««

Introduction

The IATE API provides an interface which enables an applications programmer to communicate with an Airline Reservation system through an InnoSys Gateway.

The API is implemented as a set of C calls and supplied as either a library to be linked with the users application code for Unix & DOS, or as a 16- or 32-bit dll for Windows. Messages are passed between the API and the Gateway via TCP/IP. This document describes the concepts behind the API and some of the interactions between the API, the gateway, and the operating system. For a precise description of the API, please refer to the “API Reference Manual”.

How the components work together

The API is a layer which enables an application program to communicate with an airline system via an InnoSys Gateway. The API and the Gateway communicate using sockets. There is a well-defined message protocol between the API and the Gateway, but this layer is hidden from the application by the API.

The API always links to the “ALC” gateway, regardless of whether the host line is ALC, X.25, or TCP/IP. The ALC gateway may communicate with the communication line directly (as it does in the case of an ALC connection), or it may do so via another application (as it does in the case of an X.25 connection). At the API level, data coming from any kind of a line is identical.

If the gateway being used is a Sun or an NT gateway, the API links directly to the gateway. If the gateway being used is a Mac gateway, the API links to a utility called IateTCP which runs on the Mac. The basic API calls for linking to a Mac or a Unix Gateway are identical. However, there are a some details which are different and they are discussed later in this document.

The Unix and the NT ALC gateway are both named iate_server. If the line is an X.25 line, iate_server links to the X.25 gateway (x25gate), and the X.25 gateway handles the communications with the communications card.

All information about the configuration of the host line (e.g. IAs and TAs) is configured at the ALC gateway, as is the information which allows API programs to link to and communicate with a CRS using the InnoSys Gateway(s). The X.25 gateway configuration file contains only information which is specific to the X.25 communication layer.

API programs link to objects or groups. An object is simply a name which is used by the ALC gateway to identify a particular IA and TA. A group is a collection of objects. Objects and groups are configured at the ALC gateway.

Data passed over the API

Data coming from and going to the API is in ASCII. The translation to and from alc (or padded alc, or ebcdic, or whatever is appropriate) is handled in the ALC gateway. The API includes an option to encode data passing between a client and the gateway. (Versions of the API either support or do not support encoding. The encoding option is enabled or disabled at the gateway.) The IateRead call is used to receive data from the host; the IateWrite call is used to send it. The IateRead call returns two buffers: the data buffer and the control block buffer. The data buffer contains only the data part of the message (the C1, C2 and EOM characters are stripped from the message). The control block buffer contains the screen positioning characters (C1 & C2), the End of Message character, the CCC, and the MORE character.

In addition to host data, additional information (such as the host status) can be obtained from the API using the IateControl call.

Other capabilities of the IATE API

In addition to communication with a CRS, the InnoSys API/Gateway environment provides two other services. These are Peer-to-Peer communications and a facility called TA sharing or intercept mode.

Peer-to-Peer allows an application using the API to communicate with another application via the ALC Gateway. Essentially, it allows two objects to communicate directly with each other. For example, the screen print function of the InnoSys terminal uses this facility.

The shared TA mechanism allows one application to send and receive on the behalf of another. A sharing application can intercept (or divert) data which flows between the shared object and the host. For example, this facility could be used to monitor traffic on a given TA. It may also be used to perform pre- and/or post-processing on data to and from the host. Examples are provided in the API and as part of API package. See Appendix F of the API reference Manual for further information.

Basic structure of a program using the API

There are seven calls into the API. These are:

- IateStart
- IateOpen
- IateRead
- IateWrite
- IateClose
- IateStop
- IateControl

See the API Reference Manual for detailed descriptions of each of these calls.

The IateStart and IateStop calls "start" and "stop" the API. The IateStart call initializes the API and in the case of the windows APIs initializes the TCP stack. As mentioned above, IateWrite is used to send data to the host and IateRead is used to receive data from the host. The IateControl call provides a collection of services which can be used to control the behaviour of the API and control some aspects of the ALC Gateway's behavior. In addition, Peer-to-Peer communications and the Shared TA mechanism are controlled by IateControl calls.

