About Communication Adapters

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 820–5068
07/08/08

080805@20490

Contents

About Communication Adapters

The topics listed here provide information about Communication Adapters. If you have any questions or problems, see the Java CAPS web site at `http://goldstar.stc.com/support`.

About Sun Adapter for CICS

This topic provides conceptual information about CICS and its Sun Java CAPS Adapter.

CICS Transaction Server

IBM's Customer Information Control System (CICS), is IBM's most widely used proprietary, transaction monitor. CICS provides connectivity and online transaction management for mission-critical applications. It supports real-time distributed processing environments and online transaction processing (OLTP). According to IBM, CICS handles more than thirty billion transactions, processing more than one trillion dollars, per day.

CICS is the premier OLTP (On-Line Transaction Processing) products from IBM. It is used to access many file systems and databases including third party products. For IBM product, it

interfaces with DB2, VSAM and IMS/DB. For non-IBM products, CICS interfaces with IDMS and DATACOM, to name a few. Most applications in CICS are written in COBOL, although it supports other languages such as PL/1.

OLTP systems provide accurate, up-to-date information within seconds, from terminals that give direct access to data held as either files or databases. CICS provides a company with numerous transaction processing and resource management functions, allowing the user to concentrate on developing application programs that meet that organization's specific business needs. CICS controls OLTP application programs in a distributed transaction processing (DTP) environment. CICS handles interactions between the terminal user and the application programs. Programs gain access to the CICS facilities with straightforward, high-level commands.

CICS provides:

- Communication functionality to terminals and systems that are required by application programs.
- Control of concurrently running programs that serve online users.
- Facilities for accessing databases and files.
- The ability to communicate with other CICS family members using Transmission Control Protocol/Internet Protocol (TCP/IP).
- Interactive facilities to configure specific systems.
- Recovery processing and data protection, should a problem occur.

The Sun Adapter for CICS

The Sun Adapter for CICS is an interface that enables remote bidirectional calls to CICS transactional programs. The CICS Adapter includes a build tool, the Cobol Copybook Converter, that creates an Object Type Definition (OTD) from a Cobol Copybook file and generates Java CAPS ESB Object Type Definitions for use within the Java CAPS ESB environment. The Copybook file structures are passed into the CICS environment as the data buffer (Commarea).

A fixed Object Type Definition, the CICS_eWay.CICSClient OTD, designed to expose various essential portions of the CICS Java API, provides available methods and properties, as well as access to all message attributes.

The Adapter can use either the IBM CICS Transaction Gateway version 5.1, 6.0, 6.0.1, or 6.1, or the Sun CICS Listener as the underlying connection transport for accessing CICS z/OS transactions.

IBM CICS Transaction Gateway (CTG)

CTG provides an API (the External Call Interface or ECI) to call CICS transactions on the mainframe. The ECI allows a non-CICS application program to call a CICS program in a CICS server. Sun's CICS Adapter uses this ECI method to connect to CICS. The CICS Adapter connects to CICS with CTG running on a local-host, on a second computer, or on the mainframe.

Note – When using the CICS Transaction Gateway transport, data sent to CICS must be padded with spaces, if necessary, to match the full size of the commarea.

Sun CICS Listener

The CICS Adapter connects to the IBM CICS Listener running on z/OS via the TCP/IP Sockets. The Listener accepts the incoming request and spawns a new process handing the socket connection off to the newly created process via TCP/IP givesocket()/takesocket() function calls. The spawned process invokes the user written CICS application program through an EXEC CICS LINK.

The CICS Adapter (Java version) communicates with the **Sun CICS Listener** for Synchronous Transactions as follows:

1. An incoming Connect request is handled by the IBM CICS Socket Listener, which starts the Sun CICS Listener Transaction and hands off the incoming connection via the IBM TCP/IP Give Socket and Take Socket interface.
2. The Sun CICS Listener allocates a CICS COMMAREA and copies information from the CICS Adapter COMMAREA to the actual CICS COMMAREA.
3. The Sun CICS Listener issues an EXEC CICS LINK to requested CICS Transaction Program passing it the newly allocated COMMAREA.
4. The requested CICS Transaction obtains data from the COMMAREA, performs typical business rule processing and then returns its results in the COMMAREA and returns control back to the Sun CICS Listener.
5. The Sun CICS Listener copies information from the CICS COMMAREA back to the CICS Adapter COMMAREA.
6. The Sun CICS Listener goes into a listen mode and waits for the next incoming Transaction Program request.

