
    myPSObj.lsxECLFieldList.Refresh
    If (myPSObj.lsxECLFieldList.Count) Then
        ' Check if the first field has the display attribute
        if (myPSObj.lsxECLFieldList(1).Display) then
            call display_field
        endif
    Endif

IsxECLField Methods
The following section describes the methods that are valid for the lsxECLField class.
GetText

This method retrieves the characters of the field from the text plane.

**Prototype**

GetText()

**Parameters**

None

**Return Value**

String

A string of characters from the text plane.

**Example**

The following example shows how to retrieve the characters of the field:

```
' Create a new PS object associated with connection A
dim myPSObj as new lsxECLPS("A")

dim fieldData as String

' Refresh the list of fields
myPSObj.lsxECLFieldList.Refresh
If (myPSObj.lsxECLFieldList.Count) Then
  ' Get the characters from the first field's text plane
  fieldData = myPSObj.lsxECLFieldList(1).GetText()
Endif
```

SetText

This method sends a string of characters to the field. The Text parameter is a String data type. If the text exceeds the length of the field, the text is truncated.

**Prototype**

SetText(String Text)

**Parameters**

String Text

String of characters to send to the field.

**Return Value**

None

**Example**

The following example shows how to send a string of characters to the field.

```
' Create a new PS object associated with connection A
dim myPSObj as new lsxECLPS("A")

' Refresh the list of fields
myPSObj.lsxECLFieldList.Refresh
If (myPSObj.lsxECLFieldList.Count) Then
  ' Send a string of characters to the first field
  myPSObj.lsxECLFieldList(1).SetText("This is a test")
Endif
```
**lsxECLFieldList Class**

The `lsxECLFieldList` class performs operations on fields in a connection’s presentation space. An `lsxECLFieldList` object is contained in an `lsxECLPS` object and can only be accessed through an existing `lsxECLPS` object. See “`lsxECLPS Class`” on page 355 for more information on `lsxECLPS` objects.

An `lsxECLFieldList` object provides a static snapshot of what the presentation space contained when the Refresh method was called. The `lsxECLFieldList` class contains a list of all the fields in a given presentation space. Each element of the collection is an `lsxECLField` object. See “`lsxECLField Class`” on page 339 for more information about the `lsxECLField` objects.

**Properties**

This section describes the properties of the `lsxECLFieldList` class.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long</td>
<td>Count</td>
<td>Read-only</td>
</tr>
</tbody>
</table>

**Count**

Count is the number of fields in the `lsxECLFieldList` list. This value could change after each call to the Refresh method. The Count property is a Long data type and is read-only. The following example shows this property.

```
' Create a new PS object associated with connection A
dim myPSObj as new lsxECLPS("A")

dim numFields as Long

' Refresh the list of fields
myPSObj.lsxECLFieldList.Refresh

' Get the field that contains row 2, column 1
numFields = myPSObj.lsxECLFieldList.Count
lsxECLFieldList
```

**lsxECLFieldList Methods**

The following section describes the methods that are valid for the `lsxECLFieldList` class.

- `Refresh()`
- `FindFieldByRowCol(Long row, Long col)`
- `FindFieldByText(String Text, [optional] Long dir, [optional] Long row, [optional] Long col)`

**Refresh**

This method refreshes the list of `lsxECLField` objects contained in the `lsxECLFieldList` object.

**Prototype**

```
Refresh()
```

**Parameters**

None
Return Value
None

Example
The following example shows how to refresh the list of lsxECLField objects contained in the lsxECLFieldList object.

```lisp
' Create a new connection manager
dim myCMgr as new lsxECLConnMgr

dim myPSObj as lsxECLPS
set myPSObj = myCMgr.lsxECLConnList(1).Handle

dim numFields as Long

' Build the field list and get the number of fields
myPSObj.lsxECLFieldList.Refresh
numFields = myPSObj.lsxECLFieldList.Count
```

FindFieldByRowCol
This method finds an lsxECLField object in the lsxECLFieldList that contains the position indicated by the row and col parameters, which is a position in the presentation space. See "lsxECLField Class" on page 339 for the methods and properties of the lsxECLField object.

Prototype
FindFieldByRowCol(Long row, Long col)

Parameters
- Long row: Row position in the presentation space.
- Long col: Column position in the presentation space.

Return Value
ECLField: ECLField object.

Example
The following example shows how to find an lsxECLField object in the lsxECLFieldList that contains the position indicated by the row and col parameters.

dim myFInfoObj as lsxECLField

' Create a new PS object associated with connection A
dim myPSObj as new lsxECLPS("A")

' Refresh the list of fields
myPSObj.lsxECLFieldList.Refresh

' Get the field that contains row 2, column 1
myFInfoObj = myPSObj.lsxECLFieldList.FindFieldByRowCol(2,1)
```

FindFieldByText
This method finds the lsxECLField object in the lsxECLFieldList that contains the location of the string provided in the Text parameter. The search starts at the location indicated by the row and col parameters. If the row and col parameters are not specified, the search starts at the beginning of the presentation space. The row and col parameters must both be specified or omitted. The optional dir parameter indicates the direction to search.
**Prototype**

FindFieldByText(String Text, [optional] Long dir, [optional Long row, [optional] Long col)

**Parameters**

- **String Text**: Target string to search for in the presentation space.
- **Long dir**: Direction in which to search. Valid values are 1 for Search Forward and 2 for Search Backward. The default is 1, Search Forward.
- **Long row**: Target row in the presentation space. This parameter is optional. If not specified, the search starts at the beginning of the presentation space. If row is specified, col must also be specified.
- **Long col**: Target column in the presentation space. This parameter is optional. If it is not specified, the search starts at the beginning of the presentation space. If col is specified, row must also be specified.

**Return Value**

- **lsxECLField**: An lsxECLField object.

**Example**

The following example shows how to search for the lsxECLField object that contains a specified string.

```vba
' Create an lsxECLPS object associated with ECL Connection A
dim myPSObj as new lsxECLPS("A")
dim myFieldObj as lsxECLField

' Refresh the list of fields
myPSObj.lsxECLFieldList.Refresh

' Search for the field containing the specified string.
' The search direction defaults to forward and the search
' will start from the beginning of the presentation space.
set myFieldObj = myPSObj.lsxECLFieldList.FindFieldByText("Target Text")
```

**IsxECLFieldList**

The IsxECLFieldList class provides status information from a connection’s operator information area.

The IsxECLField object is associated with a Personal Communications connection when the IsxECLField object is created. You cannot change the connection that is associated with an IsxECLField object. If you want to query the OIA of a different connection, you must create a new IsxECLField object associated with that connection.

There are three ways to create an IsxECLField object:

1. Create a new IsxECLField object by passing a Personal Communications connection name as a parameter on the new statement. A Personal Communications connection name is a single, alphabetic character from A-Z.
The following is an example of creating an lsxECLOIA object that is associated with Personal Communications connection A:

```lisp
' Create an lsxECLOIA object associated with PCOMM connection A
dim myOIAObj as new lsxECLOIA("A")
```

2. Create a new lsxECLOIA object by passing a Personal Communications connection handle as a parameter on the new statement. A Personal Communications connection handle is a Long integer and is usually obtained by querying the lsxECLConnection object corresponding to the target Personal Communications connection (see “lsxECLConnMgr Class” on page 337, “lsxECLConnList Class” on page 334 and “lsxECLConnection Class” on page 330 for more information on the properties and methods of those objects). The following is an example of creating an lsxECLOIA object using a Personal Communications connection handle:

```lisp
dim myOIAObj as lsxECLOIA
dim myConnObj as new lsxECLConnection

' Create a new lsxECLOIA object using a connection handle
set myOIAObj = new lsxECLOIA(myConnObj.Handle)
```

3. Create an lsxECLSession object to create an lsxECLOIA object. After creating the lsxECLSession object, access its lsxECLOIA attribute to get access to the lsxECLOIA object contained in the lsxECLSession object. The following is an example of accessing the lsxECLOIA object contained in an lsxECLSession object:

```lisp
dim myOIAObj as lsxECLOIA

' Create a new lsxECLSession object associated with connection A
dim mySessObj as new lsxECLSession("A")

' Get the lsxECLOIA object from the lsxECLSession object
set myOIAObj = mySessObj.lsxECLOIA
```

### Properties

This section describes the properties for the lsxECLOIA class.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>Alphanumeric</td>
<td>Read-only</td>
</tr>
<tr>
<td>Integer</td>
<td>APL</td>
<td>Read-only</td>
</tr>
<tr>
<td>Integer</td>
<td>Katakana</td>
<td>Read-only</td>
</tr>
<tr>
<td>Integer</td>
<td>Hiragana</td>
<td>Read-only</td>
</tr>
<tr>
<td>Integer</td>
<td>DBCS</td>
<td>Read-only</td>
</tr>
<tr>
<td>Integer</td>
<td>UpperShift</td>
<td>Read-only</td>
</tr>
<tr>
<td>Integer</td>
<td>Numeric</td>
<td>Read-only</td>
</tr>
<tr>
<td>Integer</td>
<td>CapsLock</td>
<td>Read-only</td>
</tr>
<tr>
<td>Integer</td>
<td>InsertMode</td>
<td>Read-only</td>
</tr>
<tr>
<td>Integer</td>
<td>CommErrorReminder</td>
<td>Read-only</td>
</tr>
<tr>
<td>Integer</td>
<td>MessageWaiting</td>
<td>Read-only</td>
</tr>
<tr>
<td>Integer</td>
<td>InputInhibited</td>
<td>Read-only</td>
</tr>
<tr>
<td>String</td>
<td>Name</td>
<td>Read-only</td>
</tr>
<tr>
<td>Long</td>
<td>Handle</td>
<td>Read-only</td>
</tr>
<tr>
<td>String</td>
<td>ConnType</td>
<td>Read-only</td>
</tr>
<tr>
<td>Long</td>
<td>CodePage</td>
<td>Read-only</td>
</tr>
<tr>
<td>Type</td>
<td>Name</td>
<td>Attributes</td>
</tr>
<tr>
<td>------------</td>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Integer</td>
<td>Started</td>
<td>Read-only</td>
</tr>
<tr>
<td>Integer</td>
<td>CommStarted</td>
<td>Read-only</td>
</tr>
<tr>
<td>Integer</td>
<td>APIEnabled</td>
<td>Read-only</td>
</tr>
<tr>
<td>Integer</td>
<td>Ready</td>
<td>Read-only</td>
</tr>
</tbody>
</table>

**Alphanumeric**
This property queries the connection’s operator information area to determine if the field at the cursor position is alphanumeric. The Alphanumeric property is set to 1 if the field is alphanumeric; otherwise, it is set to 0. Alphanumeric is an Integer data type and is read-only. The following example shows this property.

'Create a new lsxECL OIA object associated with connection A
dim myOIAObj as new lsxECL OIA("A")

'Check if the field is alphanumeric
if myOIAObj.Alphanumeric then
call abc

**APL**
This property queries the connection’s operator information area to determine if the keyboard is in APL mode. The APL property is set to 1 if the keyboard is in APL mode; otherwise, it is set to 0. APL is an Integer data type and is read-only. The following example shows this property.

'Create a new lsxECL OIA object associated with connection A
dim myOIAObj as new lsxECL OIA("A")

'Check if the keyboard is in APL mode
if myOIAObj.APL then
call abc

**Katakana**
This property queries the connection’s operator information area to determine if Katakana characters are enabled. The Katakana property is set to 1 if Katakana characters are enabled; otherwise, it is set to 0. Katakana is an Integer data type and is read-only. The following example shows this property.

'Create a new lsxECL OIA object associated with connection A
dim myOIAObj as new lsxECL OIA("A")

'Check if Katakana characters are available
if myOIAObj.Katakana then
call abc

**Hiragana**
This property queries the connection’s operator information area to determine if Hiragana characters are enabled. The Hiragana property is set to 1 if Hiragana characters are enabled; otherwise, it is set to 0. Hiragana is an Integer data type and is read-only. The following example shows this property.

'Create a new lsxECL OIA object associated with connection A
dim myOIAObj as new lsxECL OIA("A")

'Check if Hiragana characters are available
if myOIAObj.Hiragana then
call abc

**DBCS**
This property queries the connection’s operator information area to determine if the field at the cursor position is DBCS. The DBCS property is set to 1 if the field...
is DBCS; otherwise, it is set to 0. DBCS is an Integer data type and is read-only. The following example shows this property.

' Create a new 1sxECLOIA object associated with connection A
dim myOIAObj as new 1sxECLOIA("A")

' Check if DBCS is available
if myOIAObj.DBCS then
    call abc

UpperShift
This property queries the connection’s operator information area to determine if the keyboard is in uppercase mode. The UpperShift property is set to 1 if the keyboard is in uppercase mode; otherwise, it is set to 0. UpperShift is an Integer data type and is read-only. The following example shows this property.

' Create a new 1sxECLOIA object associated with connection A
dim myOIAObj as new 1sxECLOIA("A")

' Check if the keyboard is in uppercase mode
if myOIAObj.UpperShift then
    call abc

Numeric
This property queries the connection’s operator information area to determine if the field at the cursor position is numeric. The Numeric property is set to 1 if the field is numeric; otherwise, it is set to 0. Numeric is an Integer data type and is read-only. The following example shows this property.

' Create a new 1sxECLOIA object associated with connection A
dim myOIAObj as new 1sxECLOIA("A")

' Check if the field is numeric
if myOIAObj.Numeric then
    call abc

CapsLock
This property queries the connection’s operator information area to determine if the keyboard is in capslock mode. The CapsLock property is set to 1 if the keyboard is in capslock mode, otherwise it is set to 0. CapsLock is an Integer data type and is read-only. The following example shows this property.

' Create a new 1sxECLOIA object associated with connection A
dim myOIAObj as new 1sxECLOIA("A")

' Check if the keyboard is in capslock mode
if myOIAObj.CapsLock then
    call abc

InsertMode
This property queries the connection’s operator information area to determine if the keyboard is in insert mode. The InsertMode property is set to 1 if the keyboard is in insert mode; otherwise, it is set to 0. InsertMode is an Integer data type and is read-only. The following example shows this property.

' Create a new 1sxECLOIA object associated with connection A
dim myOIAObj as new 1sxECLOIA("A")

' Check if the keyboard is in insert mode
if myOIAObj.InsertMode then
    call abc

CommErrorReminder
This property queries the connection’s operator information area to determine if a communications error reminder condition exists. The CommErrorReminder
**IsxECLOIA**

property is set to 1 if a communications error reminder condition exists; otherwise, it is set to 0. CommErrorReminder is an Integer data type and is read-only. The following example shows this property.

```
' Create a new IsxECLOIA object associated with connection A
dim myOIAObj as new IsxECLOIA("A")

' See if we have a communications error reminder
' condition on connection A
if myOIAObj.CommErrorReminder then
call abc
```

**MessageWaiting**

This property queries the connection’s operator information area to determine if the message waiting indicator is on. The MessageWaiting property is set to 1 if the message waiting indicator is on; otherwise, it is set to 0. MessageWaiting is an Integer data type and is read-only. The following example shows this property.

```
' Create a new IsxECLOIA object associated with connection A
' Assume connection A is a 5250 connection
dim myOIAObj as new IsxECLOIA("A")

' See if we have a message waiting on connection A
if myOIAObj.MessageWaiting then
call abc
```

The message waiting indicator is only used in connections of SessionType “DISP5250”. For other connection types, the MessageWaiting property is always set to 0.

**InputInhibited**

This property queries whether the host is ready for input. InputInhibited is an Integer data type and is read-only. The following table shows valid values for InputInhibited.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Not Inhibited</td>
</tr>
<tr>
<td>1</td>
<td>System Wait</td>
</tr>
<tr>
<td>2</td>
<td>Communication Check</td>
</tr>
<tr>
<td>3</td>
<td>Program Check</td>
</tr>
<tr>
<td>4</td>
<td>Machine Check</td>
</tr>
<tr>
<td>5</td>
<td>Other Inhibit</td>
</tr>
</tbody>
</table>

The following example shows this property.

```
' Create a new IsxECLOIA object associated with connection A
dim myOIAObj as new IsxECLOIA("A")

' See if the host is ready for input
if myOIAObj.InputInhibited = 0 then
   ' Okay to send text
call sendtext
```

**Name**

Name is the connection name of the Personal Communications connection associated with this IsxECLOIA object. The Name property is a String data type and is read-only. Personal Communications connection names are one character in length and from the character set A-Z. The following example shows this property.
Create an lsxECLIOIA object associated with connection A

```lsl
dim myOIAObj as new lsxECLIOIA("A")
```

`dim myName as String`

' Get our connection name

```lsl
myName = myOIAObj.Name
```

**Handle**

Handle is the connection handle of the Personal Communications connection associated with this lsxECLIOIA object. The Handle property is a Long data type and is read-only. The following example shows this property.

```lsl
dim myHandle as Long

' Get our connection handle
myHandle = myOIAObj.Handle
```

**ConnType**

ConnType is the connection type of the connection that is associated with this lsxECLIOIA object. The ConnType property is a String data type and is read-only. See [Usage Notes](#usage-notes) for the list of possible connection type values. The following example shows this property.

```lsl
dim myConnType as String

' Get the connection type for connection A
myConnType = myOIAObj.ConnType
```

Connection types for the ConnType property are:

<table>
<thead>
<tr>
<th>String Returned</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISP3270</td>
<td>3270 display</td>
</tr>
<tr>
<td>DISP5250</td>
<td>5250 display</td>
</tr>
<tr>
<td>PRNT3270</td>
<td>3270 printer</td>
</tr>
<tr>
<td>PRNT5250</td>
<td>5250 printer</td>
</tr>
<tr>
<td>ASCII</td>
<td>VT emulation</td>
</tr>
<tr>
<td>UNKNOWN</td>
<td>Unknown</td>
</tr>
</tbody>
</table>

**CodePage**

CodePage is the code page of the connection associated with this lsxECLIOIA object. The CodePage property is a Long data type, is read-only and cannot be changed through this LotusScript interface. However, the code page of a connection may change if the Personal Communications connection is restarted with a new configuration (see ["lsxECLConnMgr Class" on page 337](#lsxeclconnmgr-class)) for information about starting a connection). The following example shows this property.

