UNISYS

ClearPath Dorado Servers

Application Integration Services Installation and Programming Guide

AIS for Dorado Release 4.0

April 2017 8230 0815-003

Contents

Figures

Tables

Section 1
Getting Started

This section introduces Unisys ClearPath Application Integration Services.

1.1. Documentation Updates

This document contains all the information that was available at the time of publication. Changes identified after release of this document are included in problem list entry (PLE) 19175731. To obtain a copy of the PLE, contact your service representative or access the current PLE from the product support Web site:

http://www.support.unisys.com/all/ple/19175731

Note: *If you are not logged into the product support site, you will be asked to do so.*

1.2. About Application Integration Services

The Unisys ClearPath Application Integration Services enables application developers to use a familiar development environment—the Microsoft .NET Framework—to create applications that access files in the ClearPath environment. Application Integration Services provides access to the logical IO subsystem of the OS 2200 system so the .NET application can read and write files on the OS 2200 system.

Application developers can still use all the familiar capabilities of the .NET environment to access files in a Windows environment. Application Integration Services extends the existing capabilities of the .NET environment to files in the ClearPath environment. This enables .NET programmers to create applications that, for example, read files on the OS 2200 system and write them to a Windows system.

Figure 1–1 illustrates the Application Integration Services.

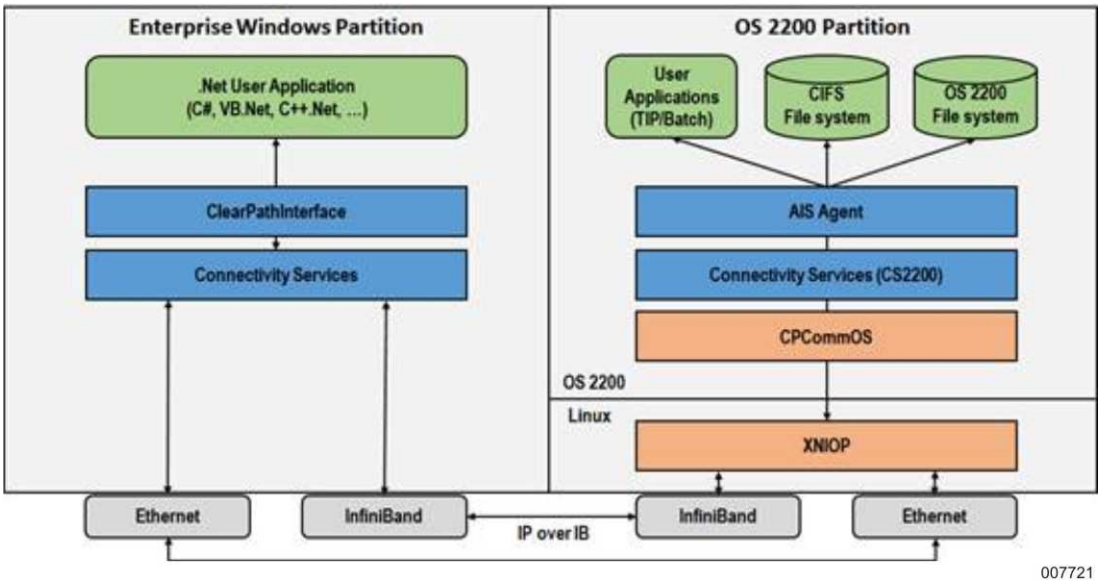


Figure 1–1. Application Integration Services

Application Programming Interfaces

Application Integration Services provides an application programming interface (API) that enables users to:

- Securely connect to a ClearPath OS 2200 environment using Connectivity Services. Either OS 2200 credentials or Windows credentials authenticate a user before a connection is established.
- Access the files in the ClearPath OS 2200 environment using interfaces similar to those they would use to access files in a Windows environment.
- Call TIP/HVTIP transactions. Transactions that use Display Processing System (DPS) forms can be called using the DPS connector interface.

File System Structure

Application Integration Services provides access to files using CIFS, which uses a POSIX-like hierarchical directory structure that supports long file and directory names.

CIFS provides access to all OS 2200 files and elements through a special directory: /os2200. During CIFS initialization, all OS 2200 files (except certain system files) are automatically added to the /os2200 directory. CIFS monitors changes to the OS 2200 file environment and updates the /os2200 hierarchy accordingly. Refer to the *CIFS for ClearPath OS 2200 User, Programmer, and Administrator Reference Manual* for more information about how CIFS manages files.

Application Integration Services uses CIFS on ClearPath Dorado servers only. It does not implement a server message block (SMB) protocol or require SMB access to ClearPath Dorado servers from a Windows server or workstation. The AIS Agent invokes CIFS using the CIFS programmatic interfaces on a ClearPath OS 2200 partition.

Components

Application Integration Services includes two main components:

- AIS Agent

  The AIS Agent runs on the OS 2200 system. It provides the entry point into the OS 2200 environment for Windows clients.

- AIS Client API

  The AIS Client API is deployed on an Enterprise Windows partition image on the ClearPath Forward fabric or a client workstation. It provides the implementation for the .NET classes. It also establishes and manages the connection to the ClearPath system.

1.3. Connectivity Services

Connectivity Services is an interface for OS 2200 network applications (sometimes called agents). The AIS Agent and AIS Client API use Connectivity Services to communicate.

See the *Connectivity Services Installation and Programming Guide* (8231 0673) for more information.

1.4. How a Developer Uses Application Integration Services

An application developer uses the Microsoft .NET Framework, running on his or her workstation, to create a standard .NET application.

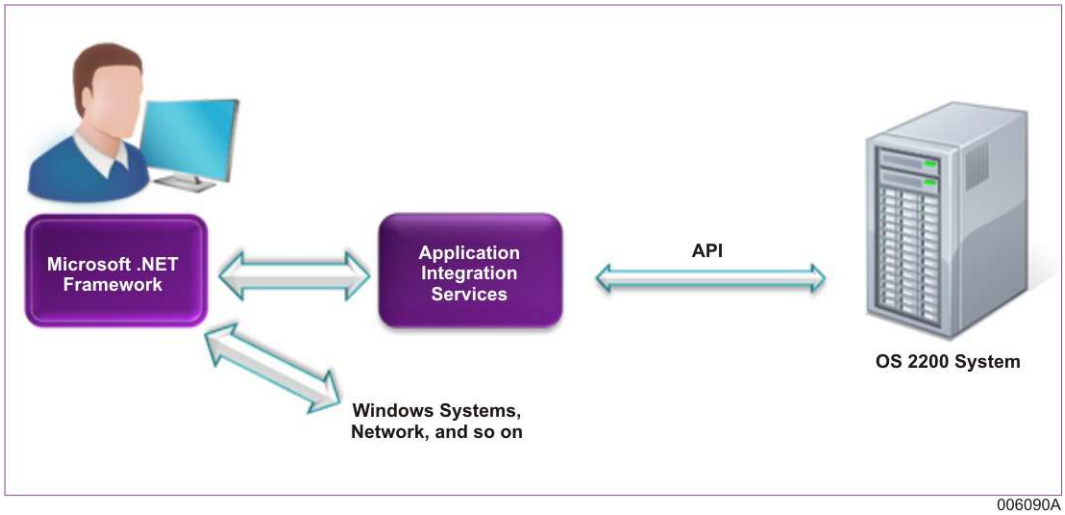


Figure 1–2. Using Application Integration Services

When the developer needs the application to access files on a ClearPath system, the developer uses the API of Application Integration Services.

The developer uses the standard capabilities of the .NET Framework for all other functions of the application, such as accessing the network or accessing the file system on a Windows system or an Enterprise Windows partition image on the ClearPath Forward fabric.

When the application is complete, the developer or an administrator deploys the application on any Windows system with suitable connectivity.

Note: *For information about the supported development and runtime environments, see* *Appendix A, Development versus Runtime Environment.*

1.5. Restrictions

The following restrictions apply to Application Integration Services:

- The OS 2200 user-id used when starting the AIS Agent background run cannot be used to open an AIS client connection. This will result in an authentication error.
- The account and project-id specified when opening an AIS client connection cannot be changed when opening a subsequent AIS client connection with the same user-id.

  Note: *If it is necessary to change the account or project-id, stop and restart the AIS Agent background run.*

1.6. Operational Considerations

The following operational considerations apply to Application Integration Services:

- The OS 2200 user-id used when opening an AIS client connection must have at least one account defined in the user-id's security record.
- The OS 2200 user-id used when opening an AIS client connection must be made known to CIFS before a connection can be opened. A user-id can be made known to CIFS by opening a demand session with the user-id and using CIFSUT to perform a command, such as 'pwd' or 'ls'.

1.7. Resources for the Application Developer

Application Integration Services Product Documentation

- *ClearPath Application Integration Services Configuration Utility Help* (8229 8548)

  This Help file describes how to configure logging and tracing for Application Integration Services.

- *ClearPath Application Integration Services API Reference Guide* (8222 4239)

  The *API Reference Guide* is a Windows Help (CHM) file that the application developer uses as he or she creates the .NET application. It describes all public classes and methods in the OS 2200 API of Application Integration Services. For information about the OS 2200 specific classes and their members, refer to the section titled "Unisys.OS2200.Interface Namespace".

- *ClearPath Dorado Servers Connectivity Services Installation and Programming Guide* (8231 0673)

  This guide describes the OS 2200 Connectivity Services, which provides an interface for OS 2200 network applications. The AIS Agent and AIS Client API use Connectivity Services to communicate.

Microsoft .NET Framework

Use the following Web sites to learn about .NET application development:

- Visual Studio

  http://msdn.microsoft.com/en-us/vstudio/aa496123

  This Microsoft web site contains resources to help you build applications using Visual Studio. The .NET Framework Development Guide on the Microsoft .NET Framework page describes how to create and deploy .NET applications.

- .NET Development in the MSDN Library

  http://msdn.microsoft.com/en-us/library/ff361664.aspx

  The .NET Development section of the MSDN library contains documentation for .NET Framework 4.5 along with a complete API reference.

ClearPath Information

A .NET application developer may find the following documents useful if he or she is unfamiliar with OS 2200 systems:

- *CIFS for ClearPath OS 2200 User, Programmer, and Administrator Reference Manual* (7859 6137)

  This manual describes CIFS for ClearPath OS 2200, an implementation of the Common Internet File System (CIFS) specification that allows access to OS 2200 files through standard Internet protocols.

- *Communications Interface for Transaction Applications (CITA) Configuration and Operation Guide* (7862 6470)

  This manual describes CITA, which provides an interface used to call TIP/HVTIP transactions.

- *Program-Callable FURPUR (PCFP) Programming Reference Manual* (7830 9796)

  This manual describes the programmatic API used to perform FURPUR functions.

ClearPath Forward Fabric Information

See the Unisys Product Support Site for information about the ClearPath Forward fabric:

http://epas1.rsvl.unisys.com/common/epa/DocumentationLibraries.aspx?PLA=FWD&NAV=FWD

Section 2
Installing Application Integration Services

Installing Application Integration Services requires you to install an OS 2200 component and a Windows component.

2.1. Installation and Configuration Overview

To install and configure Application Integration Services, complete the following tasks:

1. Make sure the OS 2200 and Windows systems satisfy the installation prerequisites. See 2.2 Installation Prerequisites.
2. Install and configure Connectivity Services. See the *Connectivity Services Installation and Programming Guide* (8231 0673) for more information.
3. Install the AIS Agent. See 2.3 Installing the AIS Agent.
4. Configure the AIS Agent. See 3.1 Setting Security Privileges for the AIS Agent through 3.2 AIS Agent Configuration Parameters.
5. Install the Windows component. See 2.4 Installing the Windows Component.
6. Configure Application Integration Services. See Section 4, AIS Client Administration.

   Use the **Manage Host Connections** tab of the Configuration Utility to specify the information Application Integration Services needs to establish a connection to ClearPath systems. See 4.2 Specifying Host Connection Information.

2.2. Installation Prerequisites

Before you install Application Integration Services, make sure the hardware is installed and configured properly, which includes configuring the system for networking services.

Table 2–1. Installation Prerequisites

Category	Prerequisite
ClearPath OS 2200 operating environment	ClearPath OS 2200 Release 16.0 and later. The following OS 2200 products must be installed and running: • CIFS for ClearPath OS 2200 • User Authentication (FLEX) for OS 2200 local authentication • Messaging Integration Services (NTSI) for Windows network authentication (NTLM) • Program-Callable FURPUR (PCFP) • UCS Runtime System (URTS) • Connectivity Services (CS2200) Check the Product Support site (www.support.unisys.com) for any interim corrections (IC) that need to be applied.
Windows server operating environment	Windows Server 2008 R2 or Windows Server 2012
Windows desktop operating environment	Windows 7, Windows 8, Windows 8.1, and Windows 10 Microsoft .NET Framework version 4.5 or later

2.3. Installing the AIS Agent

Use SOLAR to install the AIS Agent. Refer to the *Software Products Installation Guide* for information about installing products with SOLAR.