   The process continues until the Sun CICS Listener Timeout is exceeded or a disconnect request is received from the CICS Adapter.

z/OS CICS Security Considerations

Security Considerations for Sun CICS Listener

The CICS Adapter, using the Sun CICS Listener as the underlying connection transport, utilizes three modes of security with z/OS: Connection Logic, Request Link to Program, and Request Start Transaction. The userID and password are defined in the Adapter properties file. The connection manager uses the userID and password in the properties file to start the Sun CICS Listener on z/OS. During Business Rules processing, requests that flow into the Sun CICS Listener can use the userID and password from the properties file, or can be overwritten in the Collaborations.

Connection Logic

For the Connection Logic mode, the userID and password, passed from the CICS Adapter through the IBM CICS listener and into the Sun CICS Listener, must be defined for the z/OS security system (RACF, for example). The userID must be authorized by the z/OS security system to run CICS transaction "xxxx" inside of CICS. The default value for "xxxx" is STCL, and can be changed in the properties of the Connection Manager in the CICS Adapter.

Request Link to Program

For the Request Link to Program mode, the userID and password passed from the CICS Adapter to the Sun CICS Listener must be defined for the z/OS security system (RACF, for example). The userID must be authorized by the z/OS security system to run CICS program "prog1" inside of CICS. The default value for "prog1" is set in the properties of the CICS Adapter, and can be overridden in the Collaboration for each request sent into the Sun CICS Listener.

Request Start Transaction

For the Request Start Transaction mode, the userID and password passed from the CICS Adapter to the Sun CICS Listener must be defined for the z/OS security system (RACF, for example). The userID must be authorized by the z/OS security system to start CICS transaction "TRN1" inside of CICS. The default value for "TRN1" is set in the properties file of the CICS Adapter, and can be overridden in the Collaboration for each request sent into the Sun CICS Listener.

Security Considerations for IBM CICS Transaction Gateway

For information on CICS Transaction Gateway security validation refer to the following:

- IBM documentation CICS Transaction Gateway z/OS Administration or the CICS Transaction Gateway Administration Guide for your specific operating system, available at: `http://www-306.ibm.com/software/htp/cics/library/cicstgv5.html`

- Readme.txt for CTG 5.1 provided on the CTG 5.1 installation CD_ROM
- APAR OW55570 (for RACF)

About Sun Adapter for COM/DCOM

This topic provides conceptual information about COM/DCOM and its Sun Java CAPS Adapter.

About COM/DCOM

The Microsoft™ *Component Object Model* (COM) is a component software architecture that allows developers to partition an application into multiple components that can be developed and installed independently of each other. COM is the underlying architecture that forms the foundation for higher-level software services, like those provided by OLE (Object Linking and Embedding). OLE services span various aspects of component software, including compound documents, custom controls, inter-application scripting, data transfer, and other software interactions. Using COM allows software objects to be reused for a variety of applications. Because of its binary standard, COM allows any two components to communicate regardless of the language in which they were written.

The Microsoft *Distributed Component Object Model* (DCOM) is an extension of COM, and supports communication among objects residing on different computers; LANs, WANs, and the Internet. With DCOM, these software objects can be reused over a distributed Environment.

COM objects or components are individual modular software routines that can be reused within applications. COM objects are reusable compiled binary objects, as opposed to reusable sections of code. Creating an instance of a COM object provides a reference through which you can access the object's functionality.

The Sun COM/DCOM Adapter

The Sun COM/DCOM Adapter (referred to as the COM/DCOM Adapter through-out this document) allows the Sun Java Composite Application Platform Suite to create an instance of an automation-compatible COM object and access the methods and properties of that object.

One aspect of COM is "automation" based on the COM IDispatch interface. Objects that implement the IDispatch interface are known to be automation-compatible. Automation-compatible components are said to be "scriptable" and/or are capable of being "driven" by an automation client. This is possible because the IDispatch interface is well-known and those components that implement this interface adhere to a strict contract that is based on a

"late binding" concept. "Late binding" refers to a programming principle whereby the actual operation invoked is not determined until runtime. Objects that implement the IDispatch interface achieve this through the concept of the Invoke method on the IDispatch interface, which allows the user to call a method by name. The COM/DCOM Adapter is designed to work with automation-compatible components: that is, those that implement the IDispatch interface.

