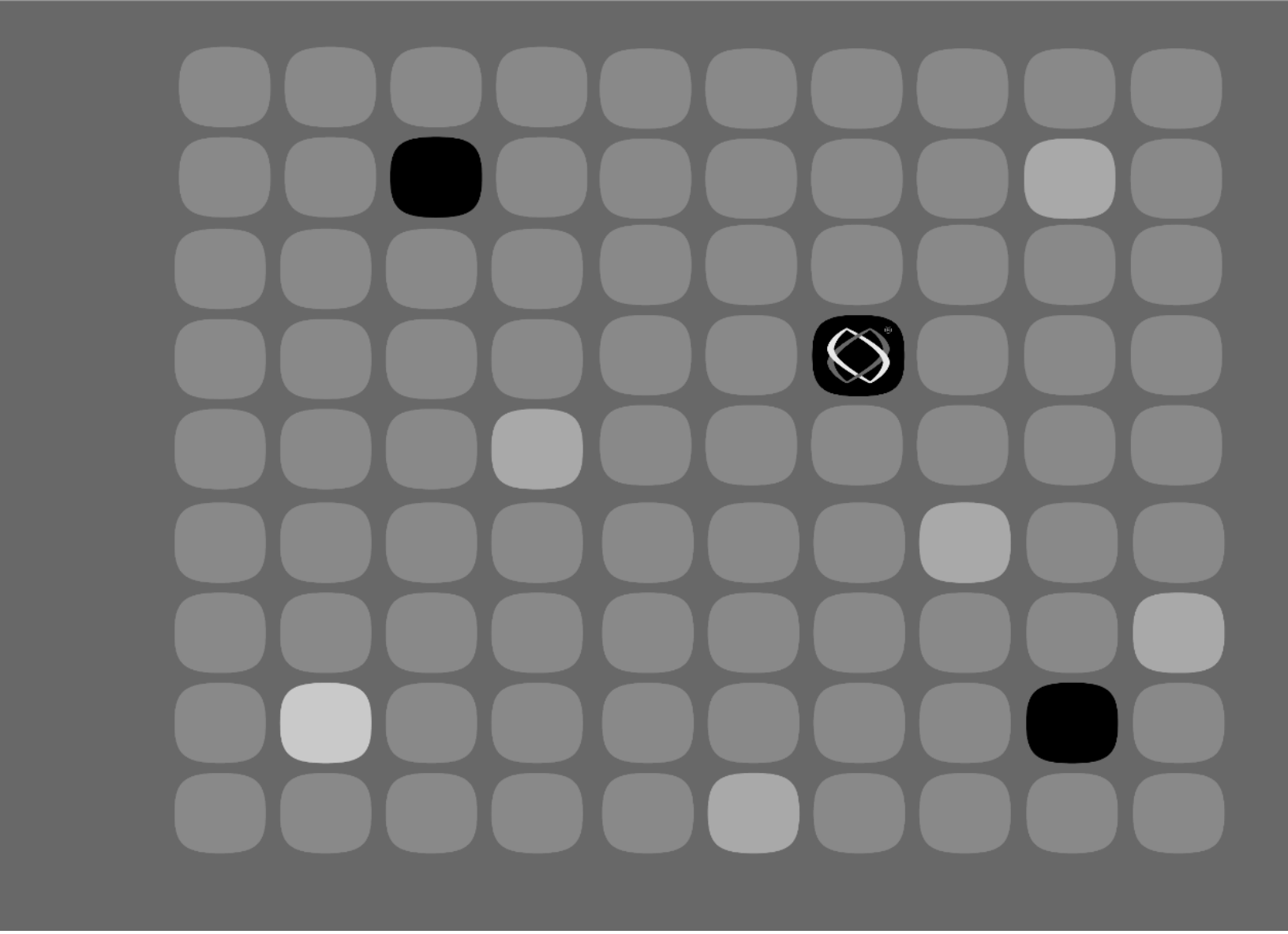


Programmer's Reference

Attachmate®

INFOCONNECT®

Enterprise Edition

PTR

Print and Transaction Router

OLE Application Programming Interface

UNISYS

Copyrights and Notices

Attachmate® INFOConnect® Enterprise Edition

Patents

This Attachmate software is protected by U.S. patents 6252607 and 6803914.

Trademarks

Attachmate, the Attachmate logo, CryptoConnect, FileXpress, and PEPgate are either registered trademarks or trademarks of Attachmate Corporation in the USA. INFOConnect is a registered trademark of Unisys Corporation. FIPS 140-1 Validated is a certification mark of NIST, which does not imply product endorsement by NIST, the U.S. or Canadian Governments. All other trademarks, trade names, or company names referenced in product materials are used for identification only and are the property of their respective owners.

Attachmate Software License Agreement

A copy of the Attachmate software license agreement governing this product can be found in a 'license' file in the root directory of the product.

Licensor

Attachmate Corporation
1500 Dexter Avenue North
Seattle, WA 98109 USA
USA
+1.206.217.7100
http://www.attachmate.com

Third-Party Notices

Third Party Terms and notices are provided in a 'thirdpartynotices' file in the root directory of the product.

Contents

About This Guide

This guide is intended for C/C++ programmers who want to build client applications for INFOConnect Print and Transaction Router (PTR) using the PTR OLE API.