The AIS Agent has four installation modes provided by Unisys: mode A, B, C, and D. During the installation process, SOLAR creates the files listed in Table 2–2 on your system from the tape Unisys provides.

Table 2–2. Installation Modes

Mode	Files
Mode A	SYSLIB*AISAGENT$A
	SYSLIB*RUN$.AISAGENTA
	SYS$INSTALL$*AISCLIENT$A
Mode B	SYSLIB*AISAGENT$B
	SYSLIB*RUN$.AISAGENTB
	SYS$INSTALL$*AISCLIENT$B

Table 2–2. Installation Modes (cont.)

Mode	Files
Mode C	SYSLIB*AISAGENT$C
	SYSLIB*RUN$.AISAGENTC
	SYS$INSTALL$*AISCLIENT$C
Mode D	SYSLIB*AISAGENT$D
	SYSLIB*RUN$.AISAGENTD
	SYS$INSTALL$*AISCLIENT$D

2.4. Installing the Windows Component

Use the following procedure to install the Windows component:

1. Copy the CPI.msi or CPIx64.msi file from the OS 2200 environment to the Windows environment. These *.msi files are located in the SYS$INSTALL$*AISCLIENT$*x*, where *x* is the installation mode.

   In a 32-bit Windows environment, copy the CPI.msi file. In a 64-bit Windows environment, copy the CPIx64.msi file.

2. Double-click the CPI.msi or CPIx64.msi to start the installation process.
3. Follow the on-screen instructions.
4. Once the installation is complete, you must create a configuration file for your application to use. See 4.2 Specifying Host Connection Information for the details.

2.5. Uninstalling Application Integration Services

If needed, use the following methods to uninstall Application Integration Services:

- Standard ClearPath OS 2200 procedures to uninstall the OS 2200 component of Application Integration Services
- Standard Windows procedures to remove the Windows component.

Section 3
AIS Agent Administration

This section provides information on how to configure the AIS Agent and control it using console commands.

3.1. Setting Security Privileges for the AIS Agent

This section describes how to set up the OS 2200 system user that is designated to start the AIS Agent.

Set up the OS 2200 system user-id for starting the AIS Agent with the following privileges:

- SSAUTHNTICAT

  Allows the AIS Agent to access OS 2200 user records.

- SSLEVEL

  Ensures that the AIS Agent runs at the requested priority. At start-up time, the AIS Agent sets its executing (switching) priority based on the value of the agent_priority configuration parameter (see 3.2.4 agent_priority).

- SSCONSOLE

  Allows use of the Console Handler capability of the AIS Agent.

- SSSWITCHUSER

  Allows an AIS Agent worker activity to impersonate the user-id used to open an AIS client connection.

3.2. AIS Agent Configuration Parameters

The following subsections describe the configuration parameters used to control the AIS Agent. The configuration parameters for the AIS Agent are located in the following element:

```
SYS$LIB$*AISAGENT$x.AIS$CONFIG
```

where *x* is the installation mode.

Note: *The original configuration is saved in AIS$CONFIG/ORIGINAL so you can always restore the default configuration.*

3.2.1. agent_listens_on

This parameter specifies the IP address the AIS Agent listens on to create new AIS client connections. The default value is zero, which indicates that Connectivity Services should listen on all IP addresses defined for access to the OS 2200 system.

Default: 0

3.2.2. agent_locale

This parameter specifies the locale to be used by the AIS Agent. The locale typically contains a 2-character language code, and optionally followed by an underscore and a 2-character country code.

Default: en_US

Range: 1 to 5 characters

3.2.3. agent_name

This parameter specifies the name associated with an AIS Agent. This is the name that is displayed in log messages and console output.

Default: AISAGENT

Range: 1 to 16 characters

3.2.4. agent_priority

This parameter specifies the executing (switching) priority for the AIS Agent. At execution, after processing and validating the configuration parameters, the agent requests that the OS 2200 operating system set the executing (switching) priority of the agent to the specified level.

Table 3–1 describes the allowed values for the agent_priority configuration parameter, which correspond to the default switching level names assigned by the Exec during system configuration, plus an additional value of LEVEL$. Your system might not be configured with the BATCH, DEMAND, or USER switching level names. BATCH and DEMAND exist only if the user switching levels are split into batch and demand groups during system configuration. USER exists only if not split into those groups.

Table 3–1. Allowed Values for agent_priority Configuration Parameter

Value	Description
TXN	Indicates the AIS Agent should operate at a transactional level. This is the recommended value.

Table 3–1. Allowed Values for agent_priority Configuration Parameter (cont.)

Value	Description
Batch	In most cases, this might provide the AIS Agent with adequate system resources. You might, however, encounter variability in AIS Agent response times if system resources become tight and batch level runs receive fewer system resources.
Demand	Using this value requires additional knowledge of how the OS 2200 system is configured at your site. It enables the AIS Agent's executing priority to follow your OS 2200 system resource and performance strategy.
User	Using this value requires additional knowledge of how the OS 2200 system is configured at your site. It enables the AIS Agent's executing priority to follow your OS 2200 system resource and performance strategy.
LEVEL$, *nnnnnnnnnnn*	If agent_priority is set to a value that is not supported by the host system, the host selects a level of executing (switching) priority for the AIS Agent, which might or might not be adequate. In this case, setting agent_priority via LEVEL$ utilizes ER LEVEL$ to set a specific switching priority for the AIS Agent. The format is as follows: `agent_priority=LEVEL$,0`*nnnnnnnnnnn*`;` where *nnnnnnnnnnn* is the octal value used as the ER LEVEL$ input parameter (type indicator in bits 0 to 5, group index in bits 6 to 17, relative level in bits 18 to 35). For example, a value of LEVEL$,0600001000001 would set the AIS Agent to operate at some transactional priority. However, the actual switching priority established depends on the various switching groups and levels set up by the system administrator. You normally operate the AIS Agent at either the transaction or batch executing priority.

3.2.5. client_receive_timeout

This parameter specifies the number of seconds the AIS Agent waits to receive a request from an AIS client.

Default: 600

Note: *A value of zero indicates an infinite wait.*

3.2.6. comapi_mode

This parameter specifies the COMAPI mode used by the AIS Agent when connecting to Communications Interface for Transaction Applications (CITA).

Default: A

3.2.7. comapi_loopback

This parameter specifies the loopback address used to open a connection between the AIS Agent and Communications Interface for Transaction Applications (CITA).

Default: 127.0.0.1

3.2.8. comapi_receive_timeout

This parameter specifies the number of seconds the AIS Agent waits for a response from Communications Interface for Transaction Applications (CITA).

Default: 60

3.2.9. cs_mode

This parameter specifies the Connectivity Services (CS2200) installation mode used by the AIS Agent.

Default: A

Range: A – D

3.2.10. host_port

This parameter specifies the listening port number used for cleartext communications.

Default: 43745

Range: 1 – 65535 (Must be a unique value on the OS 2200 system)

3.2.11. keyin_id

This parameter specifies the Keyin ID used to execute AIS Agent console commands.

Default: RUNID of the AIS Agent

Range: 1 to 8 alphanumeric characters, starting with a letter

3.2.12. log_console_output

If ON, this parameter indicates that all console output is written to the log file in addition to the console. Otherwise, output is sent only to the console.

Default: ON

Range: ON I OFF

3.2.13. max_activities

This parameter specifies the maximum number of AIS client connections supported by an AIS Agent.

Default: 50

Range: 10 – 400

3.2.14. max_queued_clients

This parameter specifies the maximum number of outstanding AIS client Open requests that can be queued. This value cannot exceed the value specified for max_activities.

Default: 50

Range: 1 – 350

3.2.15. secure_host_port

This parameter specifies the listening port number used for secure (SSL/TLS) communications.

Default: 43746

Range: 1 – 65535 (Must be a unique value on the OS 2200 system)

3.3. Starting the AIS Agent

The AIS Agent runs as a batch program on the OS 2200 system. To start the AIS Agent from the operator console, use the following ST keyin:

```
ST AISAGENTx
```

where *x* is the installation mode.

The output from the AIS Agent background run is directed to a breakpoint file AIS*x**AIS$PRINT, where *x* is the installation mode.

3.4. AIS Agent Console Commands

When the AIS Agent is active and operational on the ClearPath OS 2200 system, a number of commands are available to an administrator for dynamically viewing or controlling the operation of the agent. The administrator does this using the Console Handler capability of the AIS Agent. The administrator can direct Console Handler commands to the agent from any workstation in terminal emulation mode.

Within the running AIS Agent are one or more activities called workers. Each worker performs the work of an AIS client connection on the OS 2200 system. The Console Handler contains a number of commands to inquire about workers or the agent itself, and adjusts how you want it to operate. The commands can be grouped into the following categories:

- Displaying AIS Agent information
- Controlling the AIS Agent
- Stopping the AIS Agent
- Stopping the workers
- Controlling the log file

The format for sending a console command to the server is as follows:

```
@@cons keyin_name command
```

where:

keyin_name

is the value specified for configuration parameter keyin_id (see 3.2.11 keyin_id), unless keyin_id is not specified or is RUNID (the default). In these cases, *keyin_name* is the original run ID of the agent background run.

command

is the console command being sent to the agent. See the allowed syntax for each command in the following tables.

For example, if RUNID is the value of configuration parameter keyin_id, and the AIS Agent's original run-id is AGNTA, the following console command could be submitted:

```
@@cons agnta status
```

The Console Handler commands are case insensitive, and extra spaces are allowed in the commands. File names specified in certain commands are restricted to one of the following formats:

```
qualifier*filename
*filename
Filename
```

When you issue an @@CONS command to the AIS Agent, you should always receive some output to your terminal. Output is also placed in the log file for all commands except the display commands. For those commands, the output is placed in the AIS Agent log file if you specify the console command

```
@@cons agnta set log console output=on
```

If you do not specify this command, the console uses the value of the log_console_output configuration parameter (which has a default value of ON).

3.4.1. Console Commands for Displaying AIS Agent Information

The following table describes the commands that report information about the AIS Agent at the time the command is executed. They do not affect the operation of the agent.

Command Syntax	Description
`CONFIG` or `CONFIGURATION`	Displays the operational configuration parameter values.
`FILENAMES`	Displays the names of files used by the AIS Agent.
`HELP`	Display a list of console commands.
`LEVEL`	Display the AIS Agent level in the format: `major_version.minor_version.interim_correction`
`STATUS`	Displays information about the state of the AIS Agent.
`WORKER COUNTS`	Displays the following counts: • Free agent worker activities (workers not currently processing an AIS client task) • Assigned agent worker activities (workers currently processing a AIS client task) • Maximum number of agent worker activities (obtained from the max_activities configuration parameter; see 3.2.13 max_activities)
`WORKER {ID} STATUS`	Displays information for the specified agent worker. If you do not specify *id*, information is displayed for all active workers. The *id* is a worker's client session ID, in decimal. Use the *worker status* command to get a list of the worker ids.

3.4.2. Console Commands for Controlling the AIS Agent

The following table describes the commands that let you dynamically tailor the AIS Agent and set new values for certain agent operational values, some of which are configuration parameters. The new setting takes effect immediately. If a specified value is not valid, it returns an error and has no effect.

Command Syntax	Description
`CLEAR AGENT COUNTS`	Clears the following fields: • Total requests processed: total number of requests submitted to the agent for all clients • Last request timestamp: timestamp for the last request
`SET CLIENT RECEIVE TIMEOUT=n`	*n* is the time in seconds for the client session receive timeout value.

Command Syntax	Description
SET LOG CONSOLE OUTPUT=ON|OFF	(Default) The ON option causes the output from the console commands to be generated in the agent log file (as well as returned to the sender of the console command.) The OFF option turns off the mode that causes the output from the console commands to be generated in the agent log file.
SET MAX ACTIVITIES=n	*n* is the maximum number of agent workers, free or assigned.

3.4.3. Console Commands for Stopping the AIS Agent

The following table describes the commands that allow you to stop the AIS Agent. The Console Handler TERM command should be sufficient in most cases to terminate the agent. However, if the TERM command is unable to complete the termination gracefully, the ABORT command might be required.

Command Syntax	Description
ABORT	Normally used only when a previous TERM command fails. This command forces the AIS Agent to immediately: • Stop all activities. • Close all AIS client connections with Connectivity Services. • Shut down the console.
TERM {GRACEFULLY|IMMEDIATELY} or TERM {GR|IM}	The normal method of shutting down the AIS Agent is to do a TERM GRACEFULLY. The results are: • No new AIS clients are accepted. • Existing AIS clients are allowed to complete their processing for their connections. • The agent workers shut down when the connection is closed. • The client session with Connectivity Services is closed normally. The results of doing a TERM IMMEDIATELY are: • No new AIS clients are accepted. • For existing AIS clients, no new client request operations are accepted. • All client connections with Connectivity Services are closed. • An immediate shutdown might result in the AIS client program receiving a session termination error when the connection is dropped.