```lsl
dim myOIAObj as new lsxECLIOIA("A")
```

---

Chapter 4. Host Access Class Library LotusScript Extension  351
dim myCodePage as Long

' Get the code page for connection A
myCodePage = myOIAObj.CodePage

**Started**

Started is a Boolean flag that indicates whether the connection associated with this lsxECLOIA object is started (for example, still running as a Personal Communications connection). The Started property is an integer and is read-only. Started is 1 if the Personal Communications connection has been started; otherwise, it is 0. The following example shows this property.

' Create an lsxECLOIA object associated with connection A
dim myOIAObj as new lsxECLOIA("A")

' See if our connection is started
if myOIAObj.Started then
    call connection_started

**CommStarted**

CommStarted is a Boolean flag that indicates whether the connection associated with this lsxECLOIA object is connected to the host data stream. The CommStarted property is an integer and is read-only. CommStarted is 1 if there is communication with the host; otherwise, it is 0. The following example shows this property.

' Create an lsxECLOIA object associated with connection A
dim myOIAObj as new lsxECLOIA("A")

' See if we are communicating with the host
if myOIAObj.CommStarted then
    call communications_started

**APIEnabled**

APIEnabled is a Boolean flag that indicates whether the HLLAPI API has been enabled for the Personal Communications connection associated with this lsxECLOIA object. The APIEnabled property is an integer and is read-only. APIEnabled is 1 if the HLLAPI API is available; otherwise, it is 0. The following example shows this property.

' Create an lsxECLOIA object associated with connection A
dim myOIAObj as new lsxECLOIA("A")

' See if the HLLAPI API is enabled on this connection
if myOIAObj.APIEnabled then
    call hllapi_available

**Ready**

Ready is a Boolean flag that indicates whether the Personal Communications connection associated with this lsxECLOIA object is ready. The Ready property is a combination of the Started, CommStarted and APIEnabled properties. It is an integer and is read-only. Ready is 1 if the Started, CommStarted and APIEnabled properties are 1; otherwise, it is 0. The following example shows this property.

' Create an lsxECLOIA object associated with connection A
dim myOIAObj as new lsxECLOIA("A")

' See if our connection is ready
if myOIAObj.Ready then
    call conn_ready
IsxECLOIA Methods

The following section describes the methods that are valid for the IsxECLOIA class.

```
Integer WaitForInputReady([optional] Long TimeOut)
Integer WaitForSystemAvailable([optional] Long TimeOut)
Integer WaitForAppAvailable([optional] Long TimeOut)
Integer WaitForTransition([optional] Long Index, [optional] Long timeout)
```

**WaitForInputReady**

The WaitForInputReady method waits until the OIA of the connection associated with the IsxECLOIA object indicates that the connection is able to accept keyboard input.

**Prototype**

```
Integer WaitForInputReady([optional] Long TimeOut)
```

**Parameters**

`Long TimeOut`  
The maximum length of time in milliseconds to wait. This parameter is optional. The default is Infinite.

**Return Value**

The method returns 1 if the condition is met, or 0 if the Timeout value is exceeded.

**Example**

```
Dim lsxECLOIAObj as new lsxECLOIA("A")

if (lsxECLOIAObj.WaitForInputReady(10000)) then
    MessageBox("Ready for input")
else
    MessageBox("Timeout occurred")
end if
```

**WaitForSystemAvailable**

The WaitForSystemAvailable method waits until the OIA of the connection associated with the IsxECLOIA object indicates that the connection is connected to an SNA host system and is ready for connection to an application.

**Prototype**

```
Integer WaitForSystemAvailable([optional] Long TimeOut)
```

**Parameters**

`Long TimeOut`  
The maximum length of time in milliseconds to wait. This parameter is optional. The default is Infinite.

**Return Value**

The method returns 1 if the condition is met, or 0 if the Timeout value is exceeded.

**Example**

```
Dim lsxECLOIAObj as new lsxECLOIA("A")

if (lsxECLOIAObj.WaitForSystemAvailable(10000)) then
    MessageBox("System Available")
```
lsxECL0IA

else
 MessageBox("Timeout Occurred")
end if

**WaitForAppAvailable**
The WaitForAppAvailable method waits while the OIA of the connection associated with the lsxECL0IA object indicates that the application is being worked with.

**Prototype**
Integer WaitForAppAvailable([optional] Long TimeOut)

**Parameters**
Long TimeOut  The maximum length of time in milliseconds to wait. This parameter is optional. The default is Infinite.

**Return Value**
The method returns 1 if the condition is met, or 0 if the Timeout value is exceeded.

**Example**
Dim lsxECL0IAObj as Object
Set lsxECL0IAObj = new lsxECL0IA("A")

if (lsxECL0IAObj.WaitForAppAvailable (10000)) then
 MessageBox("Application is available")
else
 MessageBox("Timeout Occurred")
end if

**WaitForTransition**
The WaitForTransition method waits for the OIA position specified of the connection associated with the lsxECL0IA object to change.

**Prototype**
Integer WaitForTransition([optional] Long Index, [optional] Long timeout)

**Parameters**
Long Index  The 1 byte Hex position of the OIA to monitor. This parameter is optional. The default is 3.

Long TimeOut  The maximum length of time in milliseconds to wait. This parameter is optional. The default is Infinite.

**Return Value**
The method returns 1 if the condition is met, or 0 if the Timeout value is exceeded.

**Example**
Dim lsxECL0IAObj as new lsxECL0IA("A")
Dim Index as Long

Index = 03h

if (lsxECL0IAObj.WaitForTransition(Index,10000)) then
 MessageBox("OIA changed")
else
    MessageBox("Timeout Occurred")
end if

IsxECLPS Class

The IsxECLPS class performs operations on a connection’s presentation space.

The IsxECLPS object is associated with a Personal Communications connection when the IsxECLPS object is created. You cannot change the connection that is associated with an IsxECLPS object. If you want to manipulate the presentation space of a different connection, you must create a new IsxECLPS object associated with that connection.

There are three ways to create an IsxECLPS object:

1. Create a new IsxECLPS object by passing a Personal Communications connection name as a parameter on the new statement. A Personal Communications connection name is a single, alphabetic character from A-Z. The following is an example of creating an IsxECLPS object that is associated with Personal Communications connection A:
   
   ```
   ' Create an lsxECLPS object associated with PCOMM connection A
   dim myPSObj as new lsxECLPS("A")
   ```

2. Create a new IsxECLPS object by passing a Personal Communications connection handle as a parameter on the new statement. A Personal Communications connection handle is a long integer and is usually obtained by querying the lsxECLConnection object corresponding to the target Personal Communications connection (see “IsxECLConnMgr Class” on page 337 and “lsxECLConnection Class” on page 330 for more information on the properties and methods of those objects). The following is an example of creating an IsxECLPS object using a Personal Communications connection handle:
   
   ```
   dim myPSObj as lsxECLPS
   dim myCMgrObj as new lsxECLConnMgr
   
   ' Create a new lsxECLPS object associated with the first PCOMM connection found in lsxECLConnList
   ' set myPSObj = new lsxECLPS(myCMgrObj.lsxECLConnList(1).Handle)
   ```

3. Create an lsxECLSession object and an lsxECLPS object is automatically created. Access the lsxECLPS attribute to get to the lsxECLPS object contained in the lsxECLSession object. The following is an example of accessing the lsxECLPS object contained in an lsxECLSession object:
   
   ```
   dim myPSObj as lsxECLPS
   dim mySessionObj as lsxECLSession
   
   ' Create a new lsxECLSession object associated with PCOMM connection A
   ' set mySessionObj = new lsxECLSession("A")
   ' Get the lsxECLPS object from the lsxECLSession object
   set myPSObj = mySessionObj.lsxECLPS
   ```

Note: In the presentation space, the first row coordinate is row 1 and the first column coordinate is column 1. Therefore, the top, left position has a coordinate of row 1, column 1.

Properties

This section describes the properties of the lsxECLPS class.
### lsxECLPS

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long</td>
<td>NumRows</td>
<td>Read-only</td>
</tr>
<tr>
<td>Long</td>
<td>NumCols</td>
<td>Read-only</td>
</tr>
<tr>
<td>Long</td>
<td>CursorPosRow</td>
<td>Read-only</td>
</tr>
<tr>
<td>Long</td>
<td>CursorPosCol</td>
<td>Read-only</td>
</tr>
<tr>
<td>lsxECLFieldList</td>
<td>lsxECLFieldList</td>
<td>Read-only</td>
</tr>
<tr>
<td>String</td>
<td>Name</td>
<td>Read-only</td>
</tr>
<tr>
<td>Long</td>
<td>Handle</td>
<td>Read-only</td>
</tr>
<tr>
<td>String</td>
<td>ConnType</td>
<td>Read-only</td>
</tr>
<tr>
<td>Long</td>
<td>CodePage</td>
<td>Read-only</td>
</tr>
<tr>
<td>Integer</td>
<td>Started</td>
<td>Read-only</td>
</tr>
<tr>
<td>Integer</td>
<td>CommStarted</td>
<td>Read-only</td>
</tr>
<tr>
<td>Integer</td>
<td>APIEnabled</td>
<td>Read-only</td>
</tr>
<tr>
<td>Integer</td>
<td>Ready</td>
<td>Read-only</td>
</tr>
</tbody>
</table>

**NumRows**

NumRows is the number of rows in this connection’s presentation space. The NumRows property is a Long data type and is read-only. The following example shows this property.

```vbscript
' Create an lsxECLPS object associated with connection A
dim myPSObj as new lsxECLPS("A")

dim Rows as Long

' Get the number of rows in our presentation space
Rows = myPSObj.NumRows
```

**NumCols**

NumCols is the number of columns in this connection’s presentation space. The NumCols property is a Long data type and is read-only. The following example shows this property.

```vbscript
: 
' Create an lsxECLPS object associated with connection A
dim myPSObj as new lsxECLPS("A")

dim Cols as Long

' Get the number of columns in our presentation space
Cols = myPSObj.NumCols
```

**CursorPosRow**

CursorPosRow is the row of the current cursor position in this connection’s presentation space. The CursorPosRow property is a Long data type and is read-only. The following example shows this property.

```vbscript
' Create an lsxECLPS object associated with connection A
dim myPSObj as new lsxECLPS("A")

dim CursorRow as Long

' Get the row location of the cursor in our presentation space
CursorRow = myPSObj.CursorPosRow
```
CursorPosCol

CursorPosCol is the column of the current cursor position in this connection's presentation space. The CursorPosCol is a Long data type and is read-only. The following example shows this property.

```vbs
' Create an lsxECLPS object associated with connection A
dim myPSObj as new lsxECLPS("A")

dim CursorCol as Long

' Get the cursor column location in our presentation space
CursorCol = myPSObj.CursorPosCol
```

lsxECLFieldList

The lsxECLPS object contains an lsxECLFieldList object. See "lsxECLFieldList Class" on page 344 for details on the lsxECLFieldList methods and properties. The following example shows this object.

```vbs
' Create an lsxECLPS object associated with PCOM connection A
dim myPSObj as new lsxECLPS("A")

dim numFields as Long

' Get the number of fields in the presentation space
numFields = myPSObj.lsxECLFieldList.Count
```

Name

Name is the connection name of the Personal Communications connection associated with this lsxECLPS object. The Name property is a String data type and is read-only. Personal Communications connection names are one character in length and from the set of A-Z. The following example shows this property.

```vbs
' Create an lsxECLPS object associated with connection A
dim myPSObj as new lsxECLPS("A")

dim myName as String

' Get our connection name
myName = myPSObj.Name
```

Handle

Handle is the connection handle of the Personal Communications connection associated with this lsxECLPS object. Handle is a Long data type and is read-only. The following example shows this property.

```vbs
' Create an lsxECLPS object associated with connection A
dim myPSObj as new lsxECLPS("A")

dim myHandle as Long

' Get our connection handle
myHandle = myPSObj.Handle
```

ConnType

ConnType is the connection type of the connection that is associated with this lsxECLPS object. The ConnType is a String data type and is read-only. The following example shows this property.

```vbs
' Create an lsxECLPS object associated with connection A
dim myPSObj as new lsxECLPS("A")

dim myConnType as String

' Get the connection type for connection A
myConnType = myPSObj.ConnType
```
Connection types for the ConnType property are:

<table>
<thead>
<tr>
<th>String Returned</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISP3270</td>
<td>3270 display</td>
</tr>
<tr>
<td>DISP5250</td>
<td>5250 display</td>
</tr>
<tr>
<td>PRNT3270</td>
<td>3270 printer</td>
</tr>
<tr>
<td>PRNT5250</td>
<td>5250 printer</td>
</tr>
<tr>
<td>ASCII</td>
<td>VT emulation</td>
</tr>
<tr>
<td>UNKNOWN</td>
<td>Unknown</td>
</tr>
</tbody>
</table>

**CodePage**

CodePage is the code page of the connection associated with this lsxECLPS object. The CodePage property is a Long data type, is read-only and cannot be changed through this LotusScript interface. However, the code page of a connection may change if the Personal Communications connection is restarted with a new configuration (see "lsxECLConnMgr Class" on page 337 for information about starting a connection). The following example shows this property.

```literate
' Create an lsxECLPS object associated with connection A
dim myPSObj as new lsxECLPS("A")

dim myCodePage as Long

' Get the code page for connection A
myCodePage = myPSObj.CodePage
```

**Started**

Started is a Boolean flag that indicates whether the connection associated with this lsxECLPS object is started (for example, still running as a Personal Communications connection). The Started property is an integer and is read-only. Started is 1 if the Personal Communications connection has been started; otherwise, it is 0. The following example shows this property.

```literate
' Create an lsxECLPS object associated with connection A
dim myPSObj as new lsxECLPS("A")

' See if our connection is started
if myPSObj.Started then
    call connection_started
```

**CommStarted**

CommStarted is a Boolean flag that indicates whether the connection associated with this lsxECLPS object is connected to the host data stream. CommStarted is an integer and is read-only. CommStarted is 1 if there is communication with the host; otherwise, it is 0. The following example shows this property.

```literate
' Create an lsxECLPS object associated with connection A
dim myPSObj as new lsxECLPS("A")

' See if we are communicating with the host
if myPSObj.CommStarted then
    call communications_started
```

**APIEnabled**

APIEnabled is a Boolean flag that indicates whether the HLLAPI API has been enabled for the Personal Communications connection associated with this lsxECLPS object. APIEnabled is an integer and is read-only. APIEnabled is 1 if the HLLAPI API is available; otherwise, it is 0. The following example shows this property.

```literate
' Create an lsxECLPS object associated with connection A
dim myPSObj as new lsxECLPS("A")