IateStart is the first call to the API that a program must make. It is an error to call IateStart again without first calling IateStop. After the API is started, an application issues the IateOpen call to establish a connection with the host, which is used for sending and receiving data. Once a connection is opened, an application usually goes into a loop sending, receiving, and processing data using IateRead and IateWrite. Finally, the program should issue an IateClose to close the connection with the gateway, and an IateStop to stop the API.

An application may open a connection with any number of objects by issuing an IateOpen for each desired connection. On termination, an IateClose should be issued for each IateOpen.

There are many examples of API calls in the API documentation. A good starting point for development is the sample program testterm.c, which is distributed with the API package.

Different types of links

There are a number of different ways that an application can link to an object. These are:

- APILinkToName
- APILinkToTa
- APILinkToDyCrt
- APILinkToDyPrt
- APIinterceptName
- Peer-to-Peer "links"

Objects are configured at the ALC gateway. The definition of each object includes an IA and TA, an object type, an object name, and optionally a group name.

The object type is one of:

TERMINAL
PRINTER
TERMINAL_API
PRINTER_API

APILinkToName is used to link to a particular object by name or to any unused object in a group. APILinkToName will link to any type of object.

APILinkToTa is used to link to a particular IA/TA pair. If the same IA/TA pairs occur in multiple gateway configuration files, the argument from the gateway's "PORT_NAME" parameter should be supplied so that the Gateway only looks for a matching IA/TA on the desired host line. If a "PORT_NAME" isn't provided the Gateway stops searching for a match at the first matching IA/TA it encounters. APILinkToTa will link to any type of object. (Note -

if a Macintosh gateway is being used, APILinkToTa only works when issued from Macintosh clients; if a Sun or NT gateway is being used, APILinkToTa only works when issued from Unix or Windows clients.)

APILinkToDyTa and APILinkToDyPrt are similar to APILinkToName except that APILinkToDyTa only links to objects of type TERMINAL_API and APILinkToDyPrt only links to objects of type PRINTER_API. APILinkToDyTa and APILinkToDyPrt can be used to link to specific objects or to any unused object in a group.

APIInterceptName is used to link a sharing application to a specific object. (This feature is not available if using a Macintosh gateway and an NT or Sun client.) An object name must be used; group names are invalid. The object may be of any type. See Appendix F of the API Reference Manual for information on sharing TAs.

The connection between peers is not really a link; the routing is done on a per message basis using the IateControl calls:

APISendApplMsg
APIGetApplMsg
APIQuerryApplMsg

Peer-to-peer messages contain a command, some routing information and, optionally, some data. There is information about these commands in the API documentation, and also in the header files. The only peer-to-peer command which is used by the gateway is PTRnotExist, which is the error returned when the object name of the peer is not recognized at the gateway. Users may define their own protocol.

Special requirements for printer objects

Since some host protocols require that traffic to a printer be rejected if the printer is not ready, the ALC gateway distinguishes between printer objects and other objects. The Gateway will not pass data on connections to printer objects unless the application has informed the Gateway that the printer is available. An application tells the Gateway about the state of a printer by using the APIPrinterStat IateControl command.

Since most hosts have some sort of segment acknowledgment scheme, the Gateway also needs information from the application which will allow it to generate the appropriate ack. Therefore an application should acknowledge each segment (positively or negatively) with an APISendAck IateControl command.

Getting diagnostic information from the API

There are two types of diagnostic information available from the API. One type is provided via return codes and global error variables. In addition, there is a great deal of trace information built into the api. The diagnostic tracing is turned on and accessed in different ways for each API.

If an API call succeeds, it will return a value greater than or equal to zero. If it fails, it will return a negative value, usually less than or equal to -2000. API error codes are listed in the "API reference manual" and in the "Problem Solving" manual.