3.4.4. Console Commands for Stopping the Agent Workers

The following table describes the commands you can use to stop an agent worker activity. Each worker handles an AIS client.

Command Syntax	Description
`ABORT WORKER id`	Immediately stops a worker activity identified by *id*. Use the worker status command to get a list of the worker ids. The AIS client Connectivity Services session is closed immediately.

`TERM WORKER id {GRACEFULLY|IMMEDIATELY}`
or
`TERM WORKER id {GR|IM}`
	Terminate the worker activity identified by *id*. Use the worker status command to get a list of the worker ids.
	The TERM WORKER GRACEFULLY command shuts down the activity when an AIS client operation is completed.
	The AIS client Connectivity Services session is closed immediately.
	The TERM WORKER IMMEDIATELY command results in:
	• No new AIS client task requests are accepted.
	• The AIS client Connectivity Services session is closed immediately.
	• An immediate shutdown might result in the AIS client program receiving a session termination error when the connection is dropped.

3.4.5. Console Commands for Controlling the AIS Agent Log and Trace Files

While the AIS Agent is operational, it has an open log file. The agent records information in the file from time to time to keep a log of events as they occur. The AIS Agent also has a trace file available for obtaining diagnostic information. The agent records information in the file only while tracing is turned on. You can turn tracing on and off dynamically.

Each message sent to the AIS Agent log file and the trace file is prefixed with a date and timestamp value.

The runstream that starts the AIS Agent cycles up the default log file (AIS*x**AIS$LOG) and the default trace file (AIS*x**AIS$TRACE), where *x* is the installation mode. Cycling these files ensures that unique files are used whenever the agent is started.

The following table describes the commands you can use to control the log and trace files.

Command Syntax	Description
`CYCLE LOG|TRACE FILE`	Closes the existing F-cycle of the log or trace file and creates a new F-cycle of the current agent log or trace file. This file is where future log or trace file output is placed.
`SET LOG|TRACE FILE=filename`	Sets the agent log or trace file name. Closes the previous log or trace file and opens the new one with the specified name. If the file does not exist, it is cataloged. There is always an agent log file open.
`SET TRACE MASK=n`	Sets the amount of tracing done for all AIS Agent workers. Set *n* to the value provided by a Unisys support engineer.

Command Syntax	Description
`SET WORKER id TRACE MASK=n`	Sets the amount of tracing done for a particular *id*. Set *n* to the value provided by a Unisys support engineer. Default: uses the mask from the set agent trace mask command.
`TURN ON|OFF TRACE`	Turns on or off tracing in the agent trace file for the previously specified trace file. When you turn tracing on, the trace file is opened in append mode.

Section 4
AIS Client Administration

The Application Integration Services (AIS) Configuration Utility (called "Configuration Utility" in this document) is a graphical user interface that provides an easy way to create and modify the configuration files that Application Integration Services uses. This topic describes how to configure Application Integration Services using the Configuration Utility.

4.1. Understanding Host Connections

A host connection is required before communicating with an OS 2200 system. To establish this connection, you need a user-id, password, IP address or host name, and port. Your system administrator can provide these items, but you need to understand how to use them.

The OS2200Connection constructor establishes a connection and accepts the connection name as a parameter. The connection name is the name of an entry in the host configuration file that provides the IP address or host name and port for a system.

Note: *Do not confuse the connection name with the host name or computer name.*

The constructor also accepts an OS 2200 user-id and password as parameters.

A connection is established if the OS2200Connection constructor does not throw an exception.

Figure 4–1 shows the relation among the OS2200Connection constructor, configuration file, and the OS 2200 system. In this figure

- An application, running in the Windows environment, uses a connection name of "mySystem", a user-id of "user" and a password of "pass".
- The application creates a connection using the OS2200Connection constructor and passes the "mySystem" connection name, the user-id, and the password.
- The system resolves the "mySystem" connection name by referencing the "mySystem" entry in the host configuration (ClearPathHosts.config) file. The "mySystem" entry specifies the IP address of the ClearPath OS 2200 system and port that the AIS Agent is listening on.

Note: *In this example, the host configuration file specifies an IP address. Instead of the IP address, it could specify a hostname.*

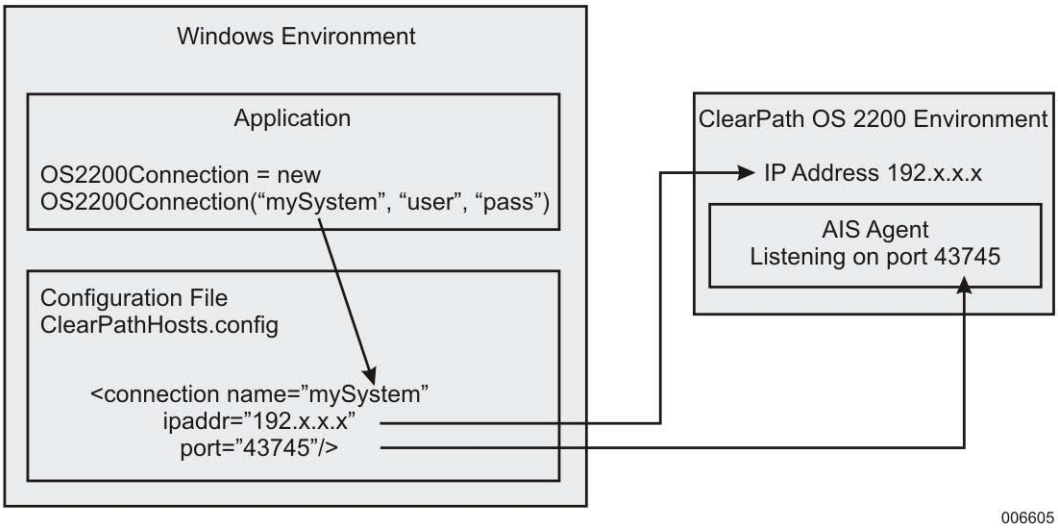


Figure 4–1. Establishing a Connection

To specify a host connection, you must add the connection name to your configuration file (see 4.2 Specifying Host Connection Information).

4.2. Specifying Host Connection Information

Before your application can use Application Integration Services, you must create a host configuration file. The host configuration file contains the information that Application Integration Services needs to establish a connection to a ClearPath OS 2200 system.

Each application requires access to a host configuration file.

Default Connections

In a host configuration file, the configuration utility supplies a default connection for each host type (*DEFAULT_MCP* or *DEFAULT_OS2200*). The default connection specifies a cleartext port number of 43745 and the appropriate system type. You still must provide either a hostname or an IP address for either of the default connections before they can be used.

In a configuration file, you can update either of the default connections by entering values for the remaining fields and the Port Number field or click **Add** to create your own connections.

Procedure

Using the following procedure to create or modify a host configuration file:

1. Start the Configuration Utility. (The path to the Configuration Utility is **Start**, **All Programs**, **Unisys**, **ClearPath Application Integration Services**, and then **AIS Configuration Utility**.)
2. Use the Configuration Utility to
   - Create a new host configuration file.
   - Add connection information to the host configuration file using either a default connection (*DEFAULT_MCP* or *DEFAULT_OS2200*) or the **Add** button.
   - Modify an existing connection.
   - Test the connection to the specified host.

   See the *Configuration Utility Help* for details.
3. Create a host configuration file. A ClearPathHosts.config file is automatically created in the default folder if the file does not already exist:

   ```
   %allusersprofile%\Unisys\ClearPathInterface\Configuration
   ```

Notes:

- The Port Number value for a secure connection must agree with the port number that the AIS Agent is using for an SSL connection. If a secure client attempts to connect to the cleartext port number, or a cleartext client attempts to connect to the SSL port number, the connection attempt either times out and the client receives an OS 2200 exception after 30 seconds or it receives an immediate OS 2200 exception.
- A secure connection can only be established using a fully qualified hostname (FQHN).
- You cannot delete or rename the default connections. The configuration utility adds the default connections to any host configuration file that does not already have them. However, it is not necessary for you to define the default connections if you want your users to explicitly define the connection name.
- The Configuration Utility allows you to specify both a hostname and an IP address. If you provide a hostname, the OS2200Connection constructor attempts to resolve the connection using that name. If you do not provide a hostname, it uses the specified IP address. Neither the Configuration Utility nor the OS2200Connection constructor attempt to confirm the hostname and IP address at runtime. The precedence is hostname first, then IP address.

Managing Host Configuration file through API

You can manage the host configuration file containing information required to establish a connection with a Clearpath system through APIs. You can add, update, and remove connections to one or more ClearPath systems through APIs, without using AIS Configuration Utility.

When you add a new connection through API, it checks if the configuration file exists and adds the connection to the existing configuration file. If the configuration file does not exist, a default configuration file with two default connections, MCP and OS2200, is created and then a new connection is added to this file.

In the host configuration file, you can modify or remove connections to one or more ClearPath systems through APIs. However, you cannot remove/rename the default connection in the configuration file.

4.3. Enabling and Disabling ClearPathInterface Tracing

Application Integration Services is released with a ClearPathParentSource.config file. When needed, this file specifies the logging and tracing options for an application.

Normal operation of Application Integration Services does not require you to specify any tracing options. However, when debugging an application, specifying one or more tracing options may provide useful information. Use the Configuration Utility to turn the tracing options on or off:

1. If the ClearPathParentSource.config file does not exist in the default folder, the Configuration Utility creates one by copying the ClearPathParentSource.config template. If the template does not exist in the default folder, a ClearPathParentSource.config file is not opened.

   The default folder is:

   ```
   %allusersprofile%\Unisys\ClearPathInterface\Configuration
   ```

2. Start the Configuration Utility. (The path to the Configuration Utility is **Start**, **All Programs**, **Unisys**, **ClearPath Application Integration Services**, and then **AIS Configuration Utility**.)
3. Using the Configuration Utility, specify the desired tracing options. See the *Configuration Utility Help* for details.

4.4. Specifying a Communication Trace

The options on the **Communication Tracing** tab enable you to start and stop tracing of the Connectivity Service or the Windows networking (Winsock) software. Normal operation of Application Integration Services does not require you to use these traces. You only need to turn on one of these traces if directed by a Unisys service representative.

To specify a communication trace

Note: *Unisys recommends that you run the Configuration Utility as administrator.*

1. In the Configuration Utility, select the **Communication Tracing** tab.
2. Start either a new or existing Connectivity Service or Winsock trace. See the *Configuration Utility Help* for details.

3. To stop the trace, select the trace name (file name) from the **List of Trace Names** box and click **Stop**.
4. Send the trace file to the Unisys service representative for analysis.

Notes

- The Configuration Utility saves the trace file in one of the following folders:

  Connectivity Service:

  ```
  %allusersprofile%\Unisys\ClearPathInterface\logs\UConnect
  ```

  WinSock:

  ```
  %allusersprofile%\Unisys\ClearPathInterface\logs\Winsock
  ```

- Trace files can become large very quickly. Make sure you monitor the trace while it's running.
- When you no longer need the trace file, use the **Delete** command to delete it from the folder.

Section 5
Getting Started Using Application Integration Services

This section describes the application samples that you can use to get started.

5.1. Using the Application Samples

Once Application Integration Services is installed and configured, you can start using it. To help you get started, the installation process installed sample applications that demonstrate many of the classes included in the ClearPathInterface.

When Application Integration Services is installed on your Windows system, the default installation directory includes two zip files containing samples for ClearPath MCP and ClearPath OS 2200.

To use the OS 2200 samples, unzip the OS 2200 version into a different location of your choice. This creates a folder labeled "Samples_OS2200/OS 2200 File Access" with subfolders for different .NET programming languages, such as C#, Visual Basic, and PowerShell. Each sample is in its own folder and contains a Visual Studio solution that demonstrates the technique for that sample.

Typically you open a project or program and change various variable definitions in the program, such as user-id, password, connection name, and file details. Then you build and run the sample program. Alternatively, you can copy a snippet of code from the sample program into your own program and make appropriate changes there.

The OS 2200 File Access folder contains a text file named Readme.txt, which provides a brief description of each of the sample programs.

In the C# folder, one of the samples is named "CsharpSamples". This sample is a single Visual Studio solution that contains a setup program and project containers for all of the other C# sample programs. You can optionally use this program to define the folders and sample files that all of the samples in this solution use.