' See if the HLLAPI is enabled
if myPSObj.APIEnabled then
    call api_enabled
```
' Create an IsxECLPS object associated with connection A
dim myPSObj as new lsxECLPS("A")

' See if the HLLAPI API is enabled on this connection
if myPSObj.APIEnabled then
    call hllapi_available

Ready
The Ready property is a Boolean flag that indicates whether the Personal Communications connection associated with this lsxECLPS object is ready. The Ready property is a combination of the Started, CommStarted and APIEnabled properties. It is an integer and is read-only. Ready is 1 if the Started, CommStarted and APIEnabled properties are 1; otherwise, it is 0. The following example shows this property.

' Create an lsxECLPS object associated with connection A
dim myPSObj as new lsxECLPS("A")

' See if our connection is ready
if myPSObj.Ready then
    call conn_ready

IsxECLPS Methods
The following section describes the methods that are valid for the lsxECLPS class.

SetCursorPos(Long row, Long col)
SendKeys(String text, [optional] Long row, [optional] Long col)
SetText(String text, [optional] Long row, [optional] Long col)
String GetTextRect(Long startrow, Long startcol, Long endrow, Long endcol)
Integer WaitForCursor(Long Row, Long Col, [optional] Long TimeOut, [optional] Integer bWaitForIr)
Integer WaitForCursor(Long Row, Long Col, [optional] Long TimeOut, [optional] Integer bWaitForIr)
Integer WaitForStringInRect(String WaitString, Long sRow, Long sCol, Long eRow,Long eCol, [optional] Long nTimeOut, [optional] Integer bWaitForIr, [optional] Integer bCaseSens)
Integer WaitForStringInRect(String WaitString, Long sRow, Long sCol, Long eRow,Long eCol, [optional] Long nTimeOut, [optional] Integer bWaitForIr, [optional] Integer bCaseSens)
public Integer WaitForScreen(Object screenDesc, [optional] Long TimeOut)
public Integer WaitForScreen(Object screenDesc, [optional] Long TimeOut)

SetCursorPos
This method sets the position of the cursor in the presentation space of the connection associated with this lsxECLPS object. The cursor is set to the position indicated by the row and col parameters.
Prototype
SetCursorPos(Long row, Long col)

Parameters
Long row Target row for the cursor.
Long col Target column for the cursor.

Return Value
None

Example
The following example shows how to set the position of the cursor in the
presentation space of the connection associated with this lsxECLPS object.
' Create an lsxECLPS object associated with connection A
dim myPSObj as new lsxECLPS("A")

' Set the cursor location in the presentation space
myPSObj.SetCursorPos(3,1)

SendKeys
This method sends a string of keystrokes to the presentation space of the
connection associated with this lsxECLPS object. The string is positioned in the
presentation space at the position indicated by the row and col parameters. The
row and col parameters must be specified together. If the row and col parameters
are not specified, the string is sent to the current cursor position.

Prototype
SendKeys(String text, [optional] Long row, [optional] Long col)

Parameters
String text String of keys to send to the presentation space.
Long row Target row within the presentation space. This
parameter is optional. If the parameter is not
specified, the location defaults to the current cursor
row position. If row is specified, col must also be
specified.

Long col Target column within the presentation space. This
parameter is optional. If the parameter is not
specified, the location defaults to the current cursor
column position. If col is specified, row must also be
specified

Return Value
None

Example
The following example shows how to send a string of keystrokes to the
presentation space of the connection associated with this lsxECLPS object.
' Create an lsxECLPS object associated with connection A
dim PSObj as new lsxECLPS("A")

' Send a string of keystrokes to the cursor location in the presentation space
PSObj.SendKeys("[clear]QUERY DISK[ENTER]")
'Send a string of keystrokes to a specific location in the presentation space
PSObj.SendKeys("[clear]QUERY DISK[ENTER]", 23, 1)

Usage Notes
This method allows you to send mnemonic keystrokes to the presentation space. See Appendix A, “Sendkeys Mnemonic Keywords,” on page 397 for a list of these keystrokes.

SearchText
This method searches for the first occurrence of a text string in the presentation space of the connection associated with this IsxECLPS object. This method returns a 1 if text is found; otherwise it returns a 0. The search begins from the position specified by the row and col parameters. The row and col parameters must be specified together. If the row and col parameters are not specified, the search begins at the beginning of the presentation space for a search forward or the end of the presentation space for a search backward. The search direction can either be forward or backward, and can be specified using the dir parameter. If dir is not specified, the default is forward.

Prototype
Integer SearchText(String text, [optional] Long dir, [optional] Long row, [optional] Long col)

Parameters
String text Target text string.
Long dir Search direction. Must be 1 (Search forward) or 2 (Search Backward). This parameter is optional. If the parameter is not specified, the default is forward.
Long row Row position at which to start the search in the presentation space. The row of the located text is returned if the search is successful. This parameter is optional. If row is specified, col must also be specified.
Long col Column position at which to start the search in the presentation space. The column of the located text is returned if the search is successful. This parameter is optional. If col is specified, row must also be specified.

Return Value
Integer 1 if text found; 0 if text is not found.

Example
The following example shows how to search for the first occurrence of a text string in the presentation space of the connection associated with this IsxECLPS object.

' Create an IsxECLPS object associated with connection A
dim PSobj as new IsxECLPS("A")
dim tRow as Long
dim tCol as Long
tRow = 1
tCol = 1
GetText

This method retrieves a text string from the presentation space of the connection associated with this lsxECLPS object. The method returns a string starting at the position indicated by the row and col parameters for the length (len) parameter. If the row, col and len parameters are not specified, the entire presentation space is returned.

Prototype


Parameters

Long row
Target row in the presentation space. This parameter is optional. If it is not specified, the entire presentation space is returned.

Long col
Target column in the presentation space. This parameter is optional. If it is not specified, the entire presentation space is returned.

Long len
Length of text to retrieve from the presentation space. This parameter is optional. If it is not specified, the entire presentation space is returned.

Return Value

String
Text retrieved from the presentation space.

Example

The following example shows how to retrieve a text string from the presentation space of the connection associated with this lsxECLPS object.

' Create an lsxECLPS object associated with connection A
dim myPSObj as new lsxECLPS("A")

dim scrnText as String

' Get all the text from the text plane.
scrnText = myPSObj.GetText()

' Get 10 characters from the text plane starting at row 3, column 1
scrnText = myPSObj.GetText(3,1,10)

SetText

This method copies a text string to the presentation space of the connection associated with this lsxECLPS object. The string is copied to the location indicated by the row and col parameters. If the row and col parameters are not specified, the string is copied to the presentation space at the current cursor location. The row and col parameters must both be specified or omitted.
**Prototype**

SetText(String Text, [optional] Long row, [optional] Long col)

**Parameters**

- **String Text**
  String to copy to the presentation space.

- **Long row**
  Target row in the presentation space. This parameter is optional. If it is not specified, the current row position of the cursor is used. If row is specified, col must also be specified.

- **Long col**
  Target column in the presentation space. This parameter is optional. If it is not specified, the current col position of the cursor is used. If col is specified, row must also be specified.

**Return Value**

None

**Example**

The following example shows how to copy a text string to the presentation space of the connection associated with an lsxECLPS object.

```
\> Create an lsxECLPS object associated with ECL Connection A
\> dim myPSObj as new lsxECLPS("A")

\> Copy a string to the current cursor position in the Presentation
\> Space of ECL Connection A
\> myPSObj.SetText("Text to copy to PS")

\> Copy a string to a specific location in the Presentation Space
\> of ECL Connection A
\> myPSObj.SetText("Text to copy to PS", 23, 1)
```

**GetTextRect**

This method retrieves a text string from a rectangular area in the presentation space of the connection associated with this lsxECLPS object and returns a String data type. The rectangle is identified by the `startrow`, `startcol`, `endrow` and `endcol` parameters. No text wrapping is done during the text string retrieval; only the text within the designated rectangle is retrieved.

**Prototype**

String GetTextRect( Long startrow, Long startcol, Long endrow, Long endcol)

**Parameters**

- **Long startrow**
  Upper left row position of the rectangle in the presentation space.

- **Long startcol**
  Upper left column position of the rectangle in the presentation space.

- **Long endrow**
  Lower right row position of the rectangle in the presentation space.

- **Long endcol**
  Lower right column position of the rectangle in the presentation space.

**Return Value**

String

Text string retrieved from the presentation space.
**Example**

The following example shows how to retrieve a text string from a rectangular area in the presentation space of the connection associated with this IsxECLPS object and return a String data type.

' Create an IsxECLPS object associated with connection A
Dim myPSObj as new IsxECLPS("A")

Dim scrnText as String

' Get text from rectangle on the text plane
scrnText = myPSObj.GetTextRect(3,1,5,10)

---

**WaitForCursor**

The `WaitForCursor` method waits for the cursor in the presentation space of the connection associated with the IsxECLPS object to be located at a specified position.

**Prototype**

Integer WaitForCursor(Long Row, Long Col, [optional] Long TimeOut, [optional] Integer bWaitForIr)

**Parameters**

- **Long Row**: Row position of the cursor.
- **Long Col**: Column position of the cursor.
- **Long TimeOut**: The maximum length of time in milliseconds to wait. This parameter is optional. The default is Infinite.
- **Integer bWaitForIr**: If this value is true, after meeting the wait condition the function will wait until the OIA is ready to accept input. This parameter is optional. The default is 0.

**Return Value**

The method returns 1 if the condition is met, or 0 if the Timeout value is exceeded.

**Example**

Dim IsxECLPSObj as new IsxECLPS("A")
Dim Row, Col as Long

Row = 20
Col = 16

If IsxECLPSObj.WaitForCursor(Row, Col, 10000) Then
    MessageBox("Cursor found")
Else
    MessageBox("Timeout Occurred")
End If

---

**WaitWhileCursor**

The `WaitWhileCursor` method waits while the cursor in the presentation space of the connection associated with the IsxECLPS object is located at a specified position.
Prototype

Integer WaitWhileCursor(Long Row, Long Col, [optional]Long TimeOut, [optional] Integer bWaitForIr)

Parameters

Long Row
Row position of the cursor.

Long Col
Column position of the cursor.

Long TimeOut
The maximum length of time in milliseconds to wait. This parameter is optional. The default is Infinite.

Integer bWaitForIr
If this value is true, after meeting the wait condition the function will wait until the OIA is ready to accept input. This parameter is optional. The default is 0.

Return Value
The method returns 1 if the condition is met, or 0 if the Timeout value is exceeded.

Example
Dim lsxECLPSObj as new lsxECLPS("A")
Dim Row, Col as Long

Row = 20
Col = 16

if (lsxECLPSObj.WaitWhileCursor(Row,Col,10000)) then
   MessageBox( "Wait condition met" )
else
   MessageBox( "Timeout Occurred" )
end if

WaitForString

The WaitForString method waits for the specified string to appear in the presentation space of the connection associated with the lsxECLPS object. If the optional Row and Column parameters are used, the string must begin at the specified position. If 0,0 are passed for Row,Col the method searches the entire PS.

Prototype


Parameters

String WaitString
The string for which to wait.

Long Row
Row position at which the string will begin. This parameter is optional. The default is 0.

Long Col
Column position at which the string will begin. This parameter is optional. The default is 0.

Long TimeOut
The maximum length of time in milliseconds to wait. This parameter is optional. The default is Infinite.
Integer bWaitForIr

If this value is true, after meeting the wait condition the function will wait until the OIA is ready to accept input. This parameter is optional. The default is 0.

Integer bCaseSens

If this value is 1, the wait condition is verified as case-sensitive. This parameter is optional. The default is 0.

Return Value

The method returns 1 if the condition is met, or 0 if the Timeout value is exceeded.

Example

Dim lsxECLPSObj as new lsxECLPS("A")
Dim Row, Col as Long, WaitString

WaitString = "Enter USERID"
Row = 20
Col = 16

if (lsxECLPSObj.WaitForString(WaitString,Row,Col,10000)) then
    MessageBox( "Wait condition met" )
else
    MessageBox( "Timeout Occurred" )
end if

WaitWhileString

The WaitWhileString method waits while the specified string appears in the presentation space of the connection associated with the lsxECLPS object. If the optional Row and Column parameters are used, the string must begin at the specified position. If 0,0 are passed for Row,Col the method searches the entire PS.

Prototype


Parameters

String WaitString

The method waits while this string value exists.

Long Row

Row position at which the string will begin. This parameter is optional. The default is 0.

Long Col

Column position at which the string will begin. This parameter is optional. The default is 0.

Long TimeOut

The maximum length of time in milliseconds to wait. This parameter is optional. The default is Infinite.

Integer bWaitForIr

If this value is true, after meeting the wait condition the function will wait until the OIA is ready to accept input. This parameter is optional. The default is 0.

Integer bCaseSens

If this value is 1, the wait condition is verified as case-sensitive. This parameter is optional. The default is 0.
Return Value
The method returns 1 if the condition is met, or 0 if the Timeout value is exceeded.

Example
Dim lsxECLPSObj as new lsxECLPS("A")
Dim Row, Col as Long
Dim WaitString as String

WaitString = "Enter USERID"
Row = 20
Col = 16

if (lsxECLPSObj.WaitWhileString(WaitString,Row,Col,10000)) then
    MessageBox( "Wait condition met" )
else
    MessageBox( "Timeout Occurred" )
end if

WaitForStringInRect
The WaitForStringInRect method waits for the specified string to appear in the presentation space of the connection associated with the lsxECLPS object in the specified Rectangle.

Prototype
Integer WaitForStringInRect(String WaitString, Long sRow, Long sCol, Long eRow, Long eCol, [optional] Long nTimeOut, [optional] Integer bWaitForIr, [optional] Integer bCaseSens)

Parameters
String WaitString The string to Wait for
Long sRow Starting row position of the search rectangle.
Long sCol Starting column position of the search rectangle.
Long eRow Ending row position of the search rectangle.
Long eCol Ending column position of the search rectangle.
Long TimeOut The maximum length of time in milliseconds to wait. This parameter is optional. The default is Infinite.

Integer bWaitForIr If this value is true, after meeting the wait condition the function will wait until the OIA is ready to accept input. This parameter is optional. The default is 0.

Integer bCaseSens If this value is 1, the wait condition is verified as case-sensitive. This parameter is optional. The default is 0.

Return Value
The method returns 1 if the condition is met, or 0 if the Timeout value is exceeded.

Example
Dim lsxECLPSObj as new lsxECLPS("A")
Dim sRow, sCol, eRow, eCol as Long
Dim WaitString as String
lsxECLPS

WaitString = "Enter USERID"
sRow = 20
sCol = 16
eRow = 21
eCol = 31

if (lsxECLPSObj.WaitForStringInRect(WaitString, sRow, sCol, eRow, eCol, 10000)) then
    MessageBox( "Wait condition met" )
else
    MessageBox( "Timeout Occurred" )
end if

WaitWhileStringInRect

The WaitWhileStringInRect method waits while the specified string appears in the
presentation space of the connection associated with the lsxECLPS object in the
specified Rectangle.

Prototype

Integer WaitWhileStringInRect(String WaitString, Long sRow, Long sCol, Long eRow,
Long eCol, [optional] Long nTimeOut, [optional] Integer bWaitForIr, [optional]
Integer bCaseSens)

Parameters

String WaitString
Long sRow
Long sCol
Long eRow
Long eCol
Long TimeOut

The method waits while this string value exists.
Starting row position of the search rectangle.
Starting column position of the search rectangle.
Ending row position of the search rectangle.
Ending column position of the search rectangle.
The maximum length of time in milliseconds to
wait. This parameter is optional. The default is
Infinite.

Integer bWaitForIr

If this value is true, after meeting the wait
condition the function will wait until the OIA is
ready to accept input. This parameter is optional.
The default is 0.

Integer bCaseSens

If this value is 1, the wait condition is verified as
case-sensitive. This parameter is optional. The
default is 0.

Return Value

The method returns 1 if the condition is met, or 0 if the Timeout value is exceeded.

Example

Dim lsxECLPSObj as new lsxECLPS("A")
Dim sRow, sCol, eRow, eCol as Long
DimWaitString as String

WaitString = "Enter USERID"
sRow = 20
sCol = 16
eRow = 21
eCol = 31

if (lsxECLPSObj.WaitWhileStringInRect(WaitString, sRow, sCol, eRow, eCol, 10000)) then
    MessageBox("Wait condition met")
else
    MessageBox("Timeout Occurred")
end if

**WaitForAttrib**

The WaitForAttrib method will wait until the specified Attribute value appears in the presentation space of the connection associated with the lsxECLPS object at the specified Row/Column position. The optional MaskData parameter can be used to control which values of the attribute you are looking for. The optional plane parameter allows you to select any of the four PS planes.

**Prototype**


**Parameters**

- **Long Row**
  - Row position of the attribute.
- **Long Col**
  - Column position of the attribute.
- **Long WaitData**
  - The 1-byte HEX value of the attribute to wait for.
- **Long MaskData**
  - The 1-byte HEX value to use as a mask with the attribute. This parameter is optional. The default value is 0xFF.
- **Long plane**
  - The plane of the attribute to get. The plane can have the following values:
    - 1: Text Plane
    - 2: Color Plane
    - 3: Field Plane (default)
    - 4: Extended Field Plane
- **Long TimeOut**
  - The maximum length of time in milliseconds to wait. This parameter is optional. The default is Infinite.
- **Integer bWaitForIr**
  - If this value is true, after meeting the wait condition the function will wait until the OIA is ready to accept input. This parameter is optional. The default is 0.

**Return Value**

The method returns 1 if the condition is met, or 0 if the Timeout value is exceeded.

**Example**

```vbscript
Dim lsxECLPSObj as new lsxECLPS("A")
Dim Row, Col, WaitData, MaskData, plane as Long

Row = 20
Col = 16
WaitData = E8h
MaskData = FFh
```
plane = 3

if (lsxECLPSObj.WaitForAttrib(Row, Col, WaitData, MaskData, plane, 10000)) then
    MessageBox( "Wait condition met" )
else
    MessageBox( "Timeout Occurred" )
end if

**WaitWhileAttrib**

The `WaitWhileAttrib` method waits while the specified Attribute value appears in the presentation space of the connection associated with the lsxECLPS object at the specified Row/Column position. The optional MaskData parameter can be used to control which values of the attribute you are looking for. The optional plane parameter allows you to select any of the 4 PS planes.

**Prototype**

```
WaitWhileAttrib(Long Row, Long Col, Long WaitData, [optional] Long MaskData,
    [optional] Long plane, [optional] Long TimeOut, [optional] Integer bWaitForIr)
```

**Parameters**

- **Long Row**
  - Row position of the attribute.
- **Long Col**
  - Column position of the attribute.
- **Long WaitData**
  - The 1-byte HEX value of the attribute to wait for.
- **Long MaskData**
  - The 1-byte HEX value to use as a mask with the attribute. This parameter is optional. The default value is 0xFF.
- **Long plane**
  - The plane of the attribute to get. The plane can have the following values:
    - 1   Text Plane
    - 2   Color Plane
    - 3   Field Plane (default)
    - 4   Extended Field Plane
- **Long TimeOut**
  - The maximum length of time in milliseconds to wait. This parameter is optional. The default is Infinite.
- **Integer bWaitForIr**
  - If this value is true, after meeting the wait condition the function will wait until the OIA is ready to accept input. This parameter is optional. The default is 0.

**Return Value**

The method returns 1 if the condition is met, or 0 if the Timeout value is exceeded.

**Example**

```
Dim lsxECLPSObj as new lsxECLPS("A")
Dim Row, Col, WaitData, MaskData, plane as Long