The following sections are included in this preface:

Conventions

This guide uses the following conventions:

- Text that you type as well as messages and prompts that appear on the screen are shown `in this type style`. The same type style in **`Bold`** is used in the examples to highlight items specifically mentioned in the preceding text or procedure.
- In addition to emphasizing text and highlighting terms used for the first time, *italic* indicates variables. For example, if you were asked to type *drive:**directory**filename.ext*, you would type the actual drive, directory, and file name in place of the italicized words.
- Microsoft® Windows® often provides several ways to select an option. For example, you can click a button using a mouse, press Enter when a particular button has the focus, or press an accelerator key. Even though other ways may exist, this guide usually explains how to select options by clicking a mouse.
- This guide explains how to perform actions using the standard Windows desktop. If you configured your desktop to operate like a Web page, you might need to use different procedures. For example, to start an application, you might single-click a file rather than double-click it.
- Functions unique to the PTR OLE API are referred to as PTR OLE API functions. OLE functions (of which Microsoft Foundation Classes are a subset) are referred to as OLE API functions.

Caution: This caution icon indicates that there is a possibility of losing data or corrupting files. When you see this caution icon, follow the instructions carefully.

Abbreviations

The following abbreviations are used throughout this guide and are provided here for quick reference.

Abbreviation	Name
ANSI	American National Standards Institute
API	Application programming interface
CLSID	Class ID
COM	Component Object Model
CTS	Clear to send
DSR	Data set ready
IDE	Integrated development environment
MDI	Multiple document interface
MFC	Microsoft Foundation Classes
OLE	Object Linking and Embedding
ProgID	Program ID
PTR	Print and Transaction Router
SDI	Single document interface
TID	Terminal identifier

Related Documentation

Additional information exists in the form of INFOConnect PTR-related guides and online help.

Guides

PTR-related documentation consists of the following guides:

- The *INFOConnect PTR API Programmer's Reference* provides you with details about how you can use the PDK (PTR development kit) to design a custom host filter.
- The *INFOConnect PTR OLE API Programmer's Reference* provides you with details on how to make an application that monitors PTR routes.
- The *INFOConnect PTR User API Programmer's Reference* provides you with details on how to make an application that prints vouchers and other specialized tickets from your PC by sending data directly to a printer.

PTR OLE API Readme file

This Readme (READOLE.TXT) contains important product notices and compatibility information about the PTR OLE API.

Introducing the PTR OLE API

1

This chapter explains what the PTR OLE API is and the functions it provides. The following sections are included:

What Is the PTR OLE API?

The PTR OLE API is a set of function calls you can use to write PTR client applications that communicate with the PTR component object model (COM) server to monitor the status of INFOConnect PTR routes.

The main part of the PTR OLE API is the type library file (PTRAPI.TLB). This file provides access to the same interfaces and functions used to develop the PTR System Tray.

PTR OLE Interfaces

The PTR OLE contains the following three interfaces:

- ISystem

  ISystem manages functions that return system type information, such as configuration file names.

- IRoutes

  IRoutes manages functions that return information about all of the routes collectively.

- IRoute

  IRoute manages functions that return information about individual routes.

Prerequisites

To build a client application that invokes PTR OLE components, you must use Microsoft Visual C++®. In addition, all of the procedures in this guide describe how to complete the tasks using Microsoft Developer Studio® 97.

You must also be familiar with INFOConnect and PTR. You should be able to configure a PTR route, which includes configuring a host path, printer queue path, and host filter using the INFOConnect Manager.

In order to use the PTR OLE API, your system must have the following software:

- INFOConnect Connectivity Services, version 4.2

Additional Requirements

To build a client application that invokes PTR OLE components, you must also be familiar with the following:

- C/C++ programming language
- OLE Automation
- Visual C++ integrated development environment (IDE)
- Microsoft Foundation Classes (MFC) library application framework
- Microsoft Win32® API

Sample Client Application

You can find a complete sample client application that uses the OLE API in the following directory on the installation CD:

\\32\PDK\OLEAPI\Sample\ATSTCP32.DLL

The project file for this application, ATBPTR.DSP, is a Microsoft Studio Visual C++ 5.0 project file; however, you can access the source and header files in any text editor.

This sample is documented in Appendix A, "Sample Client Application," beginning on page 49 of this guide.

It is recommended that you run, and even modify, the sample client application before you write your own client application.

Functions Listed by Associated Interface

The following table is organized according to the interface each group of functions is associated with. Each function in this table is further explained in its own section, beginning with "GetCount" on page 22.

Interface	Function
ISystem	GetRoutes
IRoutes	GetCount
IRoutes	GetStatusUpdate
IRoutes	Item
IRoute	GetHostFilter
IRoute	GetHostPath
IRoute	GetHostStatus
IRoute	GetHostXlateTable
IRoute	GetJobs
IRoute	GetLockState
IRoute	GetName
IRoute	GetPrinterTimeout
IRoute	GetPrintQueuePath
IRoute	GetPrintQueueStatus
IRoute	GetQueueXlateTable
IRoute	GetStatus
IRoute	GetSubmitTime

Building a PTR Client Application

2

In This Chapter

This chapter explains how to make and use a client application that can monitor PTR routes. The following sections are included:

Overview

This section provides an overview of the steps you must complete in order to make and use a client application that works with PTR. These steps are described in greater detail in the remainder of this chapter.

> **Note:** All of the procedures in this guide describe how to develop client applications using Microsoft Developer Studio 97 (Microsoft Visual C++ 5.0).

Complete the following steps to create and use a PTR client application:

1. Run the sample client application. It is also recommended that you modify the sample application and run it again before you develop your client application. For a description of the sample client application, see Appendix A, "Sample Client Application," beginning on page 49.

2. Determine which functions you want to include in your client application and how you want to design it.

3. Use the AppWizard in Microsoft Developer Studio to configure a new project. For complete instructions, see "Creating a New Client Application" beginning on page 10.

4. Use the ClassWizard in Microsoft Developer Studio to import the classes provided by the PTR OLE API. For complete instructions, see "Importing the PTR OLE API Classes" beginning on page 11.