5.2. Windows PowerShell Considerations

Note: *The minimum level required is PowerShell 2.0.*

If you use PowerShell 2.0, you need two configuration files to successfully run PowerShell scripts in the Application Integration Services environment. These configuration files enable PowerShell to load a newer version of the .NET Framework Common Language Runtime (CLR) that the ClearPathInterface.dll and its dependencies are built with. Without the configuration files, you receive a message similar to the following message:

```
Import-Module : Could not load file or assembly
'file:///C:\Program Files\Unisys\ClearPath\App Integration
Services\ClearPathInterface.dll' or one of its dependencies.
This assembly is built by a runtime newer than the
currently loaded runtime and cannot be loaded.
```

The two configuration files (one for powershell.exe and the other for powershell_ise.exe) are in the Samples_OS2200.zip file:

```
Samples_OS2200\OS 2200 File Access\PowerShell\Configuration Files
```

Copy the configuration files and place them in the following folder:

```
%Windows%\System32\WindowsPowerShell\v1.0
```

If you are using the 32-bit version of PowerShell with a 64-bit operating system, also place the files in the following folder:

```
%Windows%\SysWOW64\WindowsPowerShell\v1.0
```

Section 6
Using the Unisys.OS2200.Interface Namespace

6.1. Namespace Overview

The Unisys.OS2200.Interface namespace contains classes used to:

- Open a connection to a ClearPath system
- Create and delete directories
- Create, open, and delete files
- Read and write files
- Check for the existence of directories and files
- Control access to directories and files
- Call TIP/HVTIP transactions
- Start a batch run

Many of the classes found in the Unisys.OS2200.Interface namespace are similar to classes found in the .NET System.IO namespace. For example, the OS2200File, OS2200Directory, and OS2200FileStream classes found in the Unisys.OS2200.Interface namespace operate very much like the File, Directory, and FileStream classes found in the .NET System.IO namespace.

When possible, classes in the Unisys.OS2200.Interface namespace extend .NET Framework classes. For example, the OS2200FileStream class extends the System.IO.Stream class. This allows an OS2200FileStream class instance to be used with other .NET Framework classes that operate on a System.IO.Stream class, such as the StreamReader and StreamWriter classes found in the System.IO namespace.

6.2. Connection API

The OS2200Connection class is used to open a connection to a ClearPath OS 2200 system. An open connection object is used by other classes to communicate with the ClearPath OS 2200 system. See 4.1 Understanding Host Connections for details on configuring a connection.

The OS2200LogonProperty enumeration defines extended properties used when opening a connection. For example, the OS2200LogonProperty.AuthenticationScheme property is specified to indicate if authentication is performed using OS 2200 credentials or Windows credentials.

Table 6-1 provides a description for the OS2200LogonProperty enumeration values.

Table 6–1. Description of the OS2200LogonProperty Enumeration

Property	Description
Account	Indicates the account used to control access to files on a Fundamental Security system where file ownership is based on an account. Default: First account defined in a user-id's security record.
AuthenticationScheme	Indicates the authentication scheme used for connection authentication. Values: "OS2200USER" or "NTLM" Default: "OS2200USER"
DomainName	Indicates the Windows domain name used for connection authentication. This property only applies when the authentication scheme is NTLM.
Password	Indicates the password used for connection authentication.
ProjectId	Indicates the project-id used to control access to files on a Fundamental Security system where file ownership is based on a project-id. Default: Either QQQ$ or the first project-id defined in a user-id's security record.
UserId	Indicates the user-id used for connection authentication.

Note: *If the UserId and Password properties are not specified or are NULL and the AuthenticationScheme property is "NTLM", then connection authentication is done using the identity of the signed-on user.*

6.2.1. Opening a Connection

To open a connection to a ClearPath OS 2200 system, an application must construct an OS2200Connection object. An OS2200Connection object is constructed using one of the following class constructors:

```
OS2200Connection(
    Dictionary<OS2200LogonProperty, string> connProps)

OS2200Connection(string connectionName,
    Dictionary<OS2200LogonProperty, string> connProps)

OS2200Connection(string userId, string password)

OS2200Connection(string connectionName,
    string userId, string password)
```

Example: Opening a Connection

This example demonstrates how an OS2200Connection object is typically constructed.

```
OS2200Connection conn =
    new OS2200Connection("Sys1", "user43", "aaaaa");
```

Example: Opening a Connection Using Properties

This example demonstrates an alternative way to construct an OS2200Connection object using properties.

```
Dictionary<OS2200LogonProperty, string> props =
    new Dictionary<OS2200LogonProperty, string>()
{
    { OS2200LogonProperty.UserId, "winuser" },
    { OS2200LogonProperty.Password, "wwwwww" },
    { OS2200LogonProperty.AuthenticationScheme, "NTLM" }
};
OS2200Connection conn = new OS2200Connection("Sys1", props);
```

6.2.2. Closing a Connection

A connection represents a shared system resource so it is important for an application to close a connection when it is no longer needed. There are two methods that can be used to close a connection:

```
Close()
Dispose()
```

Example: Closing a Connection

```
OS2200Connection conn =
    new OS2200Connection("Sys1", "user43", "aaaaa");

conn.Close();
```

or

```
conn.Dispose()
```

6.2.3. Connection Properties

The OS2200Connection class define two properties:

- CharacterEncoding

  Defines the character encoding used for parameters passed to and received from the AIS Agent. The default character encoding is ASCII.

  Note: *An application is responsible for encoding file data.*

- MaximumMessageSize

  Defines the maximum size of a message passed to or received from the AIS Agent. An application can use this property to ensure that I/O operations do not exceed the maximum size limit.

6.3. Directory API

The OS2200Directory class exposes static methods used to

- Create a directory.
- Delete a directory.
- Control access to a directory.
- Determine the existence of a directory.
- Get the names of subdirectories and files in a directory.

The OS2200DirectoryInfo class is used to obtain information about a directory and file system entries contained in a directory.

6.3.1. Creating a Directory

The OS2200Directory class exposes two static methods used to create a directory. The caller of these methods becomes the owner of the directories created. Any directories in a path that do not already exist are created.

```
CreateDirectory(OS2200Connection connection, string path)
```

```
CreateDirectory(OS2200Connection connection, string path,
    OS2200DirectoryAccessControl accessControl);
```

Example: Creating a Directory Using Access Control

This example demonstrates how to create a directory and define the access control properties that determine who is allowed to access the directory.

```
OS2200DirectoryAccessControl accessControl=
    new OS2200DirectoryAccessControl();
accessControl.Owner = OS2200AccessMode.ReadWriteExecute;
accessControl.Other = OS2200AccessMode.ReadExecute;
accessControl.AccessControlRecordName =
    OS2200DirectoryAccessControl.ACR_PUBLIC;

OS2200Directory.CreateDirectory(conn, "/os2200/example/data",
    accessControl);
```

Note: *In the preceding example, the directories named "example" and "data" are created. All users have read access to the new directories but only the owner can update or delete the contents of the directories.*

Example: Creating a Directory Using Options

This example demonstrates how to create a directory with options that control the size and placement of the directory.

The OS2200DirectoryOptions class specifies options associated with the underlying OS 2200 program file created when a CIFS directory is created. Options such as device-type, maximum size, and removable pack IDs can be specified.

An OS2200DirectoryOptions class instance is a thread-local singleton that is created prior to creating a directory. The options specified remain in effect until the class instance is cleared.

```
OS2200DirectoryOptions options =
    OS2200DirectoryOptions.Create();

string[] packIDs = { "3XG70", "3XG71" };
options.DeviceType = "F";
options.GranuleSize = OS2200StorageGranuleSizeType.Track;
options.Reserve = 1000;
options.MaximumLength = 9999;
options.PackId = packIDs;

// Note:  It is not necessary to pass the OS2200DirectoryOptions
//        class instance as input to the CreateDirectory method.
OS2200Directory.CreateDirectory(conn, "/os2200/example/data");

// Clear the directory options unless they should be used
// when creating subsequent directories.
OS2200DirectoryOptions.Clear();
```

6.3.2. Deleting a Directory

The OS2200Directory class exposes two static methods used to delete a directory.

```
Delete(OS2200Connection connection, string path)

Delete(OS2200Connection connection, string path,
     bool recursive)
```

Example: Deleting a Directory Recursively

This example demonstrates how to recursively delete a directory and its contents.

```
OS2200Directory.Delete(conn, "/os2200/example/data", true);
```

Note: *All of the subdirectories and files in the "data" directory are deleted if the caller has the required permission.*

6.3.3. Checking the Existence of a Directory

The OS2200Directory class exposes two static methods used to determine if a directory exists. These methods are used in an application to determine if a directory exists and what access type a caller has in a directory.

```
Exists(OS2200Connection connection, string path)

Exists(OS2200Connection connection, string path,
    FileAccess access)
```

Example: Checking if a Directory Exists

This example demonstrates how to determine if a directory exists with a specific access type.

```
if(OS2200Directory.Exists(conn,"/os2200/example/data",
                          FileAccess.Read))
{
    // Do some work if the directory exists
}
```

Note: *The Exists method can be used to determine if a directory exists and what permissions the caller has for a directory. In the preceding example, the code determines if the caller has read access in the "data" directory.*

6.3.4. Getting Names of Directory Entries

The OS2200Directory class exposes six static methods used to get the full path names of entries in a directory and optionally in all subdirectories.

```
GetDirectories(OS2200Connection connection, string path)

GetDirectories(OS2200Connection connection, string path,
    SearchOption searchOption)

GetFiles(OS2200Connection connection, string path);

GetFiles(OS2200Connection connection, string path,
    SearchOption searchOption)

GetFileSystemEntries(OS2200Connection connection, string path)

GetFileSystemEntries(OS2200Connection connection, string path,
    SearchOption searchOptions)
```

Example: Getting the Names of Files in a Directory

This example demonstrates how to get the full path name of each file in a directory. The string array named "fileNames" is populated with the full path name of files in the "data" directory.

```
string[] fileNames =
    OS2200Directory.GetFiles(conn, "/os2200/example/data");
```

Example: Recursively Getting the Names of Subdirectories

This example demonstrates how to recursively get the full path name of all subdirectories under a directory. The string array named "dirNames" is populated with the full path name of subdirectories in the "data" directory.

```
string[] dirNames = OS2200Directory.GetDirectories(conn,
    "/os2200/example/data", SerachOption.AllDirectories);
```

6.3.5. Getting or Setting the Current Directory

The OS2200Directory class has a static method used to get the current directory and a static method used to set the current directory. Being able to control the current directory is important in applications that use relative path names.

```
GetCurrentDirectory(OS2200Connection connection)

SetCurrentDirectory(OS2200Connection connection,string path)
```

Example: Setting the Current Directory

This example demonstrates how to establish the current directory.

```
OS2200Directory.SetCurrentDirectory(conn,
    "/os2200/example/data");
```

6.3.6. Getting Information about a Directory

The OS2200DirectoryInfo class is used to obtain the information about a directory.

```
OS2200DirectoryInfo(OS2200Connection connection, string path)
```

After constructing an OS2200DirectoryInfo class instance, the Exists property is used to determine if a directory exists. The following information is available for an existing directory:

- Create time
- Access time
- Write time
- Fully-qualified path name
- Owner
- Native OS 2200 name

Example: Getting Information about a Directory

This example demonstrates how to obtain information about a directory. The Exists property is checked to verify that the directory exists before accessing other properties.

```
OS2200DirectoryInfo dirInfo =
    new OS2200DirectoryInfo(conn, "/os2200/example/data");
if (dirInfo.Exists)
{
    // Display the directory owner
    Console.WriteLine(dirInfo.Owner);
}
```

6.3.7. Getting Information for Directory Entries

The OS2200DirectoryInfo class defines methods for obtaining information about subdirectories and files in a directory.

```
OS2200DirectoryInfo[] GetDirectories()

OS2200DirectoryInfo[] GetDirectories(string searchPattern)

OS2200DirectoryInfo[] GetDirectories(string searchPattern,
    SearchOption searchOption)

OS2200FileInfo[] GetFiles()

OS2200FileInfo[] GetFiles(string searchPattern)

OS2200FileInfo[] GetFiles(string searchPattern,
    SearchOption searchOption)

OS2200FilesystemInfo[] GetFilesystemEntries()

OS2200FilesystemInfo[] GetFilesystemEntries(
    string searchPattern)

OS2200FilesystemInfo[] GetFilesystemEntries(
    string searchPattern, SearchOption searchOption)
```

The searchPattern parameter indicates a pattern used to match the names of subdirectories or files included in the resulting array. The parameter value can contain a combination of literal characters and wild-card characters (* and ?). The default pattern is "*".

The searchOption parameter indicates if subdirectories should be recursively searched for names that match the searchPattern parameter value. The default is to search only the top-level directory.