The COM/DCOM Adapter does not support Sun Java CAPS BPEL 1.0 Business Processes.

About Sun Adapter for e-Mail

The e-Mail Adapter enables the Sun Java CAPS ESB system to exchange data with an SMTP (outbound) or POP3 (inbound) mail server. The Adapter uploads messages to, and download messages from a mail server.

The e-Mail Adapter enables many typically manual e-mail operations to be automated. Functions are provided to log into a server, create e-Mails, and add recipients, subject headers, content, and attachments. Functions are also provided to read data associated with an incoming message and to save attachments.

Collaborations can be created to intelligently send e-mails with formatted content, and to receive, parse and act upon incoming messages. The SSL feature is supported through the use of Java Secure Socket Extension (JSSE) version 1.0.3.

The e-Mail Adapter's SSL (Secure Sockets Layer) feature provides secure communication channels for data exchanges, safe from unauthorized interception.

How does the e-Mail Adapter connect?

The e-Mail Adapter takes advantage of widely used standard protocol.

- POP3 (Post Office Protocol) which can be thought of as a store and forward service. POP3 is used for retrieving e-mail from a mail account. POP3 requires the mail server name, the TCP/IP Port, the e-mail account name, and the e-mail account password.
- SMTP (Simple Mail Transfer Protocol) is used for sending e-mail to an account. SMTP, like POP3 requires the mail server name, the TCP/IP Port, the e-mail account name, and the e-mail account password. Some mail servers also require POP3 authentication to send messages.
- MIME (Multi-purpose Internet Mail Extensions) protocol. Servers insert the MIME header at the beginning of any Web transmission. Clients use the header to select appropriate applications for the type of data the header indicates. Multipart MIME messages (alternate text/HTML) and multiple attachments (other than nested MIME objects) are supported.

The e-Mail Adapter provides a custom Object Type Definition OTD (MailClient) for managing e-mail content, and for sending and retrieving e-mail. The OTD allows dynamic configuration of the connection fields within a Collaboration (allowing properties to be changed "on the fly" from within an existing Collaboration).

Japanese e-Mail Text Support

The e-Mail Adapter supports Japanese character encoding in both the address and subject headers and text content of the e-mail message (both text/plain and text/HTML). The e-Mail Adapter conforms to RFC2047 standards for Multipurpose Internet Mail Extensions (MIME).

About Sun Adapter for File

The File Adapter supports sample Projects and to test your Sun Java™ Composite Application Platform Suite configuration. For a more feature-rich solution to processing flat file records, we recommend that you use the BatchAdapter. The Batch Adapter provides a number of pre- and post-processing features that the File Adapter does not offer, including system rename, copy, and delete operations that you can configure before and after you run a Collaboration, as well as other features such as regular expression matching for filenames.

The File Adapter exchanges data between an external file system and the Sun Java Composite Application Platform Suite. You can use this adapter for:

- Reading information from an external file to the Sun Java Composite Application Platform Suite.
- Writing information from the Sun Java Composite Application Platform Suite to an external file.

Adapter Operation

The File Adapter is provided as part of the Sun Java Composite Application Platform Suite and operates as follows:

- As an inbound adapter, the File Adapter polls an input directory for files based on a file name matching a specified regular expression. When the adapter detects a matching file, it opens the file and publishes the data to a Collaboration or Business Process Service.
- As an outbound adapter, the File Adapter writes processed data to an output file.

Setting Properties

You can set configuration properties for the File Adapter, including names and locations of source and destination files, in the File Adapter Properties dialog box.

About Sun Adapter for Batch

All Adapters provide a communication bridge between the eGate environment and one or more external systems.

The Batch Adapter performs a variety of FTP and FTP-related operations (depending on your specific needs, network environment, record-processing, file transfer, and external system requirements). The Batch Adapter enables Sun Java CAPS ESB to use an FTP connection to exchange data with other network hosts for the purpose of receiving and delivering objects stored in files.

Batch Adapter OTDs

The Batch Adapter provides Object Type Definitions (OTDs) that enable the creation of file transfer operations using FTP.