5. Include the appropriate files in your client application to initialize the OLE components. For complete instructions, see "Initializing OLE Automation" beginning on page 12.

6. Establish communication between your client application and the PTR server. For complete instructions, see "Communicating with the PTR Server" beginning on page 14.

7 Call whatever PTR OLE API functions that you want to use. For complete instructions, see "Calling PTR OLE API Functions" beginning on page 16.

8 Compile and run your client application.

Creating a New Client Application

This section explains how to use AppWizard in Microsoft Developer Studio to create a new project (application). Before you complete these steps, determine what kind of a client application you would like to make, such as a single document interface (SDI), a multiple document interface (MDI), or a dialog.

> **Note:** All of the procedures in this guide describe how to develop client applications using Microsoft Developer Studio 97 (Microsoft Visual C++ 5.0).

Complete the following steps to open a new project in Microsoft Developer Studio:

1. Run Microsoft Developer Studio.
2. From the File menu, click New.
3. Click the Projects tab.
4. Choose the appropriate MFC AppWizard for your project.

   > **Note:** The sample client application is a .DLL, but you can choose other application types.

5. Name your project and click OK.

   > **Note:** The name you assign this project will be the name of your client application. Certain parts of your client application, such as files, will also derive names from the name of the project, so choose a meaningful name.

6. Configure your client application by completing the AppWizard screens as you are prompted.
7. Click Finish.
8. Click OK or Cancel to accept or cancel the specifications you chose for your client application. You can modify your choices by clicking Back to access the AppWizard screens.

   AppWizard will add several files to your client application, though the specific files will vary depending on the options you chose.

Importing the PTR OLE API Classes

This section describes how to give your client application access to the classes provided with the PTR OLE API. These classes are available through the PTR type library file (PTRAPI.TLB) that you will import into your client application using ClassWizard.

> **Note:** All of the procedures in this guide describe how to develop client applications using Microsoft Developer Studio 97 (Microsoft Visual C++ 5.0).

Complete the following steps to import the PTR OLE API classes into your client application:

1. From the View menu, click ClassWizard.
2. Click the Automation tab.
3. Click Add Class, and choose From a type library.
4. Click Browse and select PTRAPI.TLB from the \\32\PDK\OLEAPI directory on the Millennium I with PTR System Tray CD.
5. Click Open.

   The Confirm Classes dialog box displays the three server classes, ISystem, IRoutes, and IRoute.
6. Click OK.
7. At the original ClassWizard screen, click OK.

   ClassWizard will read the type library file and generate two source files, PTRAPI.H and PTRAPI.CPP, for your client application.

Initializing OLE Automation

To ensure that your client application can use OLE Automation, you must include the OLE components (AFXDISP.H) along with the standard system include files in the appropriate header file (STDAFX.H). Then, your client application's InitInstance function (or in the case of the sample dialog, the OnInitDialog function) must initialize those OLE components.

Complete the following steps to initialize OLE Automation:

1 Add the following line at the end of STDAFX.H.

```
#include <afxdisp.h>
```

The following example shows where to add the line, and the actual line is shown in bold.

```
// stdafx.h : include file for standard system include
// files, or project specific include files that are used
// frequently, but are changed infrequently
.
.
.
#include <afxwin.h> // MFC core and standard components
#include <afxext.h>         // MFC extensions
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h> // MFC support for Windows Common
                  //Controls
#endif // _AFX_NO_AFXCMN_SUPPORT
#include <afxdisp.h>

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional
// declarations immediately before the previous line.
.
.
.
```

2 Add the following lines at the beginning of the client application's InitInstance function.

```
if (!AfxOleInit())
{
  AfxMessageBox("Fail OLE initialize!", MB_OK);
  return FALSE;
}
```

These lines tell the client application to return the error `"Fail OLE initialize!"` if the object does not initialize successfully.

The following example shows where to add the lines:

```
BOOL CATBPTRDlg::OnInitDialog()
{
  if (!AfxOleInit())
  {
    AfxMessageBox("Fail OLE initialize!", MB_OK);
    return FALSE;
  }

  CString strRouteName = "Miltope Printer";
  Variant = strRouteName;

  InitInterface();

  CDialog::OnInitDialog();

  SetIcon(m_hIcon, TRUE);
  SetIcon(m_hIcon, FALSE);

  m_printerName.SetWindowText("Miltope Printer");

  return TRUE;
}
```

Communicating with the PTR Server

To enable your client application to call PTR OLE API functions, your application must communicate with the PTR server.

To establish communication with the PTR server, your client application must call several functions to obtain a class ID (CLSID), a pointer, and an OLE dispatch (IDispatch) pointer for the PTR server. Then, the client application must attach the IDispatch pointer to the PTR OLE API interface whose functions you want to call.

The functions your client application must call are listed as follows:

- CLSIDFromProgID (from the Microsoft Win32 API)

  When you installed the PTR package, the Windows NT® registry received a program ID (ProgID) for the PTR server (PTR.System). You will supply this ID in the call to obtain the CLSID.

- GetActiveObject (from the OLE API)
- QueryInterface (from the OLE API)

  This function gives your client application access to the PTR Server interfaces.

- AttachDispatch (from the OLE API)

  This function is from the COleDispatchDriver class, whose functions attach, detach, create, and release IDispatch pointers.

Complete the following steps to establish communication with the PTR server:

1. Call CLSIDFromProgID function and supply the ProgID (ptr.system).
2. Call the OLE API GetActiveObject function to obtain an OLE pointer for the PTR Server.
3. Call the OLE API QueryInterface function to obtain an IDispatch pointer for the PTR Server.