Example: Get Information about Files in a Directory

This example demonstrates how to obtain information about files with names that match the specified pattern.

```
OS2200DirectoryInfo dirInfo =
    new OS2200DirectoryInfo(conn, "/os2200/example/data");
OS2200FileInfo[] files = dirInfo.GetFiles("ca*.c");
```

6.3.8. Directory API Samples

For sample applications that demonstrate usage of the OS2200Directory and OS2200DirectoryInfo classes, see the Samples folder in your installation directory

6.4. File API

The OS2200File class exposes static methods used to

- Create a file.
- Delete a file.
- Open a file for reading or writing.
- Control access to a file.
- Determine the existence of a file.
- Copy a file.
- Move a file.

The OS2200FileInfo class is used to obtain information about a file.

6.4.1. Creating a File

The OS2200File class exposes two static methods used to create a text file and two static methods used to create a binary file. The caller of these methods becomes the owner of the file. If the specified file already exists, it is truncated; otherwise, a new file is created. These methods all return an OS2200FileStream object that can be used to write data to the file.

```
Create(OS2200Connection connection, string path)

Create(OS2200Connection connection, string path,
    OS2200AccessControl accessControl)

CreateBinary(OS2200Connection connection, string path)

CreateBinary(OS2200Connection connection, string path,
    OS2200AccessControl accessControl)
```

Example: Creating a File Using Access Control

This example demonstrates how to create a file and define the access control properties that determine who is allowed to access the file.

```
OS2200AccessControl accessControl= new OS2200AccessControl();
accessControl.Owner = OS2200AccessMode.ReadWriteExecute;
accessControl.Other = OS2200AccessMode.ReadExecute;
accessControl.AccessControlRecordName =
    OS2200AccessControl.ACR_PUBLIC;

OS2200FileStream strm = OS2200File.CreateFile(conn,
```

```
    "/os2200/example/data/Info.txt", accessControl);

strm.Close();  // Always close a file when no longer needed
```

Example: Creating a File Using Options

This example demonstrates how to create a file with options that control the size and placement of the file.

The OS2200FileOptions class specifies options associated with the underlying OS 2200 data file created when a CIFS file is created. Options such as device-type, maximum size, and removable pack IDs can be specified.

An OS2200FileOptions class instance is a thread-local singleton that is created prior to creating a file. The options specified remain in effect until the class instance is cleared.

```
OS2200FileOptions options = OS2200FileOptions.Create();

string[] packIDs = { "3XG70", "3XG71" };
options.DeviceType = "F";
options.GranuleSize = OS2200StorageGranuleSizeType.Track;
options.Reserve = 1000;
options.MaximumLength = 9999;
options.PackId = packIDs;

// Note:  It is not necessary to pass the OS2200FileOptions
//        class instance as input to the CreateFile method.
OS2200FileStream strm = OS2200File.CreateFile (conn,
    "/os2200/example/datafile");

strm.Close(); // Always close a file when no longer needed

// Clear the file options unless they should be used
// when creating subsequent files.
OS2200FileOptions.Clear();
```

Example: Creating a File with Read/Write Keys

This example demonstrates how to create a file with read/write keys.

The OS2200ReadWriteKeys class specifies the read/write associated with the underlying OS 2200 data file created when a CIFS file is created.

An OS2200ReadWriteKeys class instance is a thread-local singleton that is created prior to creating a file. The read/write keys are cleared after each file operation.

```
string readKey = "R_KEY";
string writeKey = "W_KEY";

OS2200ReadWriteKeys keys = OS2200ReadWriteKeys.Create();

// The format for a key is:
//
// "readKey/writeKey" when both a read/write key is required,
```

```
// "readKey" when only a read key is required, and
// "/writeKey" when only a write key is required.
//
// Note: The read/write key is cleared after file operation.
keys.SourceKey = readKey + "/" + writeKey;

// Note: It is not necessary to pass the OS2200ReadWriteKeys
//       class instance as input to the CreateFile method.
OS2200FileStream strm = OS2200File.CreateFile (conn,
    "/os2200/example/datafile");

strm.Close(); // Always close a file when no longer needed
```

6.4.2. Deleting a File

The OS2200File class exposes one static method used to delete a file. An exception will not be thrown if the specified file does not exist.

```
Delete(OS2200Connection connection, string path)
```

Example: Deleting a File

This example demonstrates how to delete a file.

```
OS2200File.Delete(conn, "/os2200/example/data/Info.txt");
```

6.4.3. Opening a File

The OS2200File class exposes four static methods used to open a text file and two static methods used to open a binary file. These methods all return an OS2200FileStream object that can be used to read data, write data, or both.

```
Open(OS2200Connection connection, string path,
    FileMode fileMode)

Open(Os2200Connection connection, string path,
    FileMode fileMode, FileAccess fileAccess)

OpenRead(OS2200Connection connection, string path)

OpenText(OS2200Connection connection, string path)

OpenBinary(OS2200Connection connection, string path,
    FileMode fileMode)

OpenBinary(OS2200Connection connection, string path
    FileMode fileMode, FileAccess fileAcces)
```

The fileMode parameter indicates how the file is opened. Table 6–2 describes each FileMode and provides the requirements FileMode imposes.

Table 6–2. FileMode Requirements

FileMode	Requires Existing File?	Can Use Existing File?	Comments
CreateNew	No	No	A file is created.
Open	Yes	Yes	Data in the existing file is retained.
Truncate	Yes	Yes	All data in the existing file is eliminated.
Append	No	Yes	New data can be added to the end of the file. If the file does not exist, it is created.
Create	No	Yes	If the file exists, the file is processed as FileMode Truncate. If the file does not exist, the file is processed as FileMode CreateNew.
OpenOrCreate	No	Yes	If the file exists, the file is processed as FileMode Open. If the file does not exist, the file is processed as FileMode CreateNew.

The fileAccess parameter indicates if a file is opened for reading, writing, or both.

Example: Opening a Text File for Reading and Writing

This example demonstrates how to open a file and return an OS2200FileStream that is capable of reading and writing data.

```
OS2200FileStream strm =  OS2200File.Open(
    conn, "/os2200/example/data/Info.txt", FileMode.Open,
    FileAccess.ReadWrite);

strm.Close();  // Always close a file when no longer needed
```

Note: *In the preceding example, an exception is throw if the file does not exist.*

Example: Opening a Text File for Reading

This example demonstrates how to open a file and return an OS2200FileStream that is only capable of reading data.

```
OS2200FileStream strm =  OS2200File.OpenText(
    conn, "/os2200/example/data/Info.txt");

strm.Close();  // Always close a file when no longer needed
```

6.4.4. Checking the Existence of a File

The OS2200File class exposes two static methods used to determine if a file exists. These method are used in an application to determine if a directory exists and what access type a caller has for a file.

```
Exists(OS2200Connection connection, string path)

Exists(OS2200Connection connection, string path,
    FileAccess access)
```

Example: Checking if a File Exists With Write Access

This example demonstrates how to determine if a file exists and if the caller has write access.

```
if(OS2200File.Exists(conn, "/os2200/example/data/Info.txt",
                     FileAccess.Write))
{
    // Do some work if the caller has write access.
}
```

6.4.5. Copying a File

The OS2200File class exposes two static methods used to copy a file.

```
Copy(OS2200Connection connection, string sourceFileName,
    string destFileName)

Copy(OS2200Connection connection, string sourceFileName,
    string destFileName, bool overwrite)
```

Example: Copying a File With Overwrite

This example demonstrates how to copy a file to a new location and overwrite the destination file if it already exists.

```
OS2200File.Copy(conn,
    "/os2200/example/data/Info.txt",
    "/os2200/example/NewInfo", true);
```

6.4.6. Moving a File

The OS2200File class exposes a static method used to move a file.

```
Move(OS2200Connection connection, string sourceFileName,
    string destFileName)
```

Example: Moving a File

This example demonstrates how to move a file to a new location.

```
OS2200File.Move(conn,
    "/os2200/example/data/Info.txt",
    "/os2200/example/NewInfo");
```

Note: *An exception is thrown if the destination file already exists.*

6.4.7. Getting Information about a File

The OS2200FileInfo class is used to obtain the information about a file.

```
OS2200FileInfo(OS2200Connection connection, string path)
```

After constructing an OS2200FileInfo class instance, the Exists property is used to determine if a file exists. The following information is available for an existing file:

- Create time
- Access time
- Write time
- Fully-qualified path name
- Owner
- Native OS 2200 name
- Element type
- Element sub-type
- File length (bytes)

Example: Getting Information about a File

This example demonstrates how to obtain information about a file. The Exists property is checked to verify the file exists before accessing other properties.

```
OS2200FileInfo fileInfo =
    new OS2200FileInfo(conn, "/os2200/test/data/results.txt");
if (fileInfo.Exists)
{
    // Display the file owner
    Console.WriteLine(fileInfo.Owner);
}
```

6.4.8. File API Samples

For sample applications that demonstrate usage of the OS2200File and OS2200FileInfo classes, see the Samples folder in your installation directory.

6.5. File Stream API

The OS2200FileStream class represents a stream of bytes in a text file or binary file. This class is used to read or write a specified number of bytes at the current position within the file stream. An application is responsible for handling data encoding.

6.5.1. Creating a File Stream

The OS2200FileStream class defines two constructors used to create a filestream.

Typically an application uses one of the OS2200File Create or Open methods to create an OS2200FileStream. See 6.4.1 Creating a File and 6.4.3 Opening a File for more information.

```
OS2200FileStream(OS2200Connection connection, string path,
    FileMode fileMode, bool binary)

OS2200FileStream(OS2200Connection connection, string path,
    FileMode fileMode, FileAccess fileAccess, bool binary)
```

Example: Creating a File Stream

This example demonstrates how to create an OS2200FileStream using a class constructor.

```
OS2200FileStream strm = new OS2200FileStream(
   conn, "/os2200/example/data/Info.txt",
   FileMode.OpenOrCreate, false);

strm.Close();  // Always close a file when no longer needed
```

6.5.2. Closing a File Stream

A file is a shared resource that might be used by multiple programs so it is important to close a file when it is no longer being used. Also, closing a file guarantees that all data in the filestream has been written to the file.

```
Close()

Dispose()
```

Example: Closing a File Stream

This example demonstrates multiple ways to close an OS2200FileStream object.

```
OS2200FileStream strm = new OS2200FileStream(
   conn, "/os2200/example/data/Info.txt",
   FileMode.OpenOrCreate, false);

strm.Close();
```

or

```
strm.Dispose()
```

6.5.3. Reading Data from a File

If an OS2200FileStream was opened for reading, it can then be used to read bytes of data from a file. A program is responsible for defining a byte array to contain the bytes of data read from a file and to specify the number of bytes to read. File data is read from the current position within the stream. After calling the Read method, the current position in the filestream is the next available byte of data.

```
Read(byte[] buffer, int offset, int count)
```

Example: Reading Data from a File

This example demonstrates using an OS2200FileStream object to read bytes of data from a file.

```
OS2200FileStream strm =  OS2200File.OpenText(
    conn, "/os2200/example/data/Info.txt");

if (strm.CanRead)  // Make sure stream open for reading
{
    byte[] data = new byte[8192];  // Declare an 8K byte array
    int bytesRead = strm.Read(data, 0, 8192);
}

strm.Close();  // Always close a file when no longer needed
```

Note: *The number of bytes requested should never exceed MaximumMessageSize (see 6.2.3 Connection Properties). The number of bytes actually returned by the Read method might be less than the requested number of bytes.*

6.5.4. Writing Data to a File

If an OS2200FileStream was opened for writing then it can be used to write bytes of data to a file. File data is written to the current position within the stream. After calling the Write method, the current position in the filestream is the next byte after the written data.

```
Write(byte[] buffer, int offset, int count)
```

Example: Writing Data to a File

This example demonstrates using an OS2200FileStream object to write bytes of data to a file.

```
byte[] data = new byte[8192]; // Populate with data to write

OS2200FileStream strm =  OS2200File.Open(
    conn, "/os2200/example/data/Info.txt"
    FileMode.Create, FileAccess.Write);

if (strm.CanWrite)  // Make sure stream open for writing
{
    int bytesWritten = strm.Write(data, 0, 8192);
}
```

```
strm.Close();  // Always close a file when no longer needed
```

Note: *The number of bytes written should never exceed MaximumMessageSize (see 6.2.3 Connection Properties).*

6.5.5. Changing the File Stream Position Property

The OS2200FileStream class defines a Position property that indicates a byte offset from the beginning of a filestream. A program might alter the Position property to indicate where the Read method starts reading data or where the Write method starts writing data.

Another way to alter the Position property is to call the Seek method. The Seek method alters the Position property relative to the beginning, end, or current position in a file.

```
Seek(long offset, SeekOrigin origin)
```

Example: Changing the File Stream Position Property

This example demonstrates using the Seek method to alter the value of the Position property so that data can be written at the end of a file.

```
byte[] data = new byte[8192]; // Populate with data to write

OS2200FileStream strm =  OS2200File.Open(
    conn, "/os2200/example/data/Info.txt"
    FileMode.Open, FileAccess.ReadWrite);
strm.Seek(0, SeekOrigin.End); // Set Position to end-of-file

if (strm.CanWrite)  // Make sure stream open for writing
{
    int bytesWritten = strm.Write(data, 0, 8192);
}

strm.Close();  // Always close a file when no longer needed
```

6.5.6. Altering the File Stream Length

The OS2200FileStream class defines a SetLength method that can be used to alter the length of a file. If the new length is less than the current file length, the excess bytes are truncated. If the new length is greater than the current file length, extra null bytes are added at the end of the file.