In the Unix environment additional information about error conditions is accessible through the global variable APIerrno. APIerrno is used to hold error codes from failed calls to the operating system. For example, if a write fails in the API, APIerrno will be set to the value of t_errno. APIerrno values tend to be highly context sensitive; APIerrno may be set to the value of the unix variable errno, or in the case of failures on socket calls it may hold the value of t_errno, or even the value returned by the system call t_look.

In windows environments additional information is available through the IateControl call APIGetLastWsError. APIGetLastWsError returns a structure which contains the ordinal value of the Winsock call on which the error occurred and the value returned by WSAGetLastError, the "Windows Sockets error". Winsock error constants are listed in the Windows Sockets documentation which is available with various Microsoft development environments.

In the Unix environment, the following IateControl calls are used to control tracing in the API:

APISetApiDebug
APISetApiLogging
APISetDebugOut

APISetApiDebug sets the diagnostic level. This controls the amount and type of diagnostic information which is displayed. APISetApiLogging is used to instruct the API to log the diagnostic information. APISetDebugOut is used to pass the API an alternate routine to use for displaying handling diagnostic information. There are examples of all these calls, as well as an alternate debugging output routine, in testterm.c

The 16-bit windows API (iatedll.dll) does not allow applications to display or intercept tracing information directly. Instead a utility called apiwatch is provided to monitor the API dll.

The 32-bit windows API (iate32.dll) supports APISetApiDebug and APISetApiLogging, but not APISetDebugOut. A future version of apiwatch will provide for monitoring the 32-bit API dll.

Differences between API platforms

The primary difference between the Unix APIs and the Windows APIs is that for the Windows API, the startcode returned by IateStart must be passed to IateStop. There are also some differences in the way that diagnostic information is made available from the API. These are discussed above.

Differences between Gateways

The primary difference between the Mac gateway and the Unix gateways is that the Mac gateway requires heartbeats and the Unix gateways do not. However, the Unix gateways can be configured to require heartbeats, and the Mac gateway can be instructed by the API to not require heartbeats on specific connections.

Heartbeats are messages which are used by the Gateway to ensure that a connected application is still running. The Mac gateway expects to receive a heartbeat message for each terminal object in use at least every 60 seconds. If it does not, it closes the connection and frees the object.

Miscellaneous information and tips

- For WinIATE running on Windows 3.1, an application can specify an IP address instead of a host name; also an application can specify a port number instead of a service name.

- For WinIATE running on Windows 95 or NT an application can specify an IP address instead of a host name. However the Winsock call which is used to resolve port numbers doesn’t work under Windows 95 or NT, so an application may only specify a port in the services file.

- For the Unix curses based terminal, (iate) error 133 usually means that the file htable is missing. Make sure that htable has been copied to .htable in the users home directory.

- Changes in modem control lines for X.25 connections will only be reported to the API as host status changes if +CTS, +DCD, and +DSR and +FLAGS are configured at the X.25 gateway. This is not the default, and is not usually necessary.

- An X.25 line is not fully operational until the RESTART_COMPLETE message is displayed in the X.25 Gateway’s diagnostic trace.

- The gateway license limits the total number of objects of a given host type which can be configured at the gateway. If the gateway is being used to connect to multiple hosts, the license needs to be large enough to allow for all the objects in all the configuration files. If the total number of objects configured exceeds the license, the remainder will be ignored.

- Groups may span configuration files at the ALC gateway. The gateway considers all the objects to be in one big pool.

- Peers can only communicate with other peers which are configured on the same gateway. This means that an application using an object on one gateway can not communicate using the peer-to-peer mechanism with an object configured on another gateway.

IATE™

API Fundamentals

Background Information
on the IATE API

InnoSys
INCORPORATED

InnoSys Incorporated
3095 Richmond Parkway, Suite 207
Richmond, CA 94806-1900
(510) 222-7717 Voice
(510) 222-0323 FAX

««

Contents