Row = 20
Col = 16
WaitData = E8h
```
WaitForScreen

Synchronously waits for the screen described by the autECLScreenDesc parameter to appear in the Presentation Space.

Note: The wait for OIA input flag is set on the autECLScreenDesc object. It is not passed as a parameter to the wait method.

Prototype

public Integer WaitForScreen(Object screenDesc, [optional] Long TimeOut)

Parameters

Object screenDesc autECLScreenDesc object that describes the screen (see "autECLScreenDesc Class" on page 261).

Long TimeOut The maximum length of time in milliseconds to wait. This parameter is optional. The default is Infinite.

Return Value

The method returns 1 if the condition is met, or 0 if the Timeout value is exceeded.

Example

Dim lsxECLPSObj as new lsxECLPS("A")
Dim autECLScreenDescObj as new lsxECLScreenDesc()

Set autECLScreenDescObj = CreateObject("PCOMM.autECLScreenDesc")

autECLScreenDesObj.AddCursorPos 23, 1

if (lsxECLPSObj.WaitForScreen(autECLScreenDesObj, 10000)) then
    MessageBox( "Wait condition met" )
else
    MessageBox( "Timeout Occurred" )
end if

WaitWhileScreen

Synchronously waits until the screen described by the autECLScreenDesc parameter is no longer in the Presentation Space.

Note: The wait for OIA input flag is set on the autECLScreenDesc object. It is not passed as a parameter to the wait method.

Prototype

public Integer WaitWhileScreen(Object screenDesc, [optional] Long TimeOut)

Parameters

Object screenDesc autECLScreenDesc object that describes the screen (see "autECLScreenDesc Class" on page 261).
lsxECLPS

**Long TimeOut**

The maximum length of time in milliseconds to wait. This parameter is optional. The default is Infinite.

**Return Value**

The method returns 1 if the condition is met, or 0 if the Timeout value is exceeded.

**Example**

```vbnet
Dim lsxECLPSObj as new lsxECLPS("A")
Dim autECLScreenDescObj as new lsxECLScreenDesc()

autECLScreenDesObj.AddCursorPos 23, 1

if (lsxECLPSObj.WaitWhileScreen(autECLScreenDesObj, 10000)) then
    MessageBox( "Wait condition met" )
else
    MessageBox( "Timeout Occurred" )
end if
```

### lsxECLScreenReco Class

The lsxECLScreenReco class is the engine for the Host Access Class Library screen matching system. It also the logic for matching a given screen to a PS. Because LotusScript does not support asynchronous events, the rich event handling provided in the C++, ActiveX, and Java layers is not supported here. However, the IsMatch() method provided in this class is very useful for determining if the current screen in a lsxECLPS object matches an lsxECLScreenDesc object.

### lsxECLScreenReco Methods

The following method is valid for lsxECLScreenReco:

IsMatch(lsxECLPS ps, lsxECLScreenDesc sd)

#### IsMatch

Allows for passing a lsxECLPS object and a lsxECLScreenDesc object and determining if the screen description matches the PS. The screen recognition engine uses this logic, but is provided so any routine can call it.

**Prototype**

IsMatch(lsxECLPS ps, lsxECLScreenDesc sd)

**Parameters**

- **lsxECLPS ps** 
  - lsxECLPS object to compare.

- **lsxECLScreenDesc sd** 
  - lsxECLScreenDesc object to compare.

**Return Value**

1 if the screen in PS matches, 0 otherwise.

**Example**

```vbnet
Dim lsxECLPSObj as new lsxECLPS("A")
Dim lsxECLScreenDescObj as new lsxECLScreenDesc()

lsxECLScreenDesObj.AddCursorPos(23, 1)
lsxECLScreenDesObj.AddAttrib(E8h, 1, 1, 2)
```
lsxECLScreenDesObj.AddCursorPos(23,1)
lsxECLScreenDesObj.AddNumFields(45)
lsxECLScreenDesObj.AddNumInputFields(17)
lsxECLScreenDesObj.AddOIAInhibitStatus(1)
lsxECLScreenDesObj.AddString("LOGON", 23, 11, 1)
lsxECLScreenDesObj.AddStringInRect("PASSWORD", 23, 1, 24, 80, 0)

if (lsxECLScreenReco.IsMatch(lsxECLPSObj, lsxECLScreenDesObj)) then
    MessageBox("matched")
else
    MessageBox("no match")
end if

IsxECLScreenDesc Class

IsxECLScreenDesc is the class that is used to describe a screen for IBM’s Host Access Class Library Screen Recognition Technology. It uses all four major planes of the presentation space to describe it (text, field, extended field, and color planes), as well as the cursor position.

Using the methods provided on this object, the programmer can set up a detailed description of what a given screen looks like in a host side application. Once an IsxECLScreenDesc object is created and set, it may be passed to the synchronous WaitFor... methods provided on IsxECLPS.

IsxECLScreenDesc Methods

The following section describes the methods that are valid for the IsxECLScreenDesc class.

AddAttrib(Long attrib, Long row, Long col, Long plane)
AddCursorPos(Long row, Long col)
AddNumFields(Long num)
AddNumInputFields(Long num)
AddOIAInhibitStatus(Long type)
AddString(String str, Long row, Long col, [optional] Integer caseSense)
Clear()

AddAttrib

Adds an attribute value at the given position to the screen description.

Prototype
AddAttrib(Long attrib, Long row, Long col, Long plane)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long attrib</td>
<td>The 1-byte HEX value of the attribute.</td>
</tr>
<tr>
<td>Long row</td>
<td>Row position of the attribute.</td>
</tr>
<tr>
<td>Long col</td>
<td>Column position of the attribute.</td>
</tr>
<tr>
<td>Long plane</td>
<td>The plane of the attribute to get. The plane can have the following values:</td>
</tr>
<tr>
<td></td>
<td>1 Text Plane</td>
</tr>
<tr>
<td></td>
<td>2 Color Plane</td>
</tr>
</tbody>
</table>
AddCursorPos

Sets the cursor position for the screen description to the given position.

Prototype

AddCursorPos(Long row, Long col)

Parameters

Long row    Row position of the cursor.
Long col    Column position of the cursor.

Return Value

None

Example

Dim lsxECLPSObj as new lsxECLPS("A")
Dim lsxECLScreenDescObj as new lsxECLScreenDesc()

lsxECLScreenDesObj.AddCursorPos(23, 1)
lsxECLScreenDesObj.AddAttrib(E8h, 1, 1, 2)
lsxECLScreenDesObj.AddCursorPos(23,1)
lsxECLScreenDesObj.AddNumFields(45)
lsxECLScreenDesObj.AddNumInputFields(17)
lsxECLScreenDesObj.AddOIAInhibitStatus(1)
lsxECLScreenDesObj.AddString( "LOGON", 23, 11, 1)
lsxECLScreenDesObj.AddStringInRect( "PASSWORD", 23, 1, 24, 80, 0)

if (lsxECLPSObj.WaitForScreen(lsxECLScreenDesObj, 10000)) then
  MessageBox("Screen reached")
else
  MessageBox("Timeout Occurred")
end if
AddNumFields

Adds the number of fields to the screen description.

Prototype
AddNumFields(Long num)

Parameters
Long num Number of fields.

Return Value
None

Example
Dim lsxECLPSObj as new lsxECLPS("A")
Dim lsxECLScreenDescObj as new lsxECLScreenDesc()

lsxECLScreenDescObj.AddCursorPos(23, 1)
lsxECLScreenDescObj.AddAttrib(E8h, 1, 1, 2)
lsxECLScreenDescObj.AddCursorPos(23, 1)
lsxECLScreenDescObj.AddNumFields(45)
lsxECLScreenDescObj.AddNumInputFields(17)
lsxECLScreenDescObj.AddOIAInhibitStatus(1)
lsxECLScreenDescObj.AddString("LOGON", 23, 11, 1)
lsxECLScreenDescObj.AddStringInRect("PASSWORD", 23, 1, 24, 80, 0)

if (lsxECLPSObj.WaitForScreen(lsxECLScreenDescObj, 10000)) then
    MessageBox(“Screen reached”)
else
    MessageBox(“Timeout Occurred”)
end if

AddNumInputFields

Adds the number of fields to the screen description.

Prototype
AddNumInputFields(Long num)

Parameters
Long num Number of input fields.

Return Value
None

Example
Dim lsxECLPSObj as new lsxECLPS("A")
Dim lsxECLScreenDescObj as new lsxECLScreenDesc()

lsxECLScreenDescObj.AddCursorPos(23, 1)
lsxECLScreenDescObj.AddAttrib(E8h, 1, 1, 2)
lsxECLScreenDescObj.AddCursorPos(23, 1)
lsxECLScreenDescObj.AddNumFields(45)
lsxECLScreenDescObj.AddNumInputFields(17)
lsxECLScreenDescObj.AddOIAInhibitStatus(1)
lsxECLScreenDescObj.AddString("LOGON", 23, 11, 1)
lsxECLScreenDescObj.AddStringInRect("PASSWORD", 23, 1, 24, 80, 0)

if (lsxECLPSObj.WaitForScreen(lsxECLScreenDescObj, 10000)) then
**lsxECLScreenDesc**

```vba
MessageBox("Screen reached")
else
    MessageBox("Timeout Occurred")
end if
```

**AddOIAInhibitStatus**

Sets the type of OIA monitoring for the screen description.

**Prototype**

`AddOIAInhibitStatus(Long type)`

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Long type</code></td>
<td>Type of OIA status. Valid values are as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Don't care</td>
</tr>
<tr>
<td>2</td>
<td>Input not inhibited</td>
</tr>
</tbody>
</table>

**Return Value**

None

**Example**

```vba
Dim lsxECLPSObj as new lsxECLPS("A")
Dim lsxECLScreenDescObj as new lsxECLScreenDesc()

lsxECLScreenDescObj.AddCursorPos(23, 1)
lsxECLScreenDescObj.AddAttrib(E8h, 1, 1, 2)
lsxECLScreenDescObj.AddCursorPos(23, 1)
lsxECLScreenDescObj.AddNumFields(45)
lsxECLScreenDescObj.AddNumInputFields(17)
lsxECLScreenDescObj.AddOIAInhibitStatus(1)
lsxECLScreenDescObj.AddString("LOGON", 23, 11, 1)
lsxECLScreenDescObj.AddStringInRect("PASSWORD", 23, 1, 24, 80, 0)

if (lsxECLPSObj.WaitForScreen(lsxECLScreenDescObj, 10000)) then
    MessageBox("Screen reached")
else
    MessageBox("Timeout Occurred")
end if
```

**AddString**

Adds a string at the given location to the screen description.

**Prototype**

`AddString(String str, Long row, Long col, [optional] Integer caseSense)`

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>String str</code></td>
<td>String to add.</td>
</tr>
<tr>
<td><code>Long row</code></td>
<td>Row position.</td>
</tr>
<tr>
<td><code>Long col</code></td>
<td>Column position.</td>
</tr>
<tr>
<td><code>Integer caseSense</code></td>
<td>If this value is 1, the strings are added as case-sensitive. This parameter is optional. The default is 1.</td>
</tr>
</tbody>
</table>

**Return Value**

None
Example
Dim lsxECLPSObj as new lsxECLPS("A")
Dim lsxECLScreenDescObj as new lsxECLScreenDesc()

lsxECLScreenDesObj.AddCursorPos(23, 1)
lsxECLScreenDesObj.AddAttrib(E8h, 1, 1, 2)
lsxECLScreenDesObj.AddCursorPos(23, 1)
lsxECLScreenDesObj.AddNumFields(45)
lsxECLScreenDesObj.AddNumInputFields(17)
lsxECLScreenDesObj.AddOIAInhibitStatus(1)
lsxECLScreenDesObj.AddString( "LOGON", 23, 11, 1)
lsxECLScreenDesObj.AddStringInRect( "PASSWORD", 23, 1, 24, 80, 0)

if (lsxECLPSObj.WaitForScreen(lsxECLScreenDesObj, 10000)) then
    MessageBox("Screen reached")
else
    MessageBox("Timeout Occurred")
end if

AddStringInRect

Adds a string in the given rectangle to the screen description.

Prototype
AddStringInRect(String str, [optional] Long sRow, [optional] Long sCol,
                 [optional] Long eRow, [optional] Long eCol, [optional] Integer caseSense)

Parameters

String str
    String to add.

Long sRow
    Upper left row position. This parameter is optional. The default is the first row.

Long sCol
    Upper left column position. This parameter is optional. The default is the first column.

Long eRow
    Lower right row position. This parameter is optional. The default is the last row.

Long eCol
    Lower right column position. This parameter is optional. The default is the last column.

Integer caseSense
    If this value is 1, the strings are added as case-sensitive. This parameter is optional. The default is 1.

Return Value

None

Example
Dim lsxECLPSObj as new lsxECLPS("A")
Dim lsxECLScreenDescObj as new lsxECLScreenDesc()

lsxECLScreenDesObj.AddCursorPos(23, 1)
lsxECLScreenDesObj.AddAttrib(E8h, 1, 1, 2)
lsxECLScreenDesObj.AddCursorPos(23, 1)
lsxECLScreenDesObj.AddNumFields(45)
lsxECLScreenDesObj.AddNumInputFields(17)
lsxECLScreenDesObj.AddOIAInhibitStatus(1)
lsxECLScreenDesc

lsxECLScreenDesObj.AddString("LOGON", 23, 11, 1)
lsxECLScreenDesObj.AddStringInRect("PASSWORD", 23, 1, 24, 80, 0)

if (lsxECLPSObj.WaitForScreen(lsxECLScreenDesObj, 10000)) then
    MessageBox("Screen reached")
else
    MessageBox("Timeout Occurred")
end if

Clear

Removes all description elements from the screen description.

Prototype

Clear()

Parameters

None

Return Value

None

Example

Dim lsxECLPSObj as new lsxECLPS("A")
Dim lsxECLScreenDescObj as new lsxECLScreenDesc()

lsxECLScreenDesObj.AddCursorPos(23, 1)
lsxECLScreenDesObj.AddAttrib(E8h, 1, 1, 2)
lsxECLScreenDesObj.AddCursorPos(23, 1)
lsxECLScreenDesObj.AddNumFields(45)
lsxECLScreenDesObj.AddNumInputFields(17)
lsxECLScreenDesObj.AddOIAInhibitStatus(1)
lsxECLScreenDesObj.AddString("LOGON", 23, 11, 1)
lsxECLScreenDesObj.AddStringInRect("PASSWORD", 23, 1, 24, 80, 0)

if (lsxECLPSObj.WaitForScreen(lsxECLScreenDesObj, 10000)) then
    MessageBox("Screen reached")
else
    MessageBox("Timeout Occurred")
end if

lsxECLScreenDesObj.Clear // start over for a new screen

IsxECLSession Class

The lsxECLSession class provides information about a host-connected connection. The lsxECLSession class also contains several other objects that correspond to the various pieces of a host-connected connection.

An lsxECLSession object is associated with a Personal Communications connection when the lsxECLSession object is created. You cannot change the connection that is associated with an lsxECLSession object. If you want to manage a different connection, you must create a new lsxECLSession object associated with that connection.

There are two ways to create an lsxECLSession object:

• Create a new lsxECLSession object by passing a Personal Communications connection name as a parameter on the new statement. A Personal Communications connection name is a single, alphabetic character from A-Z.
The following shows how to create an lsxECLSession object that is associated with Personal Communications connection A:

- Create an lsxECLSession object associated with PCOMM connection A
  
  ```
  dim mySessObj as new lsxECLSession("A")
  ```

- Create a new lsxECLSession object by passing a Personal Communications connection handle as a parameter on the new statement. A Personal Communications connection handle is a Long integer, and is usually obtained by querying the lsxECLConnection object corresponding to the target Personal Communications connection (see "lsxECLConnMgr Class" on page 337 and "lsxECLConnection Class" on page 330 for more information on the properties and methods of those objects). The following example shows how to create an lsxECLSession object using a Personal Communications connection handle:

  ```
  dim mySessObj as lsxECLSession
  dim myConnObj as new lsxECLConnection

  ' Create a new lsxECLSession object using a connection handle
  set mySessObj = new lsxECLSession(myConnObj.Handle)
  ```

When an lsxECLSession object is created, contained lsxECLSession, lsxECLOIA, lsxECLXfer, and lsxECLWinMetrics objects are also created. Refer to them as you would any other property. The following is an example of accessing the lsxECLWinMetrics object within an lsxECLSession object:

- Set the host window to minimized
  ```
  mySessObj.lsxECLWinMetrics.Minimized = 1
  ```

## Properties

This section describes the properties of the lsxECLSession class.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>Name</td>
<td>Read-only</td>
</tr>
<tr>
<td>Long</td>
<td>Handle</td>
<td>Read-only</td>
</tr>
<tr>
<td>String</td>
<td>ConnType</td>
<td>Read-only</td>
</tr>
<tr>
<td>Long</td>
<td>CodePage</td>
<td>Read-only</td>
</tr>
<tr>
<td>Integer</td>
<td>Started</td>
<td>Read-only</td>
</tr>
<tr>
<td>Integer</td>
<td>CommStarted</td>
<td>Read-only</td>
</tr>
<tr>
<td>Integer</td>
<td>APIEnabled</td>
<td>Read-only</td>
</tr>
<tr>
<td>Integer</td>
<td>Ready</td>
<td>Read-only</td>
</tr>
<tr>
<td>lsxECLPS</td>
<td>lsxECLPS</td>
<td>Read-only</td>
</tr>
<tr>
<td>lsxECLOIA</td>
<td>lsxECLOIA</td>
<td>Read-only</td>
</tr>
<tr>
<td>lsxECLXfer</td>
<td>lsxECLXfer</td>
<td>Read-only</td>
</tr>
<tr>
<td>lsxECLWinMetrics</td>
<td>lsxECLWinMetrics</td>
<td>Read-only</td>
</tr>
</tbody>
</table>

## Name

Name is the connection name of the Personal Communications connection associated with this lsxECLSession object. The Name property is a String data type and is read-only. Personal Communications connection names are one character in length and from the character set A-Z. The following example shows this property.

- Create an lsxECLSession object associated with connection A
  ```
  dim mySessObj as new lsxECLSession("A")
  ```
dim myName as String
'm Get our connection name
myName = mySessObj.Name

Handle
Handle is the connection handle of the Personal Communications connection associated with this lsxECLSession object. The Handle property is a Long data type and is read-only. The following example shows this property.
'm Create an lsxECLSession object associated with connection A
dim mySessObj as new lsxECLSession("A")

handle = mySessObj.Handle

ConnType
ConnType is the connection type of the connection that is associated with this lsxECLSession object. The ConnType property is a String data type and is read-only. The following example shows this property.
'm Create an lsxECLSession object associated with connection A
dim mySessObj as new lsxECLSession("A")

myConnType = mySessObj.ConnType

Connection types for the ConnType property are:

<table>
<thead>
<tr>
<th>String Returned</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISP3270</td>
<td>3270 display</td>
</tr>
<tr>
<td>DISP5250</td>
<td>5250 display</td>
</tr>
<tr>
<td>PRNT3270</td>
<td>3270 printer</td>
</tr>
<tr>
<td>PRNT5270</td>
<td>5250 printer</td>
</tr>
<tr>
<td>ASCII</td>
<td>VT emulation</td>
</tr>
<tr>
<td>UNKNOWN</td>
<td>Unknown</td>
</tr>
</tbody>
</table>

CodePage
CodePage is the code page of the connection associated with this lsxECLSession object. The CodePage property is a Long data type, is read-only and cannot be changed through this LotusScript interface. However, the code page of a connection may change if the Personal Communications connection is restarted with a new configuration (see "lsxECLConnMgr Class" on page 337 for information about starting a connection). The following example shows this property.
'm Create an lsxECLSession object associated with connection A
dim mySessObj as new lsxECLSession("A")

codePage = mySessObj.CodePage
**Started**

Started is a Boolean flag that indicates whether the connection associated with this lsxECLSession object is started (for example, still running as a Personal Communications connection). The Started property is an integer and is read-only. Started is 1 if the Personal Communications connection has been started; otherwise, it is 0. The following example shows this property:

'D Create an lsxECLSession object associated with connection A
Dim mySessObj As New lsxECLSession("A")

'D See if our connection is started
If mySessObj.Started Then
    Call connection_started

**CommStarted**

CommStarted is a Boolean flag that indicates whether the connection associated with this lsxECLSession object is connected to the host data stream. The CommStarted property is an integer and is read-only. CommStarted is 1 if there is communication with the host; otherwise, it is 0. The following example shows this property:

'D Create an lsxECLSession object associated with connection A
Dim mySessObj As New lsxECLSession("A")

'D See if we are communicating with the host
If mySessObj.CommStarted Then
    Call communications_started

**APIEnabled**

APIEnabled is a Boolean flag that indicates whether the HLLAPI API has been enabled for the Personal Communications connection associated with this lsxECLSession object. The APIEnabled property is an integer and is read-only. APIEnabled is 1 if the HLLAPI API is available; otherwise, it is 0. The following example shows this property:

'D Create an lsxECLSession object associated with connection A
Dim mySessObj As New lsxECLSession("A")

'D See if the HLLAPI API is enabled on this connection
If mySessObj.APIEnabled Then
    Call hllapi_available

**Ready**

Ready is a Boolean flag that indicates whether the Personal Communications connection associated with this lsxECLSession object is ready. The Ready property is a combination of the Started, CommStarted and APIEnabled properties. It is an integer and is read-only. Ready is 1 if the Started, CommStarted and APIEnabled properties are 1; otherwise, it is 0. The following example shows this property:

'D Create an lsxECLSession object associated with connection A
Dim mySessObj As New lsxECLSession("A")

'D See if our connection is ready
If mySessObj.Ready Then
    Call conn_ready

**lsxECLPS**

This is the lsxECLPS object contained within this lsxECLSession object. Refer to "lsxECLPS Class” on page 355 for a list of the properties and methods of this object. The following example shows this object:

'D Connect to connection A
Dim mySessObj As New lsxECLSession("A")
dim PSSize as Long

' Get the PS size from the contained lsxECLPS object
PSSize = mySessObj.lsxECLPS.Size

**IsxECLIOIA**
This is the lsxECLIOIA object contained within this lsxECLSession object. Refer to "IsxECLIOIA Class" on page 346 for a list of the properties and methods of this object. The following example shows this object.

' Connect to connection A
dim mySessObj as new lsxECLSession("A")

' Check whether we have DBCS on this connection by querying the contained lsxECLIOIA object.
if mySessObj.lsxECLIOIA.DBCS then
call dbcs_enabled

**IsxECLXfer**
This is the lsxECLXfer object contained within this lsxECLSession object. Refer to "lsxECLXfer Class" on page 389 for a list of the properties and methods of this object. The following example shows this object.

' Connect to connection A
dim mySessObj as new lsxECLSession("A")

' Transfer a file to the host using the contained lsxECLXfer object
mySessObj.lsxECLXfer.Sendfile "c:\temp\filename.txt",
"filename text a0",
"CRLF ASCII"

**IsxECLWinMetrics**
This is the lsxECLWinMetrics object contained within this lsxECLSession object. Refer to "lsxECLWinMetrics Class" for a list of the properties and methods of this object. The following example shows this object.

' Connect to connection A
dim mySessObj as new lsxECLSession("A")

' Minimize the host window
mySessObj.lsxECLWinMetrics.Minimized = 1

---

**IsxECLSession Methods**
There are no methods that are valid for the lsxECLSession class.

**lsxECLWinMetrics Class**
The lsxECLWinMetrics class performs operations on a connection window. It allows you to perform window rectangle and position manipulation (for example, SetWindowRect, Ypos or Width), as well as window state manipulation (for example, Visible or Restored).

The lsxECLWinMetrics object is associated with a Personal Communications connection when the lsxECLWinMetrics object is created. You cannot change the connection that is associated with an lsxECLWinMetrics object. If you want to manipulate the window of a different connection, you must create a new lsxECLWinMetrics object associated with that connection.

There are three ways to create an lsxECLWinMetrics object:

- Create a new lsxECLWinMetrics object by passing a Personal Communications connection name as a parameter on the new statement. A Personal...
Communications connection name is a single, alphabetic character from A-Z. The following is an example of creating an lsxECLWinMetrics object that is associated with Personal Communications connection A:

```vba
' Create an lsxECLWinMetrics object associated with PCOMM connection A
dim myWMetObj as new lsxECLWinMetrics("A")
```

- Create a new lsxECLWinMetrics object by passing a Personal Communications connection handle as a parameter on the new statement. A Personal Communications connection handle is a long integer and is usually obtained by querying the lsxECLConnection object corresponding to the target Personal Communications connection (see "lsxECLConnMgr Class" on page 337, "lsxECLConnList Class" on page 334 and "lsxECLConnection Class" on page 330 for more information on the properties and methods of those objects). The following is an example of creating an lsxECLWinMetrics object using a Personal Communications connection handle:

```vba
' Create a new lsxECLWinMetrics object using a connection handle
set myWMetObj = new lsxECLWinMetrics(myConnObj.Handle)
```

- Create an lsxECLSession object and an lsxECLWinMetrics object is automatically created. Access the lsxECLWinMetrics attribute to get to the lsxECLWinMetrics object contained in the lsxECLSession object. The following is an example of accessing the lsxECLWinMetrics object contained in an lsxECLSession object:

```vba
' Create a new lsxECLSession object associated with PCOMM connection A
set mySessObj = new lsxECLSession("A")
' Get the lsxECLWinMetrics object from the lsxECLSession object
set myWMetObj = mySessObj.lsxECLWinMetrics
```

### Properties

This section describes the properties for the lsxECLWinMetrics class.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>WindowTitle</td>
<td>Read-Write</td>
</tr>
<tr>
<td>Long</td>
<td>Xpos</td>
<td>Read-Write</td>
</tr>
<tr>
<td>Long</td>
<td>Ypos</td>
<td>Read-Write</td>
</tr>
<tr>
<td>Long</td>
<td>Width</td>
<td>Read-Write</td>
</tr>
<tr>
<td>Long</td>
<td>Height</td>
<td>Read-Write</td>
</tr>
<tr>
<td>Integer</td>
<td>Visible</td>
<td>Read-Write</td>
</tr>
<tr>
<td>Integer</td>
<td>Active</td>
<td>Read-Write</td>
</tr>
<tr>
<td>Integer</td>
<td>Minimized</td>
<td>Read-Write</td>
</tr>
<tr>
<td>Integer</td>
<td>Maximized</td>
<td>Read-Write</td>
</tr>
<tr>
<td>Integer</td>
<td>Restored</td>
<td>Read-Write</td>
</tr>
<tr>
<td>String</td>
<td>Name</td>
<td>Read-only</td>
</tr>
<tr>
<td>Long</td>
<td>Handle</td>
<td>Read-only</td>
</tr>
<tr>
<td>String</td>
<td>ConnType</td>
<td>Read-only</td>
</tr>
<tr>
<td>Long</td>
<td>CodePage</td>
<td>Read-only</td>
</tr>
<tr>
<td>Integer</td>
<td>Started</td>
<td>Read-only</td>
</tr>
</tbody>
</table>
**WindowTitle**

This is the title that is currently in the title bar for the connection associated with the lsxECLWinMetrics object. The WindowTitle property is a String data type and is read/write enabled.

**Note:** If Window Title is set to blank, the window title of the connection is restored to its original setting.

The following example shows this property.

```vba
' Create an lsxECLWinMetrics object associated with connection A
dim my WMetObj as lsxECLWinMetrics("A")

' Set the window title
my WMetObjWindowTitle = "Main Office"
```

**Xpos**

This is the \( x \) coordinate of the upper left point of the connection's window rectangle. The Xpos property is a Long data type and is read/write enabled. However, if the connection to which you are attached is an inplace, embedded object, this property is read-only. The following example shows this property.

```vba
' Create an lsxECLWinMetrics object associated with connection A
dim my WMetObj as lsxECLWinMetrics("A")

' Set the Xpos of the connection window
my WMetObj.Xpos = 0
```

**Ypos**

This is the \( y \) coordinate of the upper left point of the connection's window rectangle. The Ypos property is a Long data type and is read/write enabled. However, if the connection to which you are attached is an inplace, embedded object, this property is read-only. The following example shows this property.

```vba
' Create an lsxECLWinMetrics object associated with connection A
dim my WMetObj as lsxECLWinMetrics("A")

' Set the Ypos of the connection window
my WMetObj.Ypos = 0
```

**Width**

This is the width of the connection's window rectangle. The Width property is a Long data type and is read/write enabled. However, if the connection to which you are attached is an inplace, embedded object, this property is read-only. The following example shows this property.

```vba
' Create an lsxECLWinMetrics object associated with connection A
dim my WMetObj as lsxECLWinMetrics("A")

' Set the width of the connection window
my WMetObj.Width = 6081
```
**Height**
This is the height of the connection's window rectangle. The Height property is a Long data type and is read/write enabled. However, if the connection to which you are attached is an inplace, embedded object, this property is read-only. The following example shows this property.

```
' Create an IsxECLWinMetrics object associated with connection A
Dim myWMetObj As IsxECLWinMetrics("A")

' Set the height of the connection window
myWMetObj.Height = 6081
```

**Visible**
This is a Boolean value that indicates whether the connection's window is visible. The Visible property is an integer and is read/write enabled. However, if the connection to which you are attached is an inplace, embedded object, this property is read-only. If the connection's window is visible, the Visible property has a value of 1; otherwise, it has a value of 0. The following example shows this property.

```
' Create an IsxECLWinMetrics object associated with connection A
Dim myWMetObj As IsxECLWinMetrics("A")

' Make sure our window is visible
If myWMetObj.Visible = 0 Then
myWMetObj.Visible = 1
```

**Active**
This is a Boolean property that indicates whether the connection's window has the focus. The Active property is an integer and is read/write enabled. However, if the connection to which you are attached is an inplace, embedded object, this property is read-only. If the window has the focus, Active is set to 1; otherwise, is set to 0. The following example shows this property.

```
' Create an IsxECLWinMetrics object associated with connection A
Dim myWMetObj As IsxECLWinMetrics("A")

' Make sure our window has the focus
If myWMetObj.Active = 0 Then
myWMetObj.Active = 1
```

**Minimized**
Minimized is a Boolean property that indicates whether the connection's window is minimized. The Minimized property is an integer and is read/write enabled. However, if the connection to which you are attached is an inplace, embedded object, this property is read-only. If the connection's window is minimized, the Minimized property is set to 1; otherwise, it is set to 0. The following example shows this property.

```
' Create an IsxECLWinMetrics object associated with connection A
Dim myWMetObj As IsxECLWinMetrics("A")

' Make sure our window isn't minimized
If myWMetObj.Minimized Then
myWMetObj.Minimized = 0
```

**Maximized**
Maximized is a Boolean property that indicates whether the connection's window is maximized. The Maximized property is an integer and is read/write enabled. However, if the connection to which you are attached is an inplace, embedded object, this property is read-only. If the connection's window is maximized, the Maximized property is set to 1; otherwise, it is set to 0. The following example shows this property.
Create an `lsxECLWinMetrics` object associated with connection A

Dim myWMetObj As lsxECLWinMetrics("A")

Make sure our window is maximized
If myWMetObj.Maximized = 0 Then
    myWMetObj.Maximized = 1
End If

Restored
This is a Boolean property that indicates whether the connection's window is in a restored state. The Restored property is an integer and is read/write enabled. However, if the connection to which you are attached is an inplace, embedded object, this property is read-only. If the connection's window is in a restored state, the Restored property is set to 1; otherwise, it is set to 0. The following example shows this property.

Create an `lsxECLWinMetrics` object associated with connection A
Dim myWMetObj As lsxECLWinMetrics("A")

Make sure we're in a restored state
If myWMetObj.Restored = 0 Then
    myWMetObj.Restored = 1
End If

Name
Name is the connection name of the Personal Communications connection associated with this `lsxECLWinMetrics` object. The Name property is a String data type and is read-only. Personal Communications connection names are one character in length and from the character set A-Z. The following example shows this property.

Create an `lsxECLWinMetrics` object associated with connection A
Dim myWMetObj As new lsxECLWinMetrics("A")

Dim myName As String

' Get our connection name
myName = myWMetObj.Name

Handle
Handle is the connection handle of the Personal Communications connection associated with this `lsxECLWinMetrics` object. The Handle property is a Long data type and is read-only. The following example shows this property.

Create an `lsxECLWinMetrics` object associated with connection A
Dim myWMetObj As new lsxECLWinMetrics("A")

Dim myHandle As Long

' Get our connection handle
myHandle = myWMetObj.Handle

ConnType
The connection type of the connection that is associated with this `lsxECLWinMetrics` object. The ConnType property is a String data type and is read-only. The following example shows this property.

Create an `lsxECLWinMetrics` object associated with connection A
Dim myWMetObj As new lsxECLWinMetrics("A")

Dim myConnType As String

' Get the connection type for connection A
myConnType = myWMetObj.ConnType
Connection types for the ConnType property are:

<table>
<thead>
<tr>
<th>String Returned</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISP3270</td>
<td>3270 display</td>
</tr>
<tr>
<td>DISP5250</td>
<td>5250 display</td>
</tr>
<tr>
<td>PRNT3270</td>
<td>3270 printer</td>
</tr>
<tr>
<td>PRNT5250</td>
<td>5250 printer</td>
</tr>
<tr>
<td>ASCII</td>
<td>VT emulation</td>
</tr>
<tr>
<td>UNKNOWN</td>
<td>Unknown</td>
</tr>
</tbody>
</table>

**CodePage**

CodePage is the code page of the connection associated with this IsxECLWinMetrics object. The CodePage property is a Long data type, is read-only and cannot be changed through this LotusScript interface. However, the code page of a connection may change if the Personal Communications connection is restarted with a new configuration. See “lsxECLConnMgr Class” on page 337 for information about starting a connection. The following example shows this property.

```
' Create an IsxECLWinMetrics object associated with connection A
dim myWMetObj as new IsxECLWinMetrics("A")

dim myCodePage as Long

' Get the code page for connection A
myCodePage = myWMetObj.CodePage
```

**Started**

Started is a Boolean flag that indicates whether the connection associated with this IsxECLWinMetrics object is started (for example, still running as a Personal Communications connection). This property is an integer and is read-only. The Started property is 1 if the Personal Communications connection has been started; otherwise, it is 0. The following example shows this property.

```
' Create an IsxECLWinMetrics object associated with connection A
dim myWMetObj as new IsxECLWinMetrics("A")

' See if our connection is started
if myWMetObj.Started then
  call connection_started
```

**CommStarted**

CommStarted is a Boolean flag that indicates whether the connection associated with this IsxECLWinMetrics object is connected to the host data stream. The CommStarted property is an integer and is read-only. CommStarted is 1 if there is communication with the host; otherwise, it is 0. The following example shows this property.