The Batch Adapter includes seven specific OTDs:

- **BatchFTP**: The FTP OTD connects to external FTP servers.
- **BatchFTPOverSSL**: The FTP over SSL OTD provides secure data transfer using Secure Sockets Layer (SSL) protocol.
- **BatchSCP**: The SCP OTD provides secure data transfer using Secure Copy Protocol with Secure Shell (SSH) as an underlying protocol.
- **BatchSFTP**: The SFTP OTD provides secure data transfer using SSH File Transfer Protocol (SFTP protocol). SFTP protocol provides a range of operations on remote files, such as directory listings, and remote file removal.
- **BatchLocalFile**: The local file OTD picks up or puts data files to local file systems.
- **BatchRecord**: The record-processing OTD extracts records out of files, parses files into specific records, and defines the content of files as records.
- **BatchInbound**: The inbound OTD receives a file, renames the file with GUID file name, and triggers the Business Process or Collaboration.

Note – The Batch Adapter supports standard FTP in accordance with RFC-959.

About Sun Adapter for IMS

This topic provides conceptual information about IMS and its Sun Java CAPS Adapter.

About Information Management System (IMS)

IBM's IMS, is a database and transaction management system that provides an interface for users to access information in various databases via on-line transactions. The IMS/TM (Transaction Manager) is a message-based transaction processor, that handles the execution of specific business application programs. The IMS/DB (Database) is an entirely separate component providing access to the IMS hierarchical database for applications running under the IMS/TM, as well as IMS transaction monitor and OS/390 batch jobs.

About the IMS eWay

The IMS Adapter enables Sun Java CAPS ESB to connect with IBM's IMS/TM mainframe applications through IBM's IMS Connect.

The IMS Adapter provides access to the Input and Output Descriptors (MID/MOD) of the IMS applications without requiring changes to the application. By capturing the field contents before screen formatting, the IMS Adapter is not affected by cosmetic changes to the application's screen design.

The IMS Adapter includes the IMS Message Format Service (MFS) Wizard conversion utility to facilitate the creation of input and output Object Type Definitions (OTDs) from IMS MFS files.

The implementation of the IMS Adapter is in accordance with IBM's *IMS Connect Guide and Reference*. These documents describe the OTMA protocol and contain important prerequisite information for the configuration IMS Connect on the mainframe.

A sample project for the IMS Adapter is included on the installation CD-ROM which demonstrates how a non-conversational scenario (simple send/response) is managed.

About Sun Adapter for LDAP

This topic provides conceptual information about LDAP and its Sun Java CAPS Adapter.

About LDAP

LDAP (Lightweight Directory Access Protocol), is an Internet protocol for accessing information directories. A directory service is a distributed database application designed to manage the entries and attributes in a directory. LDAP runs over TCP/IP.

LDAP allows clients to access different directory services based on entries. It makes the entries, along with their attributes and values, available to users and other applications, on a controlled-access basis.

The LDAP OTD provides access to the operations available via the LDAP protocol. To give you a better understanding of these operations and how they are implemented in the OTD, this section briefly summarizes how LDAP works.

Entries, Attributes, and Values

An LDAP directory has entries that contain information pertaining to some entity. Each of the entry's attributes has a name and one or more values. The names of attributes are most often mnemonic strings, such as **cn** for common name, or **mail** for e-mail address.

For example, a company may have an employee directory. Each entry in the employee directory represents an employee. The employee entry contains such information as the name, e-mail address, and phone number, as shown in the following example:

```
cn: John Doe
mail: johndoe@sun.com
mail: jdoe@stc.com
telephoneNumber: 471-6000 x.1234
```

Each part of the descriptive information, such as an employee's name, is known as an attribute. In the example above, the Common Name (**cn**) attribute, represents the name of the employee. The other attributes are **mail** and **telephoneNumber**.

Each attribute can have one or more values. For example, an employee entry may contain a mail attribute whose values are **johndoe@sun.com** and **jdoe@stc.com**. In the previous example, the mail attribute contains two mail values.

LDAP Directory Structure

The organization of a directory is a tree structure. The topmost entry in a directory is known as the root entry. This entry normally represents the organization that owns the directory.

Entries at the higher level of hierarchy, represent larger groupings or organizations. Entries under the larger organizations represent smaller organizations that make up the larger ones. The leaf nodes (or entries) of the tree structure represent the individual persons or resources.