4 Call COleDispatchDriver::AttachDispatch, to attach the IDispatch pointer to the PTR OLE API interface.

The following example demonstrates how to establish communication with the PTR server:

```
void CATBPTRDlg::InitInterface()
{
  CLSID clsid;
  LPDISPATCH lpDispatch;
  LPUNKNOWN pUnk;
  HRESULT hr;
  COleException e;

  if (CLSIDFromProgID(OLESTR("ptr.system"), &clsid) !=
     NOERROR)
  {
      AfxMessageBox("Fail finding CLSID from Program
     ID!", MB_OK);
      return;
   }
  if (GetActiveObject(clsid, NULL, &pUnk) == NOERROR)
  {
      hr = pUnk->QueryInterface(IID_IDispatch,
     (LPVOID*)&lpDispatch);
      pUnk->Release();
    if (hr == NOERROR)
    {
      m_system.AttachDispatch(lpDispatch, TRUE);
    }
  }
{
```

Calling PTR OLE API Functions

Whenever you call a PTR OLE API function, you must also create a PTR server object. Server objects correspond to the PTR OLE API interfaces, so you must create an object for the interface whose function you are calling. For instance, if you want to call an IRoute function, you must create an IRoute object within the call to that IRoute function.

Complete the following steps to call a PTR OLE API function:

1. Within the definition for the function, create a PTR server object by declaring the following:

   ```
   object_type object_name;
   ```

   where *object_type* is the name of the interface you want to access (either IRoute, ISystem, or IRoutes), and *object_name* is the name of your object.

2. After the object is created, call AttachDispatch to attach the IDispatch pointer to the object.

3. Call a function from the interface you named as *object_type*.

   > **Note:** For each function you call, you must call the IRoutes function, m_routes.Item(Variant), to obtain the IDispatch pointer to attach to the IRoute object. The m_routes function, which is a member of the CATBPTRDlg class, already exists in InitInterface. For more information on the Item function, see "Item" on page 47.

4. Repeat steps 1–3 for each PTR OLE API function you want to call.

5. Compile your client application once you have called all of the PTR OLE API functions you want to use.

The following example shows how to call the IRoute function GetStatus:

```
void CATBPTRDlg::GetRouteStatus()
{
   long routeStatus = 0; //declaration

/*Call the IRoutes function, m_route.Item(Variant), to
   obtain the IDispatch pointer to attach to the IRoute
   object. The m_routes function already exists in
   InitInterface().*/

   LPDISPATCH lpDispatch = m_routes.Item(Variant);

/*This is the function call to the PTR Server. The return
   value is placed in the routeStatus variable.*/

   if (lpDispatch)
   {
      IRoute route;
      route.AttachDispatch(lpDispatch, TRUE);
      routeStatus = route.GetStatus();
   }
/*With the following switch statement, the application
   will display the status when given the return value.*/

   switch (routeStatus)
   {
      case PTRRSE_DISABLED:
         m_printerState.SetWindowText("Disabled");
         break;

      case PTRRSE_ENABLED:
         m_printerState.SetWindowText("Enabled");
         break;

      case PTRRSE_ACTIVE:
         m_printerState.SetWindowText("Active");
         break;

      case PTRRSE_FAILED:
         m_printerState.SetWindowText("Failed");
         break;

      case PTRRSE_WARNING:
         m_printerState.SetWindowText("Warning");
   }
}
```

Understanding PTR OLE API Functions

3

In This Chapter

This chapter provides descriptions and examples of each PTR OLE API function. The following sections are included:

How the PTR OLE API Functions Are Documented

The following information is given for the functions described in this chapter:

- Description—A brief description of the function immediately follows the function name.
- Associated Interface—This names the PTR interface that manages the function being described.
- Prototype—This is the function syntax.
- Returned Value Type—This is the type of value returned by the function.
- Example—Sample code for most functions appears after the returned value. The prototype is shown in bold in the example.

PTR OLE API Functions

See "Functions Listed by Associated Interface" on page 5 for a complete list of functions. Each function has its own section in this chapter, and these sections are organized alphabetically by function name.

> **Note:** Your particular host filter and printer combination may return only *some* of the statuses listed for each function.

GetCount

GetCount returns the number of existing routes, including the routes that are not currently displayed on the PTR System Tray.

Associated Interface

IRoutes

Prototype

```
GetCount();
```

Returned Value Type

long

Example

```
CStringList& CATBPTRDlg::GetRouteList()
{
   LPDISPATCH lpDispatch;
   m_listRoutes.RemoveAll();

   // Parse out route name into string list
   for (int i = 0, nCount = m_routes.GetCount(); (i <
   nCount); i++)
   {
      CString strRouteName;
      COleVariant Variant = (BYTE) (i + 1);
      lpDispatch = m_routes.Item(Variant);

      if (lpDispatch)
      {
          IRoute route;
          route.AttachDispatch(lpDispatch, TRUE);
          strRouteName = route.GetName();
          m_listRoutes.AddTail(strRouteName);
      }
   }

   return (m_listRoutes);
}
```

GetHostFilter

GetHostFilter returns a host filter file name and path.

Associated Interface

IRoute

Prototype

```
GetHostFilter();
```

Returned Value Type

CString

Example

```
CString  CATBPTRDlg::GetHostFilter( )
{
   CString strHostFilter;

   LPDISPATCH lpDispatch = m_routes.Item(Variant);

   if (lpDispatch)
   {
      IRoute route;
      route.AttachDispatch(lpDispatch, TRUE);
      strHostFilter = route.GetHostFilter();
   }

   return (strHostFilter);
}
```

GetHostPath

GetHostPath returns path IDs.