```
' Create an IsxECLWinMetrics object associated with connection A
dim myWMetObj as new IsxECLWinMetrics("A")

' See if we are communicating with the host
if myWMetObj.CommStarted then
  call communications_started
```

**APIEnabled**

APIEnabled is a Boolean flag that indicates whether the HLLAPI API has been enabled for the Personal Communications connection associated with this
lsxECLWinMetrics

lsxECLWinMetrics object. The APIEnabled property is an integer and is read-only. APIEnabled is 1 if the HLLAPI API is available; otherwise, it is 0. The following example shows this property.

' Create an lsxECLWinMetrics object associated with connection A
dim myWMetObj as new lsxECLWinMetrics("A")

' See if the HLLAPI API is enabled on this connection
if myWMetObj.APIEnabled then
    call hllapi_available

Ready

Ready is a Boolean flag that indicates whether the Personal Communications connection associated with this lsxECLWinMetrics object is ready. The Ready property is a combination of the Started, CommStarted and APIEnabled properties. It is an integer and is read-only. Ready is 1 if the Started, CommStarted and APIEnabled properties are 1; otherwise, it is 0. The following example shows this property.

' Create an lsxECLWinMetrics object associated with connection A
dim myWMetObj as new lsxECLWinMetrics("A")

' See if our connection is ready
if myWMetObj.Ready then
    call conn_ready

lsxECLWinMetrics Methods

The following section describes the methods that are valid for lsxECLWinMetrics.

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

GetWindowRect

This method returns the coordinates of the top, bottom, left and right sides of the window rectangle associated with this connection. The supplied parameters are set to the coordinates of the window rectangle.

Prototype
GetWindowRect(Long left, Long top, Long right, Long bottom)

Parameters

Long left
The coordinate of the left side of the window rectangle.

Long top
The coordinate of the top of the window rectangle.

Long right
The coordinate of the right side of the window rectangle.

Long bottom
The coordinate of the bottom of the window rectangle.

Return Value
None

Example

The following example shows how to return the coordinates of the top, bottom, left and right sides of the window rectangle associated with this connection.
Create an lsxECLWinMetrics object associated with connection A

`dim myWMetObj as new lsxECLWinMetrics("A")`

Create some variables to hold our window coordinates

```vbnet
dim left as Long
dim top as Long
dim right as Long
dim bottom as Long
```

Get the window coordinates

```vbnet
myWMetObj.GetWindowRect left, top, right, bottom
```

SetWindowRect

This method sets the coordinates of the top, bottom, left and right sides of the window rectangle associated with this connection.

**Prototype**

`SetWindowRect(Long left, Long top, Long right, Long bottom)`

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long left</td>
<td>The new coordinate of the left side of the window rectangle.</td>
</tr>
<tr>
<td>Long top</td>
<td>The new coordinate of the top of the window rectangle.</td>
</tr>
<tr>
<td>Long right</td>
<td>The new coordinate of the right side of the window rectangle.</td>
</tr>
<tr>
<td>Long bottom</td>
<td>The new coordinate of the bottom of the window rectangle.</td>
</tr>
</tbody>
</table>

**Return Value**

None

**Example**

The following example shows how to set the coordinates of the top, bottom, left and right sides of the window rectangle associated with this connection.

```
' Create an lsxECLWinMetrics object associated with connection A
dim myWMetObj as new lsxECLWinMetrics("A")