6.5.7. File Stream API Samples

For sample applications that demonstrate usage of the OS2200FileStream class, see the Samples folder in your installation directory.

6.6. TIP Transaction API

The OS2200TIPTransaction class is used to open a TIP session and call TIP or HVTIP transactions. In most cases, transactions using Display Processing System (DPS) forms can also be called using the DPS connector interface.

A TIP session is established by opening a connection between the AIS Agent and CITA, using a port number defined in the CITA configuration. Each port number in the CITA configuration is associated with an application group and therefore controls which transactions can be called. Multiple TIP sessions can be established using different port numbers if an application needs to call transactions in different application groups.

Note: *See Appendix B, Example CITA Configuration, for an example of a CITA configuration.*

The OS2200Record class is used to pass input to a transaction and receive output from a transaction. An application is responsible for formatting the transaction input and processing the transaction output.

Note: *Transaction input and output cannot contain binary data.*

6.6.1. Opening a TIP Session

To open a TIP session an application creates an OS2200TIPTransaction class instance. The port number provided as input to the class constructor must be defined in a CITA configuration.

```
public OS2200TIPTransaction(OS2200Connection connection,
    int port)
```

Example: Opening a TIP Session

This example demonstrates how to open a TIP session using port number 2405.

```
OS2200TIPTransaction tran = new OS2200TIPTransaction(
    conn, 2405);
```

6.6.2. Closing a TIP Session

Since a TIP session is a shared resource, it is important that an application closes the TIP session when it is no longer needed.

Example: Closing a TIP Session

This example demonstrates how to close a TIP session.

A try-catch-finally construct is used in this example to guarantee that the TIP session is closed even if an exception is thrown.

```
OS2200TIPTransaction tran = null;

try
{
    tran = new OS2200TIPTransaction(conn, 2405);

    // Call a TIP transaction
}
catch(Exception e)
{
    // Handle exception.
}
finally
{
    if (tran != null)
    {
        tran.Close();
    }
}
```

6.6.3. Calling a TIP Transaction

After a TIP session has been established, an application can call one or more TIP transactions. If a TIP transaction is expecting input, then an OS2200Record class instance must be created and populated with the input data. An OS2200Record class instance must also be created to accept the output produced by the TIP transaction. The Call method returns the byte length of the TIP transaction output, which can be used by an application to determine if the expected output was received.

```
Call(string tranCode, OS2200Record input, OS2200Record output)

Call(string tranCode, OS2200Record output)
```

Example: Calling a TIP Transaction

This example demonstrates how to call a TIP transaction. The TIP transaction in this example accepts three strings as input and returns the same three strings as output. The registered transaction code is ECHOT.

Structure of the input record

```
01  Inrecord.
    05  FILLER         PIC X(12).
    05  IN-DATA-1      PIC X(6)  USAGE DISPLAY.
    05  IN-DATA-2      PIC X(10) USAGE DISPLAY.
    05  IN-DATA-3      PIC X(20) USAGE DISPLAY.
```

Structure of the output record

```
01  Outrecord.
    05  FILLER         PIC X(9)  USAGE DISPLAY VALUE IS 'TEXT'.
    05  OUT-DATA-1     PIC X(6)  USAGE DISPLAY.
    05  OUT-DATA-2     PIC X(10) USAGE DISPLAY.
    05  OUT-DATA-3     PIC X(20) USAGE DISPLAY.
```

The first step is to create an OS2200Record class instance and format the input data. An Encoding class instance is used in this example to convert string values to an array of bytes. The array of bytes is written to the OS2200Record at the appropriate offsets to match the structure of the input record.

```
Encoding encoding = Encoding.ASCII;
OS2200Record inRecord = new OS2200Record();
inRecord.WriteBytes(12, encoding.GetBytes("AAAA  "));
inRecord.WriteBytes(18, encoding.GetBytes("TESTDATA  "));
inRecord.WriteBytes(28, encoding.GetBytes(
    "ASDFGHJKLZqwertyuiop" ) );
```

Next, another OS2200Record class instance is created to receive the TIP transaction output and the ECHOT TIP transaction is called.

```
OS2200Record outRecord = new OS2200Record();
int outputLen = tran.Call("ECHOT", inRecord, outRecord);
```

Finally, the output returned by the TIP transaction is processed. The length returned by the Call method is checked to verify the expected output was returned. An encoding class is used to convert the bytes of output data to strings.

```
if ( outputLen == 45 )  // 45 is the byte length of Outrecord
{
    outRecord.Position = 9;
    Console.WriteLine("Transaction output: ");
    Console.WriteLine("Out_Data_1: " +
        encoding.GetString( outRecord.ReadBytes(6)));
    Console.WriteLine("Out_Data_2: " +
        encoding.GetString( outRecord.ReadBytes(10)));
    Console.WriteLine("Out_Data_3: " +
        encoding.GetString( outRecord.ReadBytes(20)));
}
else
{
    outRecord.Position = 0;
    Console.WriteLine("Transaction error: " +
        encoding.GetString(outRecord.ReadBytes(outputLen)));
}
```

6.7. OS 2200 Batch Run API

The OS2200BatchRun class starts a batch run that executes a collection of user-supplied control statements. The output produced by the batch run is captured and returned to the caller as a StreamReader class instance.

6.7.1. Starting a Batch Run

The OS2200BatchRun class exposes a *Start* method used to start a batch run.

```
StreamReader Start(IEnumerable<string> controlStatements,
                   TimeSpan runTimeout, string runId)
```

The *Start* method accepts as input a collection of control statements, a timeout interval, and a run ID. When called, the *Start* method waits until the batch run has completed or until the specified timeout interval has been exceeded.

The maximum length of a control statement is 80 characters, including the terminating newline character. Use a semicolon ';' to indicate a control statement is continued on the next line. You can include a semicolon in a control statement where spaces are allowed or following a delimiter.

Example: Starting a Batch Run

This example demonstrates how to start a batch run that compiles a COBOL source element. It also displays the output produced by the batch run.

```
// Define a 1 second time interval for the batch run to complete.
TimeSpan runTimeout = new TimeSpan(0,0,1);

//Define the control statement to execute in the batch run
List<string> controlStatements = new List<string>();

controlStatements.Add("@USE SRC,AISTRAN*SRC.");
controlStatements.Add("@USE ABS,AISTRAN*ABS$.");
controlStatements.Add("@ASG,AXZ ABS.");
controlStatements.Add("@UCOB,E SRC.ECHOT/COB,ABS.ECHOT/O,,,;");
controlStatements.Add("NO-ALLOC,APPLICATION/APPSVN");

OS2200BatchRun batchRun = new OS2200BatchRun(connection);

try
{
    // Start the batch run
    StreamReader reader =
        batchRun.Start(controlStatements, runTimeout, "MYID");

    // Display the output produced by the batch run
    while (!reader.EndOfStream)
    {
        Console.WriteLine(reader.ReadLine());
    }

    reader.Dispose();
}
catch (OS2200BatchRunTimeoutException e)
{
    Console.WriteLine("Error: " + e.Message);

    // Display the output produced by the batch run
    Console.WriteLine(e.BatchRunOutput);
}
```

Note: *Batch commands inside an OS2200BatchRun API run independently, so the run considers the user's default directory-ID as working directory (typically same as project ID, but depends on the system configuration).*

If a new current working directory has to be set for that batch run, it must be an explicit action within the batch run, specifying a control statement.

*For ex: controlStatements.Add("@CAT,P AISCS*AISDEL1."); where, 'AISCS' is the directory which contains the file AISDEL1.*

6.8. OS 2200 Mass Storage File API

The OS2200MassStorageFile class defines methods used to perform various maintenance operations common to OS 2200 data files and program files. The methods perform operations analogous to FURPUR control statements.

The methods are used to

- Acquire information about a file.
- Change file attributes.
- Delete a file.
- Erase a file.

6.8.1. Acquiring Mass Storage File Information

The OS2200MassStorageFile class exposes an *AcquireCatalogedFileInfo* method used to obtain information about a mass storage file, such as the account, project-id, number of file cycles, and last reference date.

```
OS2200CatalogedFileInfo AcquireCatalogedFileInfo()
```

Note: *Some information associated with a file is available to the file owner only.*

Example: Acquiring Mass Storage File Information

This example demonstrates how to acquire information associated with a program file.

```
OS2200ProgramFile pfFile = new OS2200ProgramFile(connection,
    "EXAMPLE", "PROGFILE");

OS2200CatalogedFileInfo info = pfFile.AcquireCatalogedFileInfo();
```

6.8.2. Changing File Attributes

The OS2200MassStorageFile class exposes a *Change* method used to change the attributes defined for a mass storage file.

```
void Change(OS2200ChangeFileArgs arguments)
```

The OS2200ChangeFileArgs class specifies the new file attributes.

Example: Changing Mass Storage File Attributes

This example demonstrates how to change the attributes associated with a data file.

```
OS2200DataFile file = new OS2200DataFile(connection,
    "EXAMPLE", "DATAFILE");

OS2200ChangeFileArgs args = new OS2200ChangeFileArgs();

args.ReadOnly = OS2200AttributeIndicatorType.Enabled;
args.RolloutProhibited = OS2200AttributeIndicatorType.Enabled;

file.Change(args);
```

6.8.3. Changing File Cycle Limit

The OS2200MassStorageFile class exposes a *ChangeFileCycleLimit* method used to change the file cycle limit for a file cycle series of mass storage files.

Note: *The ChangeFileCycleLimit method can delete files if the new file cycle limit is less than the maximum file cycle in the file cycle series of files.*

```
OS2200ChangeFileCycleResults ChangeFileCycleLimit(
    ushort maximumFileCycles, bool allowDeletion)
```

Example: Changing Mass Storage File Cycle Limit

This example demonstrates how to change the file cycle limit for a data file. In this example, only the first cycle of the data file is retained.

```
OS2200DataFile file = new OS2200DataFile(connection,
    "EXAMPLE", "DATAFILE");

file.ChangeFileCycleLimit(1, true);
```

6.8.4. Changing File Read and Write Keys

The OS2200MassStorageFile class exposes a *ChangeKeys* method used to change or clear the read and write keys defined for a mass storage file.

Note: *Passing spaces clears a read or write key.*

```
void ChangeKeys(string readKey, string writeKey)
```

Example: Changing Mass Storage File Read/Write Keys

This example demonstrates how to change the read and write keys for a data file. In this example, the read and write keys are cleared.

```
OS2200DataFile file = new OS2200DataFile(connection,
    "EXAMPLE", "DATAFILE");

file.ReadKey = "RDKEY ";
file.WriteKey = "WRTKEY";

file.ChangeKeys("      ", "      ");
```

6.8.5. Changing File Name

The OS2200MassStorageFile class exposes a *ChangeName* method used to change the name of a mass storage file.

```
void ChangeName(OS2200ChangeFileNameArgs arguments)
```

The OS2200ChangeFileNameArgs class specifies a new file qualifier and/or new file name.

Example: Changing Mass Storage File Name

This example demonstrates how to change the name of a data file.

```
OS2200DataFile file = new OS2200DataFile(connection,
    "EXAMPLE", "DATAFILE");

OS2200ChangeFileNameArgs nameArgs =
    new OS2200ChangeFileNameArgs();

nameArgs.ChangeFileName = true;
nameArgs.ChangeQualifier = true;
nameArgs.NewFileName = "DATAFILE2";
nameArgs.NewQualifier = "EXAMPLE2";

file.ChangeName(nameArgs);
```

6.8.6. Changing File Security

The OS2200MassStorageFile class exposes a *ChangeSecurity* method used to change the security attributes defined for a mass storage file.

```
void ChangeSecurity(OS2200ChangeFileSecurityArgs arguments)
```

The OS2200ChangeFileSecurityArgs class specifies new file security attributes.

Example: Changing Mass Storage File Security

This example demonstrates how to change the security attributes of a data file.

```
OS2200DataFile file = new OS2200DataFile(connection,
    "EXAMPLE", "DATAFILE");

OS2200ChangeFileSecurityArgs secArgs =
    new OS2200ChangeFileSecurityArgs();
```

```
secArgs.Privacy = OS2200FilePrivacyType.SemiPrivate;
secArgs.AccessControlRecordName = "CTRLAC";

file.ChangeSecurity(secArgs);
```

6.8.7. Deleting a Mass Storage File

The OS2200MassStorageFile class exposes a *Delete* method used to delete a mass storage file.

```
void Delete(bool exclusiveAssign, bool clearStorage)
```

The *Delete* method can optionally zero-fill the storage occupied by a file before it is deleted.