Associated Interface

IRoute

Prototype

```
GetHostPath();
```

Returned Value Type

CString

Example

```
CString CATBPTRDlg::GetHostPath()
{
   CString strHostPath;

   LPDISPATCH lpDispatch = m_routes.Item(Variant);

   if (lpDispatch)
   {
      IRoute route;
      route.AttachDispatch(lpDispatch, TRUE);
      strHostPath = route.GetHostPath();
   }

   return (strHostPath);
}
```

GetHostStatus

GetHostStatus returns the host status, which is the status of the INFOConnect host path. The following are possible values:

Returned Value	Description
PTRHSE_BROKEN	The INFOConnect host path lost the active host connection. How to determine this varies by transport type.
PTRHSE_CLOSED	The INFOConnect host path is inactive.
PTRHSE_CONNECTED	The INFOConnect host path is connected to a destination. This usually indicates an active connection with the host.
PTRHSE_DUPLICATEDTID	Two workstations are using the same terminal identifier (TID) to configure a host path.
PTRHSE_ESTABLISHED	The INFOConnect host path is established, but the host connection is not yet active.
PTRHSE_NOCONFIG	The PEPGate™ configuration file is invalid, nonexistent, or empty.
PTRHSE_NOTIDCONFIGURED	The TID in the host path is not found in the PEPGate configuration file.
PTRHSE_POLLING	The INFOConnect host path has an active host connection. How to determine this varies by transport type.
PTRHSE_SECURITYDENIED	The IP address for the workstation is not listed in the PEPGate configuration file.
PTRHSE_TRANSMITTING	The INFOConnect host path is sending data.

Associated Interface

IRoute

Prototype

```
GetHostStatus();
```

Returned Value Type

long

Example

```
void CATBPTRDlg::GetHostStatus()
{
   long hostStatus = 0;

   LPDISPATCH lpDispatch = m_routes.Item(Variant);

   if (lpDispatch)
   {
      IRoute route;
      route.AttachDispatch(lpDispatch, TRUE);
      hostStatus = route.GetHostStatus();
   }

   switch (hostStatus)
   {
      case PTRHSE_CLOSED:
          m_hostState.SetWindowText("Closed");
          break;

      case PTRHSE_ESTABLISHED:
         m_hostState.SetWindowText("Established");
         break;

      case PTRHSE_CONNECTED:
         m_hostState.SetWindowText("Connected");
         break;

      case PTRHSE_POLLING:
         m_hostState.SetWindowText("Polling");
         break;

      case PTRHSE_TRANSMITTING:
         m_hostState.SetWindowText("Transmitting");
         break;

      case PTRHSE_BROKEN:
         m_hostState.SetWindowText("Broken");
         break;

      case PTRHSE_DUPLICATEDTID:
         m_hostState.SetWindowText("Duplicated TID");
         break;

      case PTRHSE_NOCONFIG:
         m_hostState.SetWindowText("No Config");
         break;
```

```
            case PTRHSE_NOTIDCONFIGURED:
               m_hostState.SetWindowText("No TID Configured");
               break;

            case PTRHSE_SECURITYDENIED:
               m_hostState.SetWindowText("Security Denied");
         }
      }
```

GetHostXlateTable

GetHostXlateTable returns the host-to-printer translation table file name and its path.

PTR uses translation tables to convert characters sent from the printer to the host or from the host to the printer. PTR provides several sample translation tables that you can use, as follows:

File name	Description
ANSICAPS.XLT	Converts all characters to their ANSI uppercase equivalent.
ANSILOWR.XLT	Converts all characters to their ANSI lowercase equivalent.
CRTOLF.XLT	Converts all carriage return characters to line feed characters.
MAKECR.XLT	Converts all end-of-line sequences to carriage return characters.
MAKECRLF.XLT	Converts all end-of-line sequences to carriage return and line feed characters.
MAKELF.XLT	Converts all end-of-line sequences to line feed characters.

Associated Interface

IRoute

Prototype

```
GetHostXlateTable();
```

Returned Value Type

CString

Example

```
CString CATBPTRDlg::GetHostXlatTable()
{
   CString strXlateTable;

   LPDISPATCH lpDispatch = m_routes.Item(Variant);

   if (lpDispatch)
   {
      IRoute route;
      route.AttachDispatch(lpDispatch, TRUE);
      strXlateTable = route.GetHostXlateTable();
   }

   return (strXlateTable);
}
```

GetJobs

GetJobs returns the number of jobs in the printer queue.

Associated Interface

IRoute

Prototype

```
GetJobs();
```

Returned Value Type

long

Example

```
void CATBPTRDlg::GetJobCount()
{
   long jobCount = 0;

   LPDISPATCH lpDispatch = m_routes.Item(Variant);

   if (lpDispatch)
   {
      IRoute route;
      route.AttachDispatch(lpDispatch, TRUE);
      jobCount = route.GetJobs();
   }

   char buffer[20];
   itoa(jobCount, buffer, 10);


   if (jobCount)
      m_jobCount.SetWindowText(buffer);
   else
      m_jobCount.SetWindowText("0");

}
```

GetLockState

GetLockState returns the lock state of the printer queue path. Each printer queue path may be locked by only one route at any instance. Possible values are as follows:

Returned Value	Description
PTRLSE_LOCKED	The printer queue path is locked by the route.
PTRLSE_UNLOCKED	The printer queue path is available for printing.
PTRLSE_WAITING	The route is waiting for the printer queue path to become available.