' Set the window coordinates
myWMetObj.SetWindowRect 0, 0, 6081, 6081
```

---

**lsxECLXfer Class**

The lsxECLXfer Class provides file transfer services between a host and a client. The transfer is done through a Personal Communications connection and therefore, the lsxECLXfer object must be associated with a Personal Communications connection.

The lsxECLXfer object is associated with a Personal Communications connection when the lsxECLXfer object is created. You cannot change the connection that is associated with an lsxECLXfer object. If you want to transfer files on a different connection, you must create a new lsxECLXfer object associated with that connection.

There are three ways to create an lsxECLXfer object:
Create a new lsxECLXfer object by passing a Personal Communications connection name as a parameter on the new statement. A Personal Communications connection name is a single, alphabetic character from A-Z. The following is an example of creating an lsxECLXfer object that is associated with Personal Communications connection A:

' Create an lsxECLXfer object associated with PCOMM connection A
dim myXferObj as new lsxECLXfer("A")

Create a new lsxECLXfer object by passing a Personal Communications connection handle as a parameter on the new statement. A Personal Communications connection handle is a long integer and is usually obtained by querying the lsxECLConnection object corresponding to the target Personal Communications connection (see "lsxECLConnMgr Class" on page 337, "lsxECLConnList Class" on page 334 and "lsxECLConnection Class" on page 330 for more information on the properties and methods of those objects). The following is an example of creating an lsxECLXfer object using a Personal Communications connection handle:

dim myXferObj as lsxECLXfer
dim myConnObj as new lsxECLConnection

' Create a new lsxECLXfer object using the connection handle
set myXferObj = new lsxECLXfer(myConnObj.Handle)

Create an lsxECLSession object and an lsxECLXfer object is automatically created. Access the lsxECLXfer attribute to get to the lsxECLXfer object contained in the lsxECLSession object. The following is an example of how to access the lsxECLXfer object contained in an lsxECLSession object:

dim myXferObj as lsxECLXfer
dim lsxECLSessionObj as lsxECLSession

' Create a new lsxECLSession object associated with PCOMM connection A
set lsxECLSessionObj = new lsxECLSession("A")
' Get the lsxECLXfer object from the lsxECLSession object
set myXferObj = lsxECLSessionObj.lsxECLXfer

Properties

This section describes the properties of the lsxECLXfer class.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long</td>
<td>Name</td>
<td>Read-only</td>
</tr>
<tr>
<td>Long</td>
<td>Handle</td>
<td>Read-only</td>
</tr>
<tr>
<td>String</td>
<td>ConnType</td>
<td>Read-only</td>
</tr>
<tr>
<td>Long</td>
<td>CodePage</td>
<td>Read-only</td>
</tr>
<tr>
<td>Integer</td>
<td>Started</td>
<td>Read-only</td>
</tr>
<tr>
<td>Integer</td>
<td>CommStarted</td>
<td>Read-only</td>
</tr>
<tr>
<td>Integer</td>
<td>APIEnabled</td>
<td>Read-only</td>
</tr>
<tr>
<td>Integer</td>
<td>Ready</td>
<td>Read-only</td>
</tr>
</tbody>
</table>

Name

Name is the connection name of the Personal Communications connection associated with this lsxECLXfer object. The Name property is a String data type and is read-only. Personal Communications connection names are one character in length and from the character set A-Z. The following example shows this property.
' Create an lsxECLXfer object associated with connection A
dim myXferObj as new lsxECLXfer("A")

dim myName as String

' Get our connection name
myName = myXferObj.Name

**Handle**

Handle is the connection handle of the Personal Communications connection associated with this lsxECLXfer object. The Handle property is a Long data type and is read-only. The following example shows this property.

' Create an lsxECLXfer object associated with connection A
dim myXferObj as new lsxECLXfer("A")

dim myHandle as Long

' Get our connection handle
myHandle = myXferObj.Handle

**ConnType**

ConnType is the connection type of the connection that is associated with this lsxECLXfer object. The ConnType property is a String data type and is read-only. The following example shows this property.

' Create an lsxECLXfer object associated with connection A
dim myXferObj as new lsxECLXfer("A")

dim myConnType as String

' Get the connection type for connection A
myConnType = myXferObj.ConnType

Connection types for the ConnType property are:

<table>
<thead>
<tr>
<th>String Returned</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISP3270</td>
<td>3270 display</td>
</tr>
<tr>
<td>DISP5270</td>
<td>5250 display</td>
</tr>
<tr>
<td>PRNT3270</td>
<td>3270 printer</td>
</tr>
<tr>
<td>PRNT5270</td>
<td>5250 printer</td>
</tr>
<tr>
<td>ASCII</td>
<td>VT emulation</td>
</tr>
<tr>
<td>UNKNOWN</td>
<td>Unknown</td>
</tr>
</tbody>
</table>

**CodePage**

CodePage is the code page of the connection associated with this lsxECLXfer object. The CodePage property is a Long data type, is read-only and cannot be changed through this LotusScript interface. However, the code page of a connection may change if the Personal Communications connection is restarted with a new configuration (see "lsxECLConnMgr Class" on page 337 for information about starting a connection). The following example shows this property.

' Create an lsxECLXfer object associated with connection A
dim myXferObj as new lsxECLXfer("A")

dim myCodePage as Long

' Get the code page for connection A
myCodePage = myXferObj.CodePage
Started
Started is a Boolean flag that indicates whether the connection associated with this lsxECLXfer object is started (for example, still running as a Personal Communications connection). The Started property is an integer and is read-only. Started is 1 if the Personal Communications connection has been started; otherwise, it is 0. The following example shows this property.

' Create an lsxECLXfer object associated with connection A
  dim myXferObj as new lsxECLXfer("A")

  ' See if our connection is started
  if myXferObj.Started then
    call connection_started

CommStarted
CommStarted is a Boolean flag that indicates whether the connection associated with this lsxECLXfer object is connected to the host data stream. The CommStarted property is an integer and is read-only. CommStarted is 1 if there is communication with the host; otherwise, it is 0. The following example shows this property.

' Create an lsxECLXfer object associated with connection A
  dim myXferObj as new lsxECLXfer("A")

  ' See if we are communicating with the host
  if myXferObj.CommStarted then
    call communications_started

APIEnabled
APIEnabled is a Boolean flag that indicates whether the HLLAPI API has been enabled for the Personal Communications connection associated with this lsxECLXfer object. The APIEnabled property is an integer and is read-only. APIEnabled is 1 if the HLLAPI API is available; otherwise, it is 0. The following example shows this property.

' Create an lsxECLXfer object associated with connection A
  dim myXferObj as new lsxECLXfer("A")

  ' See if the HLLAPI API is enabled on this connection
  if myXferObj.APIEnabled then
    call hllapi_available

Ready
Ready is a Boolean flag that indicates whether the Personal Communications connection associated with this lsxECLXfer object is ready. The Ready property is a combination of the Started, CommStarted and APIEnabled properties. It is an integer and is read-only. Ready is 1 if the Started, CommStarted and APIEnabled properties are 1; otherwise, it is 0. The following example shows this property.

' Create an lsxECLXfer object associated with connection A
  dim myXferObj as new lsxECLXfer("A")

  ' See if our connection is ready
  if myXferObj.Ready then
    call conn_ready

lsxECLXfer Methods
The following section describes the methods that are valid for the lsxECLXfer class.

SendFile (String PCFile, String HostFile, String Options)
ReceiveFile (String PCFile, String HostFile, String Options)
SendFile

This method sends a file from the workstation to the host.

Prototype
SendFile( String PCFile, String HostFile, String Options )

Parameters
String PCFile  Name of the source file on the workstation.
String HostFile Name of the target file on the host.
String Options  File transfer options (see Usage Notes).

Return Value
None

Example
The following example shows how to send a file from the workstation to the host.

' Create a new IsxECLXfer object associated with connection A
dim myXferObj as new IsxECLXfer("A")

' Send a file from my PC to the host on connection A,
' Assume the host is a VM/CMS host
myXferObj.SendFile "c:\windows\temp\thefile.txt",
    "THEFILE TEXT A",
    "(CRLF ASCII"

Usage Notes
File transfer options are host-dependent. The following is a list of some of the
valid host options for a VM/CMS host:
    ASCII
    JISCII
    CRLF
    APPEND
    TIME n
    CLEAR
    NOCLEAR
    PROGRESS
    QUIET

Refer to Emulator Programming for the list of supported hosts and associated file
transfer options.

ReceiveFile

This method receives a file from the host to the workstation.

Prototype
ReceiveFile( String PCFile, String HostFile, String Options )

Parameters
String PCFile  Name of the file on the workstation.
String HostFile Name of the file on the host.
String Options  File transfer options (see Usage Notes).

Return Value
None
Example
The following example shows how to receive a file from the host to the workstation.

' Create a new lsxECLXfer object associated with connection A
  dim myXferObj as new lsxECLXfer("A")
'
  Receive a file from host connection A onto my workstation,
  ' Assume the host is a VM/CMS host
  myXferObj.ReceiveFile "c:\windows\temp\thefile.txt",
  "THEFILE TEXT A0",
  "(CRLF ASCII"

Usage Notes
File transfer options are host-dependent. For example, a list of some of the valid host options for a VM/CMS host are:

ASCII
JISCI
CRLF
APPEND
TIME n
CLEAR
NOCLEAR
PROGRESS
QUIET

Refer to Emulator Programming for the list of supported hosts and associated file transfer options.
Chapter 5. Host Access Class Library for Java

The Host Access Class Library (HACL) Java classes expose the Personal Communications HACL functions to the Java programming environment. This allows the creation of Java applets and applications that utilize the functions provided in the HACL classes.

The documentation for the HACL Java classes is provided in HTML format. These files are available in the ..\doc\hac1 subdirectory of the Personal Communications installation image. To view the documentation, use a Web browser to view the ECLReference.html file, which is the first file of the softcopy HACL Java reference.
Appendix A. Sendkeys Mnemonic Keywords

Table 3 contains the mnemonic keywords for the Sendkeys method.

Table 3. Mnemonic Keywords for the Sendkeys Method

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[backtab]</td>
<td>Back tab</td>
</tr>
<tr>
<td>[clear]</td>
<td>Clear screen</td>
</tr>
<tr>
<td>[delete]</td>
<td>Delete</td>
</tr>
<tr>
<td>[enter]</td>
<td>Enter</td>
</tr>
<tr>
<td>[eraseof]</td>
<td>Erase end of field</td>
</tr>
<tr>
<td>[help]</td>
<td>Help</td>
</tr>
<tr>
<td>[insert]</td>
<td>Insert</td>
</tr>
<tr>
<td>[jump]</td>
<td>Jump</td>
</tr>
<tr>
<td>[left]</td>
<td>Left</td>
</tr>
<tr>
<td>[newline]</td>
<td>New line</td>
</tr>
<tr>
<td>[space]</td>
<td>Space</td>
</tr>
<tr>
<td>[print]</td>
<td>Print</td>
</tr>
<tr>
<td>[reset]</td>
<td>Reset</td>
</tr>
<tr>
<td>[tab]</td>
<td>Tab</td>
</tr>
<tr>
<td>[up]</td>
<td>Up</td>
</tr>
<tr>
<td>[Down]</td>
<td>Down</td>
</tr>
<tr>
<td>[dbcs]</td>
<td>DBCS</td>
</tr>
<tr>
<td>[capslock]</td>
<td>CapsLock</td>
</tr>
<tr>
<td>[right]</td>
<td>Right</td>
</tr>
<tr>
<td>[home]</td>
<td>Home</td>
</tr>
<tr>
<td>[pf1]</td>
<td>PF2</td>
</tr>
<tr>
<td>[pf2]</td>
<td>PF2</td>
</tr>
<tr>
<td>[pf3]</td>
<td>PF3</td>
</tr>
<tr>
<td>[pf4]</td>
<td>PF4</td>
</tr>
<tr>
<td>[pf5]</td>
<td>PF5</td>
</tr>
<tr>
<td>[pf6]</td>
<td>PF6</td>
</tr>
<tr>
<td>[pf7]</td>
<td>PF7</td>
</tr>
<tr>
<td>[pf8]</td>
<td>PF8</td>
</tr>
<tr>
<td>[pf9]</td>
<td>PF9</td>
</tr>
<tr>
<td>[pf10]</td>
<td>PF10</td>
</tr>
<tr>
<td>[pf11]</td>
<td>PF11</td>
</tr>
<tr>
<td>[pf12]</td>
<td>PF12</td>
</tr>
<tr>
<td>[pf13]</td>
<td>PF13</td>
</tr>
<tr>
<td>[pf14]</td>
<td>PF14</td>
</tr>
<tr>
<td>[pf15]</td>
<td>PF15</td>
</tr>
</tbody>
</table>
### Table 3. Mnemonic Keywords for the Sendkeys Method (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[pf16]</td>
<td>PF16</td>
</tr>
<tr>
<td>[pf17]</td>
<td>PF17</td>
</tr>
<tr>
<td>[pf18]</td>
<td>PF18</td>
</tr>
<tr>
<td>[pf19]</td>
<td>PF19</td>
</tr>
<tr>
<td>[pf20]</td>
<td>PF20</td>
</tr>
<tr>
<td>[pf21]</td>
<td>PF21</td>
</tr>
<tr>
<td>[pf22]</td>
<td>PF22</td>
</tr>
<tr>
<td>[pf23]</td>
<td>PF23</td>
</tr>
<tr>
<td>[pf24]</td>
<td>PF24</td>
</tr>
<tr>
<td>[eof]</td>
<td>End of file</td>
</tr>
<tr>
<td>[scroll]</td>
<td>Scroll Lock</td>
</tr>
<tr>
<td>[numlock]</td>
<td>Num Lock</td>
</tr>
<tr>
<td>[pageup]</td>
<td>Page Up</td>
</tr>
<tr>
<td>[pagedn]</td>
<td>Page Down</td>
</tr>
<tr>
<td>[pa1]</td>
<td>PA 1</td>
</tr>
<tr>
<td>[pa2]</td>
<td>PA 2</td>
</tr>
<tr>
<td>[pa3]</td>
<td>PA 3</td>
</tr>
<tr>
<td>[test]</td>
<td>Test</td>
</tr>
<tr>
<td>[worddel]</td>
<td>Word Delete</td>
</tr>
<tr>
<td>[fldext]</td>
<td>Field Exit</td>
</tr>
<tr>
<td>[erinp]</td>
<td>Erase Input</td>
</tr>
<tr>
<td>[sysreq]</td>
<td>System Request</td>
</tr>
<tr>
<td>[instog]</td>
<td>Insert Toggle</td>
</tr>
<tr>
<td>[crsel]</td>
<td>Cursor Select</td>
</tr>
<tr>
<td>[fastleft]</td>
<td>Cursor Left Fast</td>
</tr>
<tr>
<td>[attn]</td>
<td>Attention</td>
</tr>
<tr>
<td>[devcance]</td>
<td>Device Cancel</td>
</tr>
<tr>
<td>[printps]</td>
<td>Print Presentation Space</td>
</tr>
<tr>
<td>[fastup]</td>
<td>Cursor Up Fast</td>
</tr>
<tr>
<td>[fastdown]</td>
<td>Cursor Down Fast</td>
</tr>
<tr>
<td>[hex]</td>
<td>Hex</td>
</tr>
<tr>
<td>[fastright]</td>
<td>Cursor Right Fast</td>
</tr>
<tr>
<td>[revvideo]</td>
<td>Reverse Video</td>
</tr>
<tr>
<td>[underscr]</td>
<td>Underscore</td>
</tr>
<tr>
<td>[rstvideo]</td>
<td>Reset Reverse Video</td>
</tr>
<tr>
<td>[red]</td>
<td>Red</td>
</tr>
<tr>
<td>[pink]</td>
<td>Pink</td>
</tr>
<tr>
<td>[green]</td>
<td>Green</td>
</tr>
<tr>
<td>[yellow]</td>
<td>Yellow</td>
</tr>
<tr>
<td>[blue]</td>
<td>Blue</td>
</tr>
</tbody>
</table>
Table 3. Mnemonic Keywords for the Sendkeys Method (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[turq]</td>
<td>Turquoise</td>
</tr>
<tr>
<td>[white]</td>
<td>White</td>
</tr>
<tr>
<td>[rstcolor]</td>
<td>Reset Host Color</td>
</tr>
<tr>
<td>[printpc]</td>
<td>Print (PC)</td>
</tr>
<tr>
<td>[wordright]</td>
<td>Forward Word Tab</td>
</tr>
<tr>
<td>[wordleft]</td>
<td>Backward Word Tab</td>
</tr>
<tr>
<td>[field-]</td>
<td>Field -</td>
</tr>
<tr>
<td>[field+]</td>
<td>Field +</td>
</tr>
<tr>
<td>[rcdbacksp]</td>
<td>Record Backspace</td>
</tr>
<tr>
<td>[printhost]</td>
<td>Print Presentation Space on Host</td>
</tr>
<tr>
<td>[dup]</td>
<td>Dup</td>
</tr>
<tr>
<td>[fieldmark]</td>
<td>Field Mark</td>
</tr>
<tr>
<td>[dispsoSI]</td>
<td>Display SO/SI</td>
</tr>
<tr>
<td>[gensosi]</td>
<td>Generate SO/SI</td>
</tr>
<tr>
<td>[dispatr]</td>
<td>Display Attribute</td>
</tr>
<tr>
<td>[fwdchar]</td>
<td>Forward Character</td>
</tr>
<tr>
<td>[splitbar]</td>
<td>Split Vertical Bar</td>
</tr>
<tr>
<td>[altcsrc]</td>
<td>Alternate Cursor</td>
</tr>
<tr>
<td>[backspace]</td>
<td>Backspace</td>
</tr>
<tr>
<td>[null]</td>
<td>Null</td>
</tr>
</tbody>
</table>
Appendix B. ECL Planes — Format and Content

This appendix describes the format and contents of the different data planes in the ECL presentation space model. Each plane represents a distinct aspect of the host presentation space, such as its character contents, color specifications, field attributes, and so on. The ECL::GetScreen methods and others return data from the different presentation space planes.

Each plane contains one byte per host presentation space character position. Each plane is described in the following sections in terms of its logical contents and data format. The plane types are enumerated in the ECLPS.HPP header file.

**TextPlane**

The text plane represents the visible characters of the presentation space. Non-display fields are shown in the text plane. The byte value of each element of the text plane corresponds to the ASCII value of the displayed character. The text plane does not contain any binary zero (null) character values. Any null characters in the presentation space (such as null-padded input fields) are represented as ASCII blank (0x20) characters.

**FieldPlane**

The field plane represents the field positions and their attributes in the presentation space. This plane is meaningful only for field-formatted presentation spaces. (For example, VT connections are not formatted).

This plane is a sparse-array of field attribute values. All values in this plane are binary zero except for where field attribute characters are present in the presentation space. At those positions, the values are the attributes of the field which starts at that location. The length of a field is the linear distance between the field attribute position and the next field attribute in the presentation space, not including the attribute position itself.

The value of the field attribute positions are as shown in the following tables.

**Note:** Attribute values are different for different types of connections.

*Table 4. 3270 Field Attributes*

<table>
<thead>
<tr>
<th>Bit Position (0 is least significant bit)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Always &quot;1&quot;</td>
</tr>
<tr>
<td>6</td>
<td>Always &quot;1&quot;</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
Table 4. 3270 Field Attributes (continued)

<table>
<thead>
<tr>
<th>Bit Position (0 is least significant bit)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>3, 2</td>
<td>0, 0  Normal intensity, not pen detectable</td>
</tr>
<tr>
<td></td>
<td>0, 1  Normal intensity, pen detectable</td>
</tr>
<tr>
<td></td>
<td>1, 0  High intensity, pen detectable</td>
</tr>
<tr>
<td></td>
<td>1, 1  Nondisplay, not pen detectable</td>
</tr>
<tr>
<td>1</td>
<td>Reserved</td>
</tr>
<tr>
<td>0</td>
<td>0    Field has not been modified</td>
</tr>
<tr>
<td></td>
<td>1    Unprotected field has been modified</td>
</tr>
</tbody>
</table>

Table 5. 5250 Field Attributes

<table>
<thead>
<tr>
<th>Bit Position (0 is least significant bit)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Always “1”</td>
</tr>
<tr>
<td>6</td>
<td>0    Nondisplay</td>
</tr>
<tr>
<td></td>
<td>1    Display</td>
</tr>
<tr>
<td>5</td>
<td>0    Unprotected</td>
</tr>
<tr>
<td></td>
<td>1    Protected</td>
</tr>
<tr>
<td>4</td>
<td>0    Normal intensity</td>
</tr>
<tr>
<td></td>
<td>1    High intensity</td>
</tr>
<tr>
<td>3, 2, 1</td>
<td>0, 0, 0  Alphanumeric data</td>
</tr>
<tr>
<td></td>
<td>0, 0, 1  Alpha only</td>
</tr>
<tr>
<td></td>
<td>0, 1, 0  Numeric shift</td>
</tr>
<tr>
<td></td>
<td>0, 1, 1  Numeric data plus numeric specials</td>
</tr>
<tr>
<td></td>
<td>1, 0, 1  Numeric only</td>
</tr>
<tr>
<td></td>
<td>1, 1, 0  Magnetic stripe reading device data only</td>
</tr>
<tr>
<td></td>
<td>1, 1, 1  Signed numeric only</td>
</tr>
<tr>
<td>0</td>
<td>0    Field has not been modified</td>
</tr>
<tr>
<td></td>
<td>1    Unprotected field has been modified</td>
</tr>
</tbody>
</table>

Table 6 defines the various mask values:

Table 6. Mask Values

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Mask</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FATTR_MDT</td>
<td>0x01</td>
<td>Modified field</td>
</tr>
<tr>
<td>FATTR_PEN_MASK</td>
<td>0x0C</td>
<td>Pen detectable field</td>
</tr>
<tr>
<td>FATTR_BRIGHT</td>
<td>0x08</td>
<td>Intensified field</td>
</tr>
<tr>
<td>FATTR_DISPLAY</td>
<td>0x0C</td>
<td>Visible field</td>
</tr>
<tr>
<td>FATTR_ALPHA</td>
<td>0x10</td>
<td>Alphanumeric field</td>
</tr>
</tbody>
</table>
Table 6. Mask Values (continued)

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Mask</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FATTR_NUMERIC</td>
<td>0x10</td>
<td>Numeric only field</td>
</tr>
<tr>
<td>FATTR_PROTECTED</td>
<td>0x20</td>
<td>Protected field</td>
</tr>
<tr>
<td>FATTR_PRESENT</td>
<td>0x80</td>
<td>Field attribute present</td>
</tr>
<tr>
<td>FATTR_52_BRIGHT</td>
<td>0x10</td>
<td>5250 intensified field</td>
</tr>
<tr>
<td>FATTR_52_DISP</td>
<td>0x40</td>
<td>5250 visible field</td>
</tr>
</tbody>
</table>

ColorPlane

The color plane contains color information for each character of the presentation space. The foreground and background color of each character is represented as it is specified in the host data stream. The colors in the color plane are not modified by any color display mapping of the emulator window. Each byte of the color plane contains the following color information.

Table 7. Color Plane Information

<table>
<thead>
<tr>
<th>Bit Position (0 is least significant bit)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 - 4</td>
<td>Background character color</td>
</tr>
<tr>
<td>0x0</td>
<td>Blank</td>
</tr>
<tr>
<td>0x1</td>
<td>Blue</td>
</tr>
<tr>
<td>0x2</td>
<td>Green</td>
</tr>
<tr>
<td>0x3</td>
<td>Cyan</td>
</tr>
<tr>
<td>0x4</td>
<td>Red</td>
</tr>
<tr>
<td>0x5</td>
<td>Magenta</td>
</tr>
<tr>
<td>0x6</td>
<td>Brown (3270), Yellow (5250)</td>
</tr>
<tr>
<td>0x7</td>
<td>White</td>
</tr>
</tbody>
</table>
Table 7. Color Plane Information (continued)

<table>
<thead>
<tr>
<th>Bit Position (0 is least significant bit)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-0</td>
<td>Foreground character color</td>
</tr>
<tr>
<td>0x0</td>
<td>Blank</td>
</tr>
<tr>
<td>0x1</td>
<td>Blue</td>
</tr>
<tr>
<td>0x2</td>
<td>Green</td>
</tr>
<tr>
<td>0x3</td>
<td>Cyan</td>
</tr>
<tr>
<td>0x4</td>
<td>Red</td>
</tr>
<tr>
<td>0x5</td>