Note: *Exclusively assign the mass storage file when zero-filling the storage it occupies.*

Example: Deleteing a Mass Storage File

This example demonstrates how to delete a program file. The example also zero-fills the storage occupied by the program file.

```
OS2200ProgramFile pfFile = new OS2200ProgramFile(connection,
    "EXAMPLE", "PROGFILE");

pfFile.Delete(true, true);
```

6.8.8. Erasing a Mass Storage File

The OS2200MassStorageFile class exposes an *Erase* method used to erase the contents of a mass storage file.

```
long Erase(OS2200StorageReleaseType storageToRelease)
```

The *Erase* method can optionally release storage occupied by a file.

Example: Erasing a Mass Storage File

This example demonstrates how to erase a program file. The example also releases the storage beyond the initial reserve defined for the program file.

```
OS2200ProgramFile pfFile = new OS2200ProgramFile(connection,
    "EXAMPLE", "PROGFILE");

pfFile.Erase(OS2200StorageReleaseType.AllButInitialReserve);
```

6.8.9. OS 2200 Mass Storage File API Samples

For sample applications that demonstrate usage of the OS2200MassStorageFile methods, see the Samples folder in your installation directory.

6.9. OS 2200 Data File API

The OS2200DataFile class extends the OS2200MassStorageFile class. The OS2200DataFile class exposes a static method used to catalog an OS 2200 data file. It also inherits several methods used to perform various maintenance operations on a data file.

The methods are used to

- Catalog a data file.
- Delete a data file.
- Erase a data file.
- Change various data file attributes.

Note: *See 6.8 OS 2200 Mass Storage File API for examples of using methods inherited from the OS2200MassStorageFile class.*

6.9.1. Cataloging a Data File

The OS2200DataFile class exposes a static method used to catalog a data file. The caller of the method becomes the owner of the data file.

```
OS2200DataFile Catalog(OS2200Connection connection,
                       OS2200CatalogFileArgs arguments)
```

The OS2200CatalogFileArgs class specifies the file name and defines the attributes used to catalog the data file.

Example: Cataloging a Data File

This example demonstrates how to catalog an OS 2200 data file.

```
OS2200CatalogFileArgs args =
    new OS2200CatalogFileArgs("EXAMPLE", "DATAFILE");

args.DeviceType = "F";
args.Public = true;
args.MaximumLength = 9999;
args.CycleType = OS2200FileCycleType.Absolute;
args.CycleNum = 100;

OS2200DataFile file = OS2200DataFile.Catalog(connection, args);
```

6.9.2. OS 2200 Data File API Samples

For sample applications that demonstrate usage of the OS2200DataFile class, see the Samples folder in your installation directory.

6.10. OS 2200 Program File API

The OS2200ProgramFile class extends the OS2200MassStorageFile class. The OS2200ProgramFile class exposes a static method used to catalog an OS 2200 program file and perform various operations on program file elements. It also inherits several methods used to perform various maintenance operations on a program file.

The methods are used to

- Catalog a program file.
- Delete a program file.
- Erase a program file.
- Change various program file attributes.
- Change various program file element attributes.
- Copy and delete program file elements.

Note: *See 6.8 OS 2200 Mass Storage File API for examples of using methods inherited from the OS2200MassStorageFile class.*

6.10.1. Cataloging a Program File

The OS2200ProgramFile class exposes a static method used to catalog a program file. The caller of the method becomes the owner of the program file.

```
OS2200ProgramFile Catalog(OS2200Connection connection,
                          OS2200CatalogFileArgs arguments)
```

The OS2200CatalogFileArgs class specifies the file name and defines the attributes used to catalog the program file.

Example: Cataloging a Program File

This example demonstrates how to catalog an OS 2200 program file.

```
string[] packIds = { "3XG70", "3XG71" };

OS2200CatalogFileArgs args =
    new OS2200CatalogFileArgs( "EXAMPLE", "PROGFILE" );

args.DeviceType = "F";
args.Public = true;
args.MaximumLength = 9999;
args.CycleType = OS2200FileCycleType.Relative;
args.CycleNum = 1;
args.PackId = packIds;

OS2200ProgramFile file =
    OS2200ProgramFile.Catalog(connection, args);
```

6.10.2. Acquiring Basic Program File Information

The OS2200ProgramFile class exposes an *AcquireBasicProgramFileInfo* method used to obtain information associated with a program file.

```
OS2200BasicProgramFileInfo AcquireBasicProgramFileInfo(
    bool includeSummaryInfo)
```

The *AcquireBasicProgramFileInfo* method can optionally return summary information, such as the number of each element type, number of deleted elements, and dead space size.

Example: Acquiring Basic Program File Information

This example demonstrates how to obtain basic information associated with a program file.

```
OS2200ProgramFile pfFile = new OS2200ProgramFile(connection,
    "EXAMPLE", "PROGFILE");

OS2200BasicProgramFileInfo fileInfo =
    pfFile.AcquireBasicProgramFileInfo(true);
```

6.10.3. Acquiring Program File Element Information

The OS2200ProgramFile class exposes an *AcquireElementInfo* method used to obtain information for one or more program file elements.

```
OS2200ElementInfoCollection AcquireElementInfo(
   OS2200AcquireElementInfoArgs arguments)
```

The OS2200AcquireElementInfoArgs class is used to identify the elements for which information is obtained. The class also contains a property used to specify if information is obtained for undeleted elements, deleted elements, or both.

Note: *The element name and version name can contain a combination of static characters and the wild-card characters '*' and '?'. The '*' character is used to match zero or more characters. The '?' character is used to match any single character.*

The *AcquireElementInfo* method returns a read-only collection of OS2200ElementInfo class instances. An OS2200ElementInfo class instance is returned for each program file element that matched the criteria specified in the OS2200AcquireElementInfoArgs class instance.

Table 6–3 shows the names of subclasses that extend the OS2200ElementInfo class. A different subclass is used to return information for each of the four element types.

Table 6–3. Program File Element Types

Element Type	Information Subclass
Absolute	OS2200AbsoluteElementInfo
Omnibus	OS2200OmnibusElementInfo
Relocatable	OS2200RelocatableElementInfo
Symbolic	OS2200SymbolicElementInfo

Example: Acquiring Program File Element Information

This example demonstrates how to obtain information about program file elements. The example obtains information for all symbolic and omnibus elements having a name that starts with "test".

```
OS2200ProgramFile pfFile = new OS2200ProgramFile(connection,
   "EXAMPLE", "PROGFILE");

OS2200AcquireElementInfoArgs eltInfoArgs = new
   OS2200AcquireElementInfoArgs("test*");

eltInfoArgs.SymbolicElements = true;
eltInfoArgs.OmnibusElements = true;

OS2200ElementInfoCollection elementInfo =
   pfFile.AcquireElementInfo(eltInfoArgs);

// Print the name of each element in the resulting collection.

foreach(OS2200ElementInfo info in elementInfo)
{
   Console.WriteLine("Element name: " + info.Name);
}
```

6.10.4. Changing Program File Elements

The OS2200ProgramFile class exposes a *ChangeElements* method used to change the attributes associated with one or more program file elements.

```
OS2200ChangeElementResults ChangeElements(
    OS2200ChangeElementArgs arguments)
```

The OS2200ChangeElementArgs class identifies an element name and optionally a version name of the elements impacted by the change operation. The class also contains properties used to specify the type of change to perform.

Note: *The element name and version name can contain a combination of static characters and the wild-card characters '*' and '?'. The '*' character matches zero or more characters. The '?' character matches any single character.*

Example: Changing Program File Elements

This example demonstrates how to change the attributes associated with program file elements. The example changes the cycle limit and date/time update for all symbolic elements that start with "test".

```
OS2200ProgramFile pfFile = new OS2200ProgramFile(connection,
    "EXAMPLE", "PROGFILE");

OS2200ChangeElementArgs changeArgs = new OS2200ChangeElementArgs(
    "test*");

changeArgs.SymbolicElements = true;
changeArgs.ChangeDateTime = true;
changeArgs.NewCycleLimit = 4;

OS2200ChangeElementResults results =
    pfFile.ChangeElements(changeArgs);
```

6.10.5. Copying Program File Elements

The OS2200ProgramFile class exposes a *CopyElements* method used to copy one or more elements to a different program file.

```
OS2200CopyElementResults CopyElements(
    OS2200MassStorageFile destinationFile,
    OS2200CopyElementArgs arguments)
```

The OS2200CopyElementArgs class identifies an element name and optionally a version name of the elements impacted by the copy operation.

Note: *The element name and version name can contain a combination of static characters and the wild-card characters '*' and '?'. The '*' character matches zero or more characters. The '?' character matches any single character.*

Example: Copying Program File Elements

This example demonstrates how to copy program file elements to a different program file. The example copies all absolute elements with a name that start with "test" to a different program file.

```
OS2200ProgramFile srcFile = new OS2200ProgramFile(connection,
    "EXAMPLE", "PROGFILE");

OS2200ProgramFile destFile = new OS2200ProgramFile(connection,
    "EXAMPLE", "DESTPROGFILE");

OS2200CopyElementArgs copyArgs = new OS2200CopyElementArgs();
copyArgs.AbsoluteElements = true;
copyArgs.ElementName = "test*";

OS2200CopyElementResults copyResults =
    srcFile.CopyElements(destFile, copyArgs);
```

6.10.6. Deleting Program File Elements

The OS2200ProgramFile class exposes a *DeleteElements* method used to delete one or more program file elements.

```
public long DeleteElements(OS2200DeleteElementArgs arguments)
```

The OS2200DeleteElementArgs class identifies an element name and optionally a version name of the elements impacted by the delete operation.

Note: *The element name and version name can contain a combination of static characters and the wild-card characters '*' and '?'. The '*' character matches zero or more characters. The '?' character matches any single character.*

Example: Deleting Program File Elements

This example demonstrates how to delete program file elements. The example deletes all relocatable and omnibus elements from a program file.

```
OS2200ProgramFile pfFile = new OS2200ProgramFile(connection,
    "EXAMPLE", "PROGFILE");

OS2200DeleteElementArgs args = new OS2200DeleteElementArgs();
args.RelocatableElements = true;
args.OmnibusElements = true;

pfFile.DeleteElements(args);
```

6.10.7. Prep a Program File

The OS2200ProgramFile class exposes a *Prep* method used to build a relocatable or an object-module entry point table in a program file.

```
OS2200PrepResults Prep(OS2200EntryPointTableType epTableType,
    bool overwriteEPTable)
```

Example: Prep a Program File

This example demonstrates how to prep a program file.

```
OS2200ProgramFile pfFile = new OS2200ProgramFile(connection,
    "EXAMPLE", "PROGFILE");

OS2200PrepResults results =
    pfFile.Prep(OS2200EntryPointTableType.Both, true);
```

6.10.8. OS 2200 Program File API Samples

For sample applications that demonstrate usage of the OS2200ProgramFile class, see the Samples folder in your installation directory.

Section 7
Troubleshooting

This section provides information to help diagnose problems with Application Integration Services. It provides an overview of the items you should review when debugging a problem and the commands and expected results. It does not provide detailed command syntax or instructions. This information is available in other manuals.

7.1. Debugging the Inability to Establish a Connection

The inability to establish a connection with the ClearPath system could be caused by one of the following problems:

- Invalid user-id and password

  Ensure the credentials you are using are valid on the system you are attempting to use. Logging on using a terminal emulator will enable you to verify your credentials. Remember that the password may be case sensitive.

- The connection name is incorrect

  Verify that the connection name you are using in your program matches the ClearPathHosts.config file. The entry is case sensitive.

- The host name is incorrect

  Verify that the host name or IP address in the ClearPathHosts.config file is correct. Your program must provide a connection name not a host name. The connection name must be in the ClearPathHosts.config file and the entry must include a valid IP address or host name. If you are using an SSL connection, the host name must be a fully qualified domain name.

- The port in the ClearPathHosts.config file is incorrect

  If you are using a cleartext connection, you must use the cleartext port number configured in the AIS Agent. If you are using a secure connection, you must use the secure port number configured in the AIS Agent.

- TCP/IP connection between your Windows system and the ClearPath system is not operational

  Ping the ClearPath system from your Windows system to ensure the link is operational.

- OS 2200 component is not operational.

  See 7.2 Verifying the Operation of the ClearPath OS 2200 Components for details.

7.2. Verifying the Operation of the ClearPath OS 2200 Components

Use the following topics to verify the operation of the ClearPath OS 2200 components:

7.2.1. ClearPath OS 2200 Connectivity Services is running

Verify that the ClearPath OS 2200 Connectivity Services is running by entering the CS2200 STATUS console command using either the CS2200 background run ID or the CS2200 KEYIN name as configured on the ADMIN statement of the Connectivity Services configuration.