Associated Interface

IRoute

Prototype

```
GetLockState();
```

Returned Value Type

long

Example

```
void  CATBPTRDlg::GetLockStatus( )
{
   long lockStatus = 0;

   LPDISPATCH lpDispatch = m_routes.Item(Variant);

   if (lpDispatch)
   {
      IRoute route;
      route.AttachDispatch(lpDispatch, TRUE);
      lockStatus = route.GetLockState();
   }

   switch (lockStatus)
   {
      case PTRLSE_UNLOCKED:
         break;

      case PTRLSE_LOCKED:
         break;

      case PTRLSE_WAITING:
   }
}
```

GetName

GetName returns the route name, which is defined by the user in PTR.

Associated Interface

IRoute

Prototype

```
GetName();
```

Returned Value Type

Cstring

Example

```
CStringList&CATBPTRDlg::GetRouteList()
{
   LPDISPATCH lpDispatch;
   m_listRoutes.RemoveAll();

   // Parse out route name into string list
   for (int i = 0, nCount = m_routes.GetCount(); (i <
   nCount); i++)
   {
      CString strRouteName;
      COleVariant Variant = (BYTE) (i + 1);
      lpDispatch = m_routes.Item(Variant);

      if (lpDispatch)
      {
          IRoute route;
          route.AttachDispatch(lpDispatch, TRUE);
          strRouteName = route.GetName();
          m_listRoutes.AddTail(strRouteName);
      }
   }

   return (m_listRoutes);
}
```

GetPrinterTimeout

GetPrinterTimeout returns the printer timeout value. Printer timeout is the length of time (0–3600 seconds) the host will wait for a response from the printer.

Associated Interface

IRoute

Prototype

```
GetPrinterTimeout();
```

Returned Value Type

long

Example

```
long CATBPTRDlg::GetTimeout( )
{
   long printerTimeout = 0;

   LPDISPATCH lpDispatch = m_routes.Item(Variant);

   if (lpDispatch)
   {
      IRoute route;
      route.AttachDispatch(lpDispatch, TRUE);
      printerTimeout = route.GetPrinterTimeout();
   }

   return (printerTimeout);
}
```

GetPrintQueuePath

GetPrintQueuePath returns the printer queue path, which is the communication path between PTR and an output device, such as a printer or a file.

Associated Interface

IRoute

Prototype

```
GetPrintQueuePath();
```

Returned Value Type

CString

Example

```
CATBPTRDlg::GetDefaultPrintQueue( )
{
   CString strDefaultPrintQueue;

   LPDISPATCH lpDispatch = m_routes.Item(Variant);

   if (lpDispatch)
   {
      IRoute route;
      route.AttachDispatch(lpDispatch, TRUE);
      strDefaultPrintQueue = route.GetPrintQueuePath();
   }
   return (strDefaultPrintQueue);
}
```

GetPrintQueueStatus

GetPrintQueue Status returns the status of the printer queue path. For routes that support more than one printer queue path, PTR will display an overall status of the queues. Possible values are as follows:

Returned Value	Description
PTRQSE_AVAILABLE	The printer queue path is available.
PTRQSE_BLOCKCHECK	Data received by the device did not validate correctly.
PTRQSE_BUSY	The printer queue path indicates that the printer is busy. For serial printers this means that clear to send (CTS) is low. **Note:** Currently only serial printers can return this status.
PTRQSE_ERROR	The printer queue path has received an INFOConnect error. The host filter may or may not be attempting to recover.
PTRQSE_NOTAVAILABLE	The printer queue path is not available.
PTRQSE_OFFLINE	The printer queue path indicates that the printer is offline. For serial printers, this means that data set ready (DSR) is low. **Note:** Currently only serial printers can return this status.
PTRQSE_ONLINE	The printer queue path is ready to accept jobs. **Note:** This value alone does not indicate that the route is available. Another route may have the queue locked. Check the value in the Lock State field. See also "GetLockState" on page 31.
PTRQSE_OUTOFPAPER	The printer queue path indicates that the printer is out of paper.
PTRQSE_PAPERJAM	The printer queue path indicates that the printer has a paper jam.
PTRQSE_LOCKED	The printer queue path is currently locked by another host filter.
PTRQSE_PRINTING	The printer queue path is processing a print job.

Associated Interface

IRoute

Prototype

```
GetPrintQueueStatus();
```

Returned Value Type

long

Example

```
void CATBPTRDlg::GetQueueStatus()
{
   long queueStatus = 0;

   LPDISPATCH lpDispatch = m_routes.Item(Variant);

   if (lpDispatch)
   {
      IRoute route;
      route.AttachDispatch(lpDispatch, TRUE);
      queueStatus = route.GetPrintQueueStatus();
   }



   switch (queueStatus)
   {
      case PTRQSE_NOTAVAILABLE:
         m_queueState.SetWindowText("NotAvailable");
         break;

      case PTRQSE_AVAILABLE:
         m_queueState.SetWindowText("Available");
         break;

      case PTRQSE_ONLINE:
         m_queueState.SetWindowText("Online");
         break;

      case PTRQSE_OFFLINE:
         m_queueState.SetWindowText("Offline");
         break;

      case PTRQSE_BUSY:
         m_queueState.SetWindowText("Busy");
         break;
```

```
            case PTRQSE_PRINTING:
               m_queueState.SetWindowText("Printing");
               break;

            case PTRQSE_ERROR:
               m_queueState.SetWindowText("Error");
               break;

            case PTRQSE_OUTOFPAPER:
               m_queueState.SetWindowText("Out of Paper");
               break;

            case PTRQSE_PAPERJAM:
               m_queueState.SetWindowText("Paper Jam");
               break;

            case PTRQSE_BLOCKCHECK:
               m_queueState.SetWindowText("Block Check");
               break;

            case PTRQSE_LOCKED:
               m_queueState.SetWindowText("Printer Locked");
         }
      }
```

GetQueueXlateTable

GetQueueXlateTable returns the printer-to-host translation table file name and path.