<td>Magenta</td>
</tr>
<tr>
<td>0x6</td>
<td>Brown (3270), Yellow (5250)</td>
</tr>
<tr>
<td>0x7</td>
<td>White (normal intensity)</td>
</tr>
<tr>
<td>0x8</td>
<td>Gray</td>
</tr>
<tr>
<td>0x9</td>
<td>Light blue</td>
</tr>
<tr>
<td>0xA</td>
<td>Light green</td>
</tr>
<tr>
<td>0xB</td>
<td>Light cyan</td>
</tr>
<tr>
<td>0xC</td>
<td>Light red</td>
</tr>
<tr>
<td>0xD</td>
<td>Light magenta</td>
</tr>
<tr>
<td>0xE</td>
<td>Yellow</td>
</tr>
<tr>
<td>0xF</td>
<td>White (high intensity)</td>
</tr>
</tbody>
</table>

ExfieldPlane

This plane contains extended character attribute data.

This plane is a sparse-array of extended character attribute values. All values in the array are binary zero except for character in the presentation space for which the host has specified extended character attributes. The meaning of the extended character attribute values are as follows.

Table 8. 3270 Extended Character Attributes

<table>
<thead>
<tr>
<th>Bit Position (0 is least significant bit)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>7, 6</td>
<td>Character highlighting</td>
</tr>
<tr>
<td>0, 0</td>
<td>Normal</td>
</tr>
<tr>
<td>0, 1</td>
<td>Blink</td>
</tr>
<tr>
<td>1, 0</td>
<td>Reverse video</td>
</tr>
<tr>
<td>1, 1</td>
<td>Underline</td>
</tr>
</tbody>
</table>
Table 8. 3270 Extended Character Attributes (continued)

<table>
<thead>
<tr>
<th>Bit Position (0 is least significant bit)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>5, 4, 3</td>
<td>Character color</td>
</tr>
<tr>
<td></td>
<td>0, 0, 0  Default</td>
</tr>
<tr>
<td></td>
<td>0, 0, 1  Blue</td>
</tr>
<tr>
<td></td>
<td>0, 1, 0  Red</td>
</tr>
<tr>
<td></td>
<td>0, 1, 1  Pink</td>
</tr>
<tr>
<td></td>
<td>1, 0, 0  Green</td>
</tr>
<tr>
<td></td>
<td>1, 0, 1  Turquoise</td>
</tr>
<tr>
<td></td>
<td>1, 1, 0  Yellow</td>
</tr>
<tr>
<td></td>
<td>1, 1, 1  White</td>
</tr>
<tr>
<td>2, 1</td>
<td>Character attribute</td>
</tr>
<tr>
<td></td>
<td>00      Default</td>
</tr>
<tr>
<td></td>
<td>11      Double byte character</td>
</tr>
<tr>
<td>0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

Table 9. 5250 Extended Character Attributes

<table>
<thead>
<tr>
<th>Bit Position (0 is least significant bit)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Normal image</td>
</tr>
<tr>
<td></td>
<td>Reverse image</td>
</tr>
<tr>
<td>6</td>
<td>No underline</td>
</tr>
<tr>
<td></td>
<td>Underline</td>
</tr>
<tr>
<td>5</td>
<td>No blink</td>
</tr>
<tr>
<td></td>
<td>Blink</td>
</tr>
<tr>
<td>4</td>
<td>No column separator</td>
</tr>
<tr>
<td></td>
<td>Column separator</td>
</tr>
<tr>
<td>3, 2, 1, 0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

Appendix B. ECL Planes — Format and Content 405
Appendix C. Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make them available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Subject to IBM’s valid intellectual property or other legally protectable rights, any functionally equivalent product, program, or service may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, New York 10594
USA

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Site Counsel
IBM Corporation
P.O. Box 12195
3039 Cornwallis Road
Research Triangle Park, NC
27709-2195
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement.

This document is not intended for production use and is furnished as is without any warranty of any kind, and all warranties are hereby disclaimed including the warranties of merchantability and fitness for a particular purpose.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States, or other countries, or both:

IBM
VisualAge

Lotus, Notes, SmartSuite, and 1–2–3 are trademarks of Lotus Development Corporation in the United States, other countries, or both.
Java, JavaBeans, and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Lotus, Notes, and SmartSuite are trademarks of the Lotus Development Corporation in the United States, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.
Index

A
autECLConnList
class description 201
methods
Collection Element Methods 204
FindConnectionByHandle 205
FindConnectionByName 205
Refresh 204
StartCommunication 206
StopCommunication 206
properties
APIEnabled 203
CodePage 202
CommStarted 203
ConnType 202
Count 201
Handle 202
Name 202
overview 201
Ready 203
Started 203
autECLConnMgr
class description 207
events
event processing example 210
NotifyStartError 210
NotifyStartEvent 210
NotifyStartStop 210
overview 210
methods
RegisterStartEvent 207
StartConnection 208
StopConnection 209
UnRegisterStartEvent 207
properties, autECLConnList 207
autECLFieldList
class description 211
methods
Collection Element Methods 216
FindFieldByRowCol 217
FindFieldByText 217
GetText 218
overview 216
Refresh 216
SetText 219
properties
Count 211
Display 216
EndCol 213
EndRow 213
HighIntensity 215
Length 214
Modified 214
Numeric 215
overview 211
PenDetectable 215
Protected 214
StartCol 212
StartRow 212
autECLOIA
class description 219
autECLOIA (continued)
methods
CancelWaits 232
overview 227
NotifyCommEvent 227
SelectConnectionByName 227
StartCommunication 229
StopCommunication 229
UnregisterCommEvent 227
WaitForAppAvailable 231
WaitForInputReady 230
WaitForSystemAvailable 230
WaitForTransition 231
properties
Alphanumeric 220
APIEnabled 226
APL 220
CapsLock 222
CodePage 225
CommErrorReminder 223
CommStarted 225
ConnType 224
DBCS 221
Handle 224
Hiragana 221
InputInhibited 223
InsertMode 222
Katakana 221
MessageWaiting 223
Name 224
Numeric 222
NumLock 226
overview 220
Ready 226
Started 225
UpperShift 222
autECLPageSettings
class description 304
methods 311
RefreshTextDefaults 311
SelectConnectionByName 312
SetConnectionByName 312
properties 305
APIEnabled 310
CodePage 309
CommStarted 310
ConnType 309
CPI 306
FontCPI 306
FontLPI 307
FontSize 307
Handle 309
LPI 306
autECLPageSettings (continued)
properties (continued)
  MaxCharsPerLine  308
  MaxLinesPerPage  308
  Name  308
  Ready  311
  Started  310

autECLPrinterSettings
class description  313
methods  321
  SetConnectionByHandle  325
  SetConnectionByName  324
  SetPDTMode  321
  SetPrtToDskAppend  322
  SetPrtToDskSeparate  322
  SetSpecificPrinter  323
  SetWinDefaultPrinter  324
properties  314
  APIEnabled  320
  CodePage  319
  CommStarted  319
  ConnType  318
  Handle  318
  Name  317
  PDTFile  315
  PDTMode  314
  Printer  316
  PrintMode  315
  PromptDialogOption  317
  PrtToDskAppendFile  316
  PrtToDskSeparateFile  317
  Ready  320
  Started  319

autECLPS
class description  233
events
  event processing example  260
  NotifyCommError  259
  NotifyCommEvent  258
  NotifyCommStop  260
  NotifyKeyError  259
  NotifyKeysEvent  258
  NotifyKeyStop  259
  NotifyPSEvent  257
  NotifyPSStop  259
  overview  257
methods
  GetText  244
  GetTextRect  246
  overview  239
  RegisterCommEvent  240
  RegisterKeyEvent  240
  RegisterPSEvent  239
  SearchText  243
  SendKeys  243
  SetConnectionByHandle  242
  SetConnectionByName  241
  SetCursorPos  242
  SetText  245
  StarMacro  247
  StartCommunication  246
  StopCommunication  247
  UnregisterCommEvent  241
  UnregisterKeyEvent  240
  UnregisterPSEvent  240

autECLPS (continued)
properties
  APIEnabled  237
  autECLFieldList  234
  CodePage  236
  CommStarted  237
  ConnType  236
  CursorPosCol  235
  CursorPosRow  235
  Handle  236
  Name  235
  NumCols  235
  NumRows  234
  overview  234
  Ready  238
  Started  237

wait functions
  CancelWaits  257
  Wait  248
  WaitForAttrib  253
  WaitForCursor  248
  WaitForScreen  256
  WaitForString  250
  WaitForStringInRect  251
  WaitWhileAttrib  255
  WaitWhileCursor  249
  WaitWhileScreen  256
  WaitWhileString  250
  WaitWhileStringInRect  252

autECLScreenDesc
class description  261
methods
  AddAttrib  262
  AddCursorPos  263
  AddNumFields  263
  AddNumInputFields  264
  AddOIAInhibitStatus  265
  AddString  265
  AddStringInRect  266
  Clear  267
  overview  262

autECLScreenReco
class description  267
events
  event processing example  271
  NotifyRecoError  270
  NotifyRecoEvent  270
  NotifyRecoStop  270
  overview  270
methods
  AddPS  268
  IsMatch  268
  overview  268
  RegisterScreen  269
  RemovePS  269
  UnregisterScreen  269

autECLSession
class description  271
events
  event processing example  281
  NotifyCommError  280
  NotifyCommEvent  280
  NotifyCommStop  281
  overview  280
methods
  overview  277
  RegisterCommEvent  277
autECLSession (continued)
methods (continued)
  RegisterSessionEvent 277
  SetConnectionByHandle 278
  SetConnectionByName 278
  StartCommunication 279
  StopCommunication 280
  UnregisterCommEvent 278
  UnregisterSessionEvent 277

properties
  APIEnabled 274
  autECLOIA object 275
  autECLPageSettings object 276
  autECLPrinterSettings object 276
  autECLPS object 275
  autECLWinMetrics object 276
  autECLXfer object 275
  CodePage 273
  CommStarted 274
  ConnType 273
  Handle 273
  Name 272
  overview 272
  Ready 275
  Started 274

autECLWinMetrics
class description 281
events
  event processing example 293
  NotifyCommError 293
  NotifyCommEvent 292
  NotifyCommStop 293
  overview 292

methods
  GetWindowRect 290
  overview 288
  RegisterCommEvent 289
  SetConnectionByHandle 290
  SetConnectionByName 289
  SetWindowRect 291
  StartCommunication 291
  StopCommunication 292
  UnregisterCommEvent 289

properties
  Active 284
  APIEnabled 288
  CodePage 287
  CommStarted 287
  ConnType 286
  Handle 286
  Height 284
  Maximized 285
  Minimized 285
  Name 286
  overview 282
  Ready 288
  Restored 285
  Started 287
  Visible 284
  Width 283
  WindowTitle 282
  Xpos 283
  Ypos 283

autECLXfer
class description 294
events
  event processing example 302

autECLXfer (continued)
events (continued)
  NotifyCommError 302
  NotifyCommEvent 301
  NotifyCommStop 302
  overview 301

methods
  overview 297
  ReceiveFile 299
  RegisterCommEvent 297
  SendFile 298
  SetConnectionByHandle 298
  SetConnectionByName 297
  StartCommunication 300
  StopCommunication 301
  UnregisterCommEvent 297

properties
  APIEnabled 296
  CodePage 295
  CommStarted 296
  ConnType 295
  Handle 294
  Name 294
  overview 294
  Ready 296
  Started 295

autSystem
class description 303
methods
  overview 303
  Shell 303

B
Building C++ ECL Programs
description 13
  Microsoft Visual C++ 15

C
ColorPlane 403
Count 335

E
ECL Concepts
  Addressing 5
  Connections, Handles and Names 2
  ECL Container Objects 4
  ECL List Objects 4
  Error Handling 5
  Events 4
  Sessions 3
  ECL Planes 401
  ECLBase
class description 15
methods
  ConvertHandle2ShortName 16
  ConvertPos 18
  ConvertShortName2Handle 17
  ConvertTypeToString 17
  GetVersion 16
  overview 16

ECLCommNotify
class description 44
derivation 45
ECLCommNotify (continued)
methods
  NotifyError 47
  NotifyEvent 47
  NotifyStop 47
overview 46

ECLConnection
class description 19
derivation 19
methods
  ECLConnection Constructor 20
  ECLConnection Destructor 21
  GetCodePage 21
  GetConnType 23
  GetEncryptionLevel 25
  GetHandle 22
  GetName 24
  IsAPIEnabled 27
  IsCommStarted 26
  IsDBCSHost 28
  IsReady 28
  IsStarted 26
overview 20
  RegisterCommEvent 30
  StartCommunication 29
  StopCommunication 29
  UnregisterCommEvent 31

ECLConnList
class description 31
derivation 32
methods
  ECLConnList Constructor 33
  ECLConnList Destructor 33
  FindConnection 35
  GetCount 36
  GetFirstConnection 34
  GetNextConnection 35
overview 32
  Refresh 37

ECLConnMgr
class description 38
derivation 38
methods
  ECLConnMgr Constructor 38
  ECLConnMgr Destructor 39
  GetConnList 39
overview 38
  RegisterStartEvent 42
  StartConnection 40
  StopConnection 41
  UnregisterStartEvent 43

ECLErr
class description 48
derivation 48
methods
  GetMsgNumber 48
  GetMsgText 50
  GetReasonCode 49
overview 48

ECLField
class description 50
derivation 51
methods
  GetAttribute 64

ECLField (continued)
methods (continued)
  GetEnd 56
  GetEndCol 58
  GetEndRow 57
  GetLength 59
  GetScreen 59
  GetStart 53
  GetStartCol 55
  GetStartRow 54
  IsDisplay 62
  IsHighIntensity 62
  IsModified 62
  IsNumeric 62
  IsPenDetectable 62
  IsProtected 62
overview 53
  SetText 61

ECLFieldList
class description 65
derivation 65
methods
  FindField 69
  GetFieldCount 66
  GetFirstField 67
  GetNextField 68
overview 65
  Refresh 66
properties 65

ECLKeyNotify
class description 71
derivation 72
methods
  NotifyError 75
  NotifyEvent 74
  NotifyStop 75
overview 74

ECLListener
class description 75
derivation 76

ECLOIA
class description 76
derivation 76
methods
  ECLOIA Constructor 77
  GetStatusFlags 86
  InputInhibited 85
  IsAlphanumeric 77
  IsAPL 78
  IsCapsLock 81
  IsCommErrorReminder 83
  IsDBCS 80
  IsHiragana 79
  IsInsertMode 82
  IsKatakana 79
  IsMessageWaiting 83
  IsNumeric 81
  IsUpperShift 80
overview 76
  RegisterOIAEvent 87
  UnregisterOIAEvent 87

wait functions
  WaitForAppAvailable 84
  WaitForInputReady 84
  WaitForSystemAvailable 84
  WaitForTransition 85
ECLIOANotify

class description 88
derivation 88
methods
NotifyError 89
NotifyEvent 89
NotifyStop 89
overview 88

ECLPageSettings

class description 177
derivation 177
methods 178
ECLPageSettings Constructor 178
GetCPI 180
GetFontFaceName 183
GetFontSize 183
GetLPI 181
GetMaxCharsPerLine 185
GetMaxLinesPerPage 184
IsFontCPI 180
IsFontLPI 182
RestoreDefaults 186
SetCPI 179
SetFontFaceName 182
SetFontSize 183
SetLPI 181
SetMaxCharsPerLine 185
SetMaxLinesPerPage 184
properties 177
usage notes 177

ECLPrinterSettings

class description 186
derivation 186
methods 187
ECLPrinterSettings Constructor 187
GetPDTFile 189
GetPrinterName 196
GetPrintMode 190
GetPrtToDskAppendFile 192
GetPrtToDskSeparateFile 194
IsPDTMode 190
IsPromptDialogEnabled 197
SetPDTMode 188
SetPromptDialog 197
SetPrtToDskAppend 191
SetPrtToDskSeparate 193
SetSpecificPrinter 195
SetWinDefaultPrinter 195
properties 186
usage notes 187

ECLPS

class description 90
derivation 90
Japanese, 1390/1399 code page
methods 105
ConvertPosToCol 111
ConvertPosToRow 110
ConvertPosToRowCol 109
ConvertRowColToPos 110
ECLPS Constructor 92
ECLPS Destructor 93
GetCursorPos 96
GetCursorPosCol 98
GetCursorPosRow 97
GetFieldList 113
GetHostCodePage 94

ECLPS (continued)

methods (continued)
GetOSCodePage 94
GetPCCodePage 94
GetScreen 103
GetScreenRect 106
GetSize 94
GetSizeCols 96
GetSizeRows 95
overview 90
RegisterKeyEvent 112
RegisterPSEvent 122
SearchText 101
SendKeys 99
SetCursorPos 99
SetText 108
StartMacro 123
UnregisterKeyEvent 113
UnregisterPSEvent 124

ECLPS Field
Japanese, 1390/1399 code page
SearchText 102
SendKeys 100

ECLPSEvent

class description 124
derivation 125
methods
GetEnd 126
GetEndCol 127
GetEndRow 127
GetPS 125
GetStart 126
GetStartCol 127
GetStartRow 126
GetType 126
overview 125

ECLPSListener

class description 128
derivation 128
methods
NotifyError 129
NotifyEvent 129
NotifyStop 130
overview 129

ECLPSNotify

class description 130
derivation 131
methods
NotifyError 132
NotifyEvent 131
NotifyStop 132
overview 131

ECLRecoNotify

class description 133
derivation 133
methods
ECLNotify Deconstructor 133
ECLRecoNotify Constructor 133
NotifyError 134
NotifyEvent 134
NotifyStop 134
overview 133

ECLScreenDesc

class description 135
derivation 135

Index 413
ECLScreenDesc (continued)
methods
AddAttrib 136
AddCursorPos 137
AddNumFields 138
AddNumInputFields 138
AddOIAInhibitStatus 139
AddString 139
AddStringInRect 140
Clear 141
ECLScreenDesc Constructor 135
ECLScreenDesc Destructor 136
overview 135
ECLScreenReco Class 141
ECLSession
class description 145
derivation 145
methods
ECLSession Constructor 146
ECLSession Destructor 147
GetOIA 148
GetPageSettings 150
GetPrinterSettings 150
GetPS 147
GetWinMetrics 149
GetXfer 148
overview 146
RegisterUpdateEvent 151
UnregisterUpdateEvent 151
ECLStartNotify
class description 151
derivation 152
methods
NotifyError 154
NotifyEvent 154
NotifyStop 155
overview 154
ECLUpdateNotify
class description 155
ECLWinMetrics
class description 155
derivation 155
methods
Active 168
ECLWinMetrics Constructor 156
ECLWinMetrics Destructor 157
GetHeight 164
GetWidth 162
GetWindowRect 165
GetWindowTitle 158
GetXpos 159
GetYpos 161
IsMaximized 170
IsMinimized 169
IsRestored 171
IsVisible 167
overview 156
SetActive 168
SetHeight 165
SetMaximized 170
SetMinimized 169
SetRestored 172
SetVisible 167
setWidth 163
SetWindowRect 166
SetWindowTitle 158
GetXpos 160
ECLWinMetrics (continued)
methods (continued)
SetYpos 162
ECLXfer
class description 172
derivation 172
methods
ECLXfer Constructor 173
ECLXfer Destructor 173
overview 173
ReceiveFile 175
SendFile 174
ELLHAPI, migrating from
Events 8
Execution/Language Interface 6
Features 6
Presentation Space Models 8
PS Connect/Disconnect, Multithreading 9
SendKey Interface 8
Session IDs 7
ExtendedFieldPlane 404

F
FieldPlane 401

J
Japanese, 1390/1399 code page
GetScreen, ECLField 60
GetScreen, ECLPS 105
overview 2
SearchText, ECLPS 102
SendKeys, ECLPS 100
SetText, ECLField 61
Java, Host Access Class Library 395

K
keywords 397

L
lsxECLConnection
class description 330
methods
overview 333
StartCommunication 333
StopCommunication 334
properties
APIEnabled 333
CodePage 332
CommStarted 332
ConnType 331
Handle 331
Name 331
overview 331
Ready 333
Started 332
lsxECLConnList
class description 334
methods
FindConnectionByHandle 336
FindConnectionByName 336
overview 335

414 Host Access Class Library
lsxECLConnList (continued)  
methods (continued)  
  Refresh 335  
properties, Count 335  
lsxECLConnMgr  
class description 337  
methods  
  overview 337  
  StartConnection 337  
  StopConnection 338  
properties, lsxECLConnList 337  
lsxECLField  
class description 339  
methods  
  GetText 343  
 overview 342  
  SetText 343  
properties  
  Display 342  
  EndCol 340  
  EndRow 340  
  HighIntensity 342  
  Length 341  
  Modified 341  
  Numeric 341  
  overview 339  
  PenDetectable 342  
  Protected 341  
  StartCol 340  
  StartRow 340  
lsxECLFieldList  
class description 344  
methods  
  FindFieldByRowCol 345  
  FindFieldByText 345  
  overview 344  
  Refresh 344  
properties, Count 344  
lsxECLIOIA  
class description 346  
methods  
  overview 353  
  WaitForAppAvailable 354  
  WaitForInputReady 353  
  WaitForSystemAvailable 353  
  WaitForTransition 354  
properties  
  Alphanumeric 348  
  APIEnabled 352  
  APL 348  
  CapsLock 349  
  CodePage 351  
  CommErrorReminder 349  
  CommStarted 352  
  ConnType 351  
  DBCS 348  
  Handle 351  
  Hiragana 348  
  InputInhibited 350  
  InsertMode 349  
  Katakana 348  
  MessageWaiting 350  
  Name 350  
  Numeric 349  
  overview 347  
  Ready 352  
  Started 352  
lsxECLPS  
class description 355  
description 329  
methods  
  GetText 362  
  GetTextRect 363  
  overview 359  
  SearchText 361  
  SendKeys 360  
  SetCursorPos 359  
  SetText 362  
  WaitForAttrib 369  
  WaitForCursor 364  
  WaitForScreen 371  
  WaitForString 365  
  WaitForStringInRect 367  
  WaitForWhileAttrib 370  
  WaitForWhileScreen 371  
  WaitForWhileString 366  
  WaitForWhileStringInRect 368  
  WatiWhileCursor 364  
properties  
  APIEnabled 358  
  CodePage 358  
  CommStarted 358  
  ConnType 357  
  CursorPosCol 357  
  CursorPosRow 356  
  Handle 357  
  lsxECLFieldList 357  
  Name 357  
  NumCols 356  
  NumRows 356  
  overview 355  
  Ready 359  
  Started 358  
lsxECLScreenDesc  
class description 373  
methods  
  AddAttrib 373  
  AddCursorPos 374  
  AddInputFields 375  
  AddNumFields 375  
  AddOIAInhibitStatus 376  
  AddString 376  
  AddStringInRect 377  
  Clear 378  
  overview 373  
lsxECLScreenReco  
class description 372  
methods, IsMatch 372  
lsxECLSession  
class description 378  
methods, overview 382  
properties  
  APIEnabled 381  
  CodePage 380  
  CommStarted 381  
  ConnType 380  
  Handle 380  
  lsxECLWinMetrics 382  
  lsxECLIOIA 382  
  lsxECLPS 381  
  lsxECLXfer 382  

Index 415
lsxECLSession (continued)
  properties (continued)
    Name 379
    overview 379
    Ready 381
    Started 381

lsxECLWinMetrics
  class description 382
  methods
    GetWindowRect 388
    overview 388
    SetWindowRect 389

properties
  Active 385
  APIEnabled 387
  CodePage 387
  CommStarted 387
  ConnType 386
  Handle 386
  Height 385
  Maximized 385
  Minimized 385
  Name 386
  overview 383
  Ready 388
  Restored 386
  Started 387
  Visible 385
  Width 384
  WindowTitle 384
  Xpos 384
  Ypos 384

lsxECLXfer
  class description 389
  methods
    overview 392
    ReceiveFile 393
    SendFile 393

properties
  APIEnabled 392
  CodePage 391
  CommStarted 392
  ConnType 391
  Handle 391
  Name 390
  overview 390
  Ready 392
  Started 392

M

Migrating from EHLLAPI
  Events 8
  Execution/Language Interface 6
  Features 6
  Presentation Space Models 8
  PS Connect/Disconnect, Multithreading 9
  SendKey Interface 8
  Session IDs 7
  mnemonic 397

O

objects, automation (continued)
  autECLFieldList 211
  autECLOIA 219
  autECLField 233
  autECLPageSettings 276
  autECLPrinterSettings 276
  autECLS 261
  autECLScreenRec 267
  autECLSession 271
  autECLWinMetrics 281
  autECLXfer 294
  autSystem 303
  description 199

objects, C++
  description 11
  ECLBase 15
  ECLCommNotify 44
  ECLConnection 19
  ECLConnList 31
  ECLConnMgr 38
  ECLErr 48
  ECLField 50
  ECLFieldList 65
  ECLKeyNotify 71
  ECLListener 75
  ECOIA 76
  ECOIA Notify 88
  ECLPS 90
  ECLPSNotify 124
  ECLPSListener 128
  ECLPSNotify 130
  ECLRecNotify 133
  ECLScreenDesc 135
  ECLScreenRec 141
  ECLSession 145
  ECLStartNotify 151
  ECLXfer 172

objects, LotusScript
  lsxECLConnection 330
  lsxECLConnList 334
  lsxECLConnMgr 337
  lsxECLField 339
  lsxECLFieldList 344
  lsxECLIOIA 346
  lsxECLPS 355
  lsxECLScreenDesc 373
  lsxECLScreenRec 372
  lsxECLSession 378
  lsxECLWinMetrics 382
  lsxECLXfer 389

S

Sendkeys mnemonic keywords 397

T

TextPlane 401

U

Unicode, 1390/1399 code page
  GetScreen, ECLField 60
  GetScreen, ECLPS 105
  overview 2
  SearchText, ECLPS 102
Unicode, 1390/1399 code page (continued)
SendKeys, ECLPS 100
SetText, ECLField 61
Program Number: 5639-I70

Printed in USA