Example:

```
CONASV STATUS
```

The output returned is similar to the following:

```
CSF101018: Status of CS background run
     Original Runid - CONASV   Generated Runid - CONASV
     3R1 3-15 3-15 Mode A Site RS05
     (150109 1336:08) Wed Jan 21 11:20:48 2015
     CPComm[A] receive activity is attached - level 106
     CS master activity is        active
     General logging is set to    OFF
     Log file in use SYS$LIB$*CS$LOG(1)
```

From the statement: "CPComm[A] receive activity is attached - level 106," you can see that Connectivity Services is running and attached to CPComm Mode A.

If the message displayed shows, "CPComm[A] receive activity is NOT attached", then the communications product for your system (CPComm or CPCommOS) is not running.

If there is no response to the STATUS command, Connectivity Services is not running. Examine the CS$LOG file, if one exists, and the console message display or log to determine the reason it is not running. Also, Connectivity Services requires the ASIS (FLEX) product for user authentication. Since Connectivity Services is unusable without authentication, Connectivity Services displays a message and terminates at initialization time if ASIS is not installed or running. View the console display/log for any Connectivity Services messages describing any initialization failures. If necessary, install and start ASIS and then start Connectivity Services. After starting Connectivity Services, verify that it is running by entering the CS2200 STATUS console command. If ASIS is terminated during Connectivity Services execution, then a log entry is made in the Connectivity Services LOG file when a client authentication request is received and it fails. Review the Connectivity Services LOG file for any authentication errors to determine the cause.

If the name on the Connectivity Services CPCOMM statement does not match the PROCESS name in the Communications Services configuration, Connectivity Services terminates with the following error:

```
CSF100002: CPComm's configuration does not allow CS to Attach.
```

Check the configuration and restart Connectivity Services.

7.2.2. AIS Agent is Running

Verify that the AIS Agent is running by entering the STATUS console command using the AIS Agent keyin_id value as defined in the AIS Agent configuration file. The output of this command provides state information for the AIS Agent and configured connection listeners.

An AIS Agent can be configured to have a connection listener for cleartext data transmission, a connection listener for secure (SSL) data transmission, or both. A unique port number is associated with each connection listener.

Example

```
AGNTA STATUS
```

The output returned is similar to the following:

```
OS 2200 AIS Agent (3.1.0):
   start time:                   Wed Jan 21 12:25:10 2015
   state:                        AGENT_RUNNING
   total requests processed:     0
   last request timestamp:       none
Connectivity Services Information:
   Mode:                         A
   IP address:                   (0.0.0.0)
   Connection Listener[0]:
      port, session ID:          43745, 014042000003
      state:                     LISTENER_RUNNING
      interface status:          API-NORMAL
      workers assigned:          0
    Connection Listener[1]:
      port, session ID:          43746, 014043000007
      state:                     LISTENER_RUNNING
      interface status:          API-NORMAL
      workers assigned:          0
```

The first 5 lines of the STATUS command output gives general information about the state of the AIS Agent, such as when the agent was last started, the total number of client requests processed, and when the last request was received.

The remaining line of output provides information about Connectivity Services and the configured connection listeners. In the above example, the AIS Agent is configured to use Mode A Connectivity Services and the listeners will accept requests on any configured IP addresses.

The first connection listener (Connection Listener[0]) is used for cleartext data transmission and the second connection listener (Connection Listener[1]) is used for secure (SSL) data transmission.

Each connection listener section displays the configured port number, the Connectivity Services session ID, the current listener state, and the number of active client sessions initiated through the connection listener.

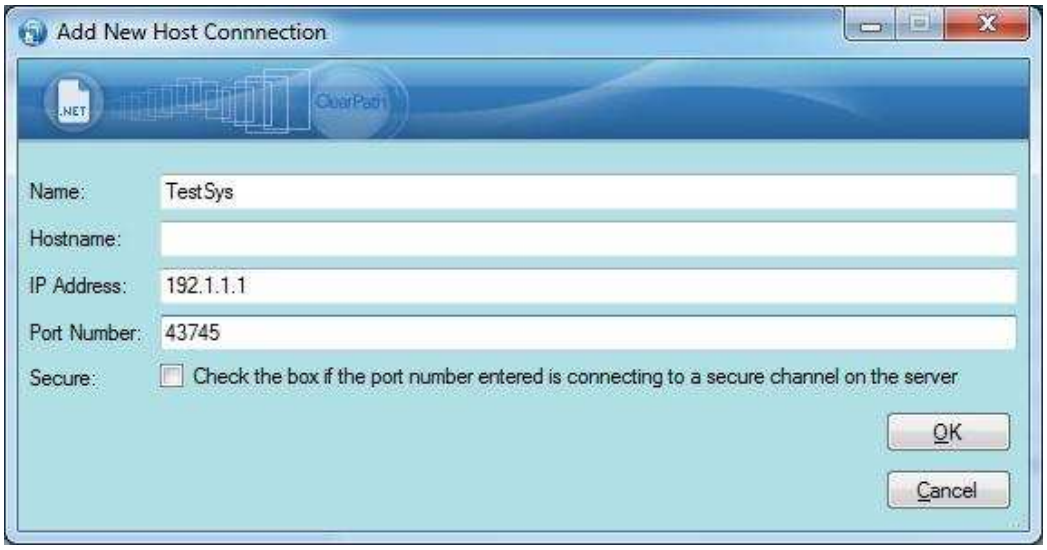
In the following example output, the state indicates the connection listeners are still trying to open a session using the configured port number. This could be an indication that Connectivity Services is currently not running, or there is a problem with some portion of the communications configuration. When the problem is resolved, the AIS Agent initiates a listener session and the state changes to LISTENER_RUNNING.

```
Connectivity Services Information:
   Mode:                          A
   IP address:                    (0.0.0.0)
   Connection Listener[0]:
      port, session ID:           43745, 000000000000
      state:                      LISTENER_CONNECTING
      interface status:           API-ACCESS-ERROR
      workers assigned:           0
   Connection Listener[1]:
      port, session ID:           43746, 000000000000
      state:                      LISTENER_CONNECTING
      interface status:           API-ACCESS-ERROR
      workers assigned:           0
```

7.3. Examining the ClearPathHosts Configuration File

The Configuration Utility creates entries in the ClearPathHosts.config file that is used when a connection to the ClearPath system is established. The name used in your application must match the connection in the configuration file. For example, the connection below uses a connectioni name of TestSys. This name must appear as shown in the following figure.

```
OS2200Connection conn = new OS2200Connection("TestSys",
                                             "MyUserId",
                                             "MyPassword");
```



7.4. Resolving a ConfigurationErrorsException

If a ConfigurationErrorsException is returned, one of the configuration files contains an error. The exception may direct you to the configuration file causing the error and it may provide information about the error within the configuration file.

If the error is in the

- ClearPathParentSource.config file

  Replace your existing ClearPathParentSource.config file with the template file that was installed during installation.

- ClearPathHosts.config file

  Use the Configuration Utility to create a new configuration file.

Section 8
Support

Unisys brings together powerful hardware and software, satellite links, multi-language support, and a multitude of professionals to deliver a round-the-clock, global service. At Unisys, we understand that your business simply cannot afford to wait for support. That is why all Unisys support services are fully integrated to provide you with real-time access to the critical information you need.

8.1. Using Support eService

For any questions or concerns about your Unisys products, turn to Unisys eService, your complete support information source. eService, the Unisys electronic support service portal, provides technical information through the World Wide Web for Unisys customers who are covered under a Support Service Agreement. You can access the Unisys eService site at

http://www.service.unisys.com

Unisys eService puts a wealth of technical information on hardware and software product support services at your fingertips. Unisys technical specialists provide updated symptom and solution information, including frequently asked questions. You can also explore the Unisys Web site to learn more about services designed to add value to your information technology operations.

Product Name for Service Requests

When you sign in to eService, please log all service requests for Application Integration Services against the following product name:

```
AIS2200
```

Select the product component that seems most appropriate for your request.

8.2. Obtaining Direct Telephone Support

Direct telephone support is yet another support option offered by Unisys. If you are located within the continental United States or Canada, you can call one of the following toll-free numbers during the times indicated in your service agreement:

- United States 800.328.0440 (prompt 4)
- Canada (English) 800.387.6181
- Canada (French) 800.361.8097

Customers outside the continental United States or Canada should contact their local Unisys office.

8.3. Application Integration Services Support

If you have a problem that requires Unisys customer support, contact your designated Unisys Customer Support Center. If no one else reported the problem, the support center authorizes a UCF.

When reporting a software problem, please include the following information to help Unisys isolate your problem:

- Level of Application Integration Services you are using.
- Description of what happened immediately before the error occurred.
- Remote System Support (RSS) files generated by the AIS Agent background run. This includes a copy of the following files:
  - AIS$*n**AIS*n*$PRINT
  - AIS$*n**AIS$LOG
  - AIS$*n**AIS$DIAG

    This file is useful to determine the cause of a contingency in the AIS Agent.

  n is an alpha character A through D that indicates the installation mode.

Appendix A
Development versus Runtime Environment

Table A–1 lists the Microsoft operating system that developers can use to develop and compile a .NET application, and the corresponding Microsoft operating systems under which that application runs.

For both development and runtime environment, the Microsoft .NET Framework version is 4.5.

Table A–1. Application Development Environment and the Corresponding Runtime Environments

Development Environment	Runtime Environment
Windows 10	Windows 10 Windows Server 2012 Windows Server 2008 R2
Windows 7	Windows 7 Windows Server 2008 R2 Windows Server 2012
Windows 8 or Windows 8.1	Windows 8 or Windows 8.1 Windows Server 2008 R2 Windows Server 2012
Windows Server 2008 R2	Windows Server 2008 R2 Windows Server 2012
Windows Server 2012	Windows Server 2012

Appendix B
Example CITA Configuration

Before you can use the OS2200TIPTransaction class to call TIP/HVTIP transactions, you must define a suitable Communications Interface for Transactions Applications (CITA) configuration.

Use the following CITA configuration statements as a template for defining a CITA configuration that is compatible with ClearPath Application Integration Services. See the *Communications Interface for Transaction Applications Configuration and Operations Guide* for a complete description of CITA configuration statements.

```
ADMIN KEYIN,CITATX MAX-QUEUE-ENTRIES,10 ;
      TRACE,INPUT,OFF TRACE,OUTPUT,OFF

APPLICATION,7

L5INSET,IN TCODE-OFFSET,0 HDRLEN,32 DL-OFFSET,7 DL-TYPE,S ;
           DL-LEN,6 DL-INCL-HDR,YES MAXSIZE,10000 PASSHDR,NO

L5OUTSET,OUT HDRLEN,32 DL-OFFSET,7 DL-TYPE,S DL-LEN,6 ;
             DL-INCL-HDR,YES

LISTEN-NORM,2700 PIDPOOL,400 APNUM,7 LPORT,2700 LIP,127.0.0.1

PIDPOOL,400,100 L5INSET,IN L5OUTSET,OUT ;
                AUTO-VALIDATION,YES

LISTEN-NORM,2800 PIDPOOL,500 APNUM,7 LPORT,2800 LIP,127.0.0.1

PIDPOOL,500,100 L5INSET,IN L5OUTSET,OUT ;
                TERM-TYPE,129 ;
                AUTO-VALIDATION,YES
```

The *APPLICATION* configuration statement represents one Message Control Bank (MCB) and its associated application group. A CITA configuration must define at least one *APPLICATION* configuration statement.

The *L5INSET* configuration statement defines the structure of input messages. ClearPath Application Integration Services requires the transaction code to be in the first 6 bytes of the header and the header length must be 32 bytes. The user data length can be up to 6 digits and starts in byte 7 of the header. The header length is included in the user data length.

The *L5OUTSET* configuration statement defines the header that CITA appends to each output message. ClearPath Application Integration Services requires the header length to be 32 bytes. The user data length can be up to 6 digits and starts in byte 7 of the header. The header length is included in the user data length.

The *LISTEN-NORM* configuration statement defines the port number and IP address used to establish a new session.

Note: *The LIP parameter should match the value defined for the AIS Agent comapi_loopback configuration parameter (see 3.2.7 comapi_loopback).*

The *PIDPOOL* configuration statement defines a pool of PIDs. Each new CITA session is associated with a PID from the pool.

If ClearPath Application Integration Services is used to call a transaction that uses Display Processing System (DPS) forms, set the *TERM-TYPE* parameter to 129. This is a special terminal type that causes DPS to use a message format suitable for communicating with an external client program.

Note: *Set the AUTO-VALIDATION parameter to YES when using ClearPath Application Integration Services to call a TIP/HVTIP transaction.*

8230 0815-003