PTR uses translation tables to convert characters sent from the printer to the host or from the host to the printer. PTR provides several sample translation tables that you can use as follows:

File name	Description
ANSICAPS.XLT	Converts all characters to their ANSI uppercase equivalent.
ANSILOWR.XLT	Converts all characters to their ANSI lowercase equivalent.
CRTOLF.XLT	Converts all carriage return characters to line feed characters.
MAKECR.XLT	Converts all end-of-line sequences to carriage return characters.
MAKECRLF.XLT	Converts all end-of-line sequences to carriage return and line feed characters.
MAKELF.XLT	Converts all end-of-line sequences to line feed characters.

Associated Interface

IRoute

Prototype

```
GetQueueXlateTable();
```

Returned Value Type

CString

Example

```
CString CATBPTRDlg::GetQueueXlateTable()
{
   CString strXlateTable;

   LPDISPATCH lpDispatch = m_routes.Item(Variant);

   if (lpDispatch)
   {
      IRoute route;
      route.AttachDispatch(lpDispatch, TRUE);
      strXlateTable = route.GetQueueXlateTable();
   }

   return (strXlateTable);
}
```

GetRoutes

GetRoutes obtains the dispatch pointer associated with the current routes from the PTR Server.

Associated Interface

ISystem

Prototype

```
GetRoutes();
```

Returned Value Type

LPDISPATCH

Example

```
CStringList& CATBPTRDlg::GetRouteList()
{
   LPDISPATCH lpDispatch;

   if (m_routes.m_lpDispatch == NULL)
   {
      LPDISPATCH lpDispatch = m_system.GetRoutes();

      if (lpDispatch)
      {
         m_routes.AttachDispatch(lpDispatch, TRUE);
      }
      else
      {
         return (m_listRoutes);
      }
   }
   .
   .
   .

}
```

GetStatus

GetStatus returns the route status, which is the overall status of the route. It accounts for the current INFOConnect host path status and the printer queue path status. Possible values are as follows:

Returned Value	Description
PTRRSE_ACTIVE	Both the host and printer connections are working.
PTRRSE_DISABLED	The route is disabled in the configuration.
PTRRSE_ENABLED	The route is enabled in the configuration.
PTRRSE_FAILED	The host connection failed to initialize or encountered a fatal error while being executed. For information about the status of the host connection, see "GetHostStatus" beginning on page 25.
PTRRSE_WARNING	This status can indicate a lost host connection or particular printer events, such as a busy or offline printer. For more detailed information about the status of the host or printer connections, see "GetHostStatus" beginning on page 25 and "GetPrintQueueStatus" beginning on page 36.

Associated Interface

IRoute

Prototype

```
GetStatus();
```

Returned Value Type

long

Example

```
void CATBPTRDlg::GetRouteStatus()
{
   long routeStatus = 0;

   LPDISPATCH lpDispatch = m_routes.Item(Variant);

   if (lpDispatch)
   {
      IRoute route;
      route.AttachDispatch(lpDispatch, TRUE);
      routeStatus = route.GetStatus();
```

```
    }

    switch (routeStatus)
    {
       case PTRRSE_DISABLED:
          m_printerState.SetWindowText("Disabled");
          break;

       case PTRRSE_ENABLED:
          m_printerState.SetWindowText("Enabled");
          break;

       case PTRRSE_ACTIVE:
          m_printerState.SetWindowText("Active");
          break;

       case PTRRSE_FAILED:
          m_printerState.SetWindowText("Failed");
          break;

       case PTRRSE_WARNING:
          m_printerState.SetWindowText("Warning");
    }
}
```

GetStatusUpdate

GetStatusUpdate returns the status of events for all of the routes.

Associated Interface

IRoutes

Prototype

```
GetStatusUpdate(LPCTSTR szRouteName);
```

Parameter

```
LPCTSTR szRouteName
```

Returned Value Type

CString

Example

```
UINT CPTRSystemTrayDoc::StatusMonitorThread(LPVOID
   lpParam)
{
   CPTRSystemTrayDoc *pSystemTrayDoc =
   (CPTRSystemTrayDoc *) lpParam;

   while (pSystemTrayDoc->m_bMonitoring)
   {
      STATUS Status;
      while (pRouteInfo->GetStatusEvent(Status))
      {
         pSystemTrayDoc->UpdateView(Status);
         Sleep(0);
      }

      Sleep(pSystemTrayDoc->m_lPollInterval);
   }

   delete pRouteInfo;
   return (0);
}

BOOL CATBPTRDlg::GetStatusEvent(STATUS& Status)
{
CString strStatusEvent =
m_PTRRoutes.GetStatusUpdate(strRouteName);
```

```
   if (strStatusEvent.GetLength())
   {
      char *lpStatus = strStatusEvent.LockBuffer();
      memcpy(&Status, lpStatus, sizeof(STATUS));
      strStatusEvent.ReleaseBuffer();
      return (TRUE);
   }
   else
   {
      return (FALSE);
   }
}
```

GetSubmitTime

GetSubmitTime returns the timestamp (time submitted) of the current start of the document (SOD) message being processed by the printer queue path.

Associated Interface

IRoute

Prototype

```
GetSubmitTime();
```

Returned Value Type

long

Example

```
void CATBPTRDlg::GetSubmitTime()
{
   long submitTime = 0;

   LPDISPATCH lpDispatch = m_routes.Item(Variant);

   if (lpDispatch)
   {
      IRoute route;
      route.AttachDispatch(lpDispatch, TRUE);
      submitTime = route.GetSubmitTime();
   }

   char buffer[20];
   itoa(submitTime, buffer, 10);

   if (submitTime)
      m_printTime.SetWindowText(buffer);
   else
      m_printTime.SetWindowText("0:00");
}
```

Item

The Item function supplies lpDispatch, the pointer associated with the current routes.

Associated Interface

IRoutes

Prototype

```
Item(const VARIANT& Routes);
```

Parameter

```
const VARIANT& Routes
```

Returned Value Type

LPDISPATCH

Example

The following example shows how this function is used every time you make a call.

```
void CATBPTRDlg::GetSubmitTime()
{
   long submitTime = 0;

   LPDISPATCH lpDispatch = m_routes.Item(Variant);
```

Sample Client Application

A

In This Appendix

This appendix explains how to use the sample PTR OLE API client application. The following section is included:

Using the Sample Client Application

A sample client application that uses the PTR OLE API, ATSTCP32.DLL, is provided in the following directory on the Millennium I with PTR System Tray CD:

\\32\PDK\OLEAPI\SAMPLE

The following files are part of ATSTCP32.DLL:

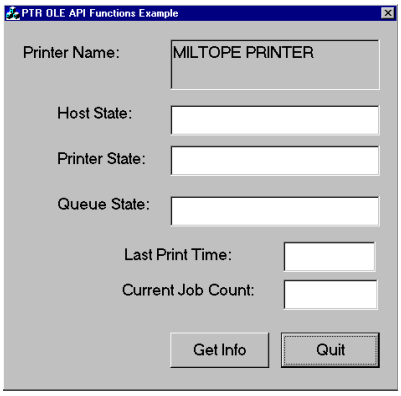
- ATBPTR.APS
- ATBPTR.CLW
- ATBPTR.CPP
- ATBPTR.DSP
- ATBPTR.H
- ATBPTR.PLG
- ATBPTR.RC
- ATBPTRDLG.CPP
- ATBPTRDLG.H
- PTRAPI.CPP
- PTRAPI.H
- RESOURCE.H
- STDAFX.CPP
- STDAFX.H

ATBPTR.DSP is a project file generated in Microsoft Developer Studio 97(Visual C++ 5.0); however, you can access the source and header files in any text editor.

Description

ATSTCP32.DLL monitors the status of the Miltope Printer route by calling functions from the PTR OLE API. To obtain the status, you must click the GetInfo button in the PTR OLE API Function Example dialog box.

The dialog box for ATSTCP32.DLL is shown as follows:



PTR OLE API Functions Called

The following table lists each field in ATSTCP32.DLL that uses a PTR OLE API function. It also provides the name of the function associated with each of those fields:

Field	Associated Function
Host State	GetHostStatus
Printer State	GetLockState
Queue State	GetPrintQueueStatus
Last Print Time	GetSubmitTime
Current Job Count	GetJobs
Get Info	GetStatus

ATBPTRDLG.CPP shows how to call the functions listed above.

Running ATSTCP32.DLL

You must do the following before you can run ATSTCP32.DLL:

- Copy ATSTCP32.DLL to a local drive from the following location on the Millennium I with PTR System Tray CD:

  \\32\PDK\OLEAPI\SAMPLE

- Create a PTR route named Miltope Printer in INFOConnect Manager.

PTRState

B

In This Appendix

This appendix explains the new utility PTRState.exe. The following sections are included:

PTRState.exe

PTRState.exe is a utility that can determine whether PTR is running. PTRState.exe accepts a single command line parameter, a timeout interval in seconds. If no input is given, a default timeout interval of 20 seconds is used.

The utility uses the PTR OLE API to test whether PTR is active. It checks the status of each of the enabled routes to ensure that they have been started, and that the corresponding INFOConnect paths were successfully opened. If PTR is down, then PTRState waits (until the timeout interval elapses) for PTR to come up. PTRState will return immediately once PTR is up and ready. You can verify the state visually, as PTRState outputs a single line to indicate the state of PTR. The possible states can be inspected programmatically via a return code.

The possible return code values are:

Value	Description
0	Success. PTR is up and ready.
1	Timeout. PTR is down and the timeout interval expired.
2	Failed. There is a problem with at least one PTR Route.
3	BadInput. The command line input was not recognized.

If you run the utility with an input parameter of "0", PTRState.exe checks the state of PTR only once, and then immediately reports the result. In that case, it does not wait for PTR to come up.

No changes to PTR or the configuration are needed. PTRState.exe can be placed in any desired directory location. If you can, get a usage output by specifying a command line input of: ?, /?, -?, Help, /Help, or -Help. If the input is unrecognized, a usage statement will be displayed, and no PTR checking is done.

Glossary

Class ID (CLSID) The unique identifier for the PTR COM server.

host filter A link between the host and an output device that initializes the host connection, manages data for the output device, and sends the data to the output device. Host filters are specific to the host and output device.

INFOConnect Manager The application that runs in the background each time you run an INFOConnect product and that controls interaction between accessories and transports.

You can also use this application to create, modify, and delete paths; set administrator and user-level security; control user access to configuration; and set user preferences.

path A named set of configuration options that define an INFOConnect communication connection between a PC and a host (host path) or between PTR and an output device (printer queue path). A path consists of a path template and other configuration data associated with the library(s) listed in the path template.

PTR System Tray An INFOConnect PTR application that monitors and displays the status of INFOConnect PTR routes. The PTR System Tray displays not only status information about the route as a whole but also information about the individual route components, such as the host filter and the host path. The PTR OLE API provides the basis for the PTR System Tray.

Program ID (ProgID) The programmatic ID used to create the PTR COM server.

route A named set of configurations that describe a complete host to peripheral connection, consisting of the following three parts: host path, host filter, and printer queue path.

type library file A binary file that gives your client application runtime access to the functions within the PTR interfaces.

Index

E

F

G

Q

R

S

T