Distinguished Names and Relative Distinguished Names

An entry is made up of a collection of attributes that have a unique identifier called a distinguished name (DN). A DN consists of a name that uniquely identifies the entry at that hierarchical level. In the example above, John Doe and Jane Doe are different common names (**cn**) that identify different entries at that same level.

A DN is also a fully qualified path of names that trace the entry back to the root of the tree. For example, the distinguished name of the John Doe entry is:

```
cn=John Doe, ou=People, dc=sun.com
```

A relative distinguished name (RDN) is a component of the distinguished name. For example, **cn=John Doe, ou=People** is a RDN relative to the root RDN **dc=sun.com**. DNs are used to describe the fully qualified path to an entry while an RDN is used to describe the partial path to the entry relative to another entry in the tree.

Wherever necessary, the LDAP OTD mimics this same directory structure.

LDAP Service and LDAP Client

A directory service is a distributed database application designed to manage the entries and attributes in a directory. A directory service also makes the entries and attributes available to users and other applications. OpenLDAP server is an example of a directory service. Other directory services include Sun Active Directory Service (Sun Microsystems) and Microsoft Active Directory.

A directory client accesses a directory service using the LDAP protocol. A directory client may use one of several client APIs available in order to access the directory service.

Referrals

The native APIs developed for the LDAP Adapter query the results of a search based on specified criteria. The search results may consist of a number of referrals.

A referral is an entity that is used to redirect a client's request to another server. A referral contains the names and locations of other objects. For example, an LDAP server sends a referral to the client to indicate that the information that the client has requested can be found at another location (or locations), possibly at another server or several servers.

The referral contains the URL of the LDAP server that holds the actual entry. The LDAP URL contains the server's host/port and an object's DN.

About the LDAP Adapter

This section describes the general information about the LDAP Adapter and its operation with Sun Java CAPS ESB.

Adapter General Operation

The LDAP Adapter enables Sun Java CAPS ESB to exchange data with an LDAP directory on an LDAP server. The Adapter consists of two components, an LDAP connector and an LDAP Object Type Definition (OTD). The OTD utilizes the connector to connect to a particular LDAP server.

By connecting to an LDAP server, the Adapter enables Sun Java CAPS ESB to search, compare, and modify an LDAP directory using the LDAP protocol. The Adapter utilizes the LDAP OTD to perform these functions. This OTD carries LDAP information through Sun Java CAPS ESB and allows the information to be processed by Sun Java CAPS ESB's Java-based Collaborations.

In addition, the LDAP OTD exposes the application programming interface (API) for accessing the LDAP directory. The LDAP OTD enables you to create Java-based Collaboration Definitions that execute LDAP operations, for example, searching an LDAP directory, adding entries to the directory, and modifying entries in the directory.

A given instance of an LDAP OTD uses only one instance of an LDAP connector. You can use as many instances of the LDAP OTD in a single data-exchange scenario, as necessary.

Java Naming and Directory Interface

The LDAP Adapter uses Sun Microsystem's Java Naming and Directory Interface (JNDI) LDAP provider. This set of APIs allows a Java program to store objects and look up objects using multiple naming services in a standard manner.

The JNDI is included in the Java 2 Software Developer's Kit (SDK) version 1.4 installed as part of Sun Java CAPS ESB.

Third-Party License File Agreement

A disclaimer readme file is available for review when you install the LDAP Adapter. The disclaimer is applicable to the jCookie Library, a robust and easy to use library for client-side HTTP state management in Java applications.

After successful installation, you can view the following third-party file using any text file viewer:

LDAPeWay_THIRDPARTYLICENSEREADME.txt

Third-party license files are located at:

```
\repository\ThirdPartyLicenses
```

where repository indicates the folder where the Sun Java CAPS ESB Repository is installed.

About Sun Adapter for MSMQ

This topic provides conceptual information about MSMQ and its Sun Java CAPS Adapter.

About Microsoft Message Queuing (MSMQ)

Microsoft Message Queuing, or MSMQ, is message queuing communications "middleware" that enables applications on diverse systems to communicate with each other. MSMQ allows applications to communicate through messages, and can guarantee the delivery of messages even in the event of system or network failures.

By using messages, MSMQ can provide asynchronous communication between applications running on the same computer or on separate computers connected through a network. When an application receives a message, it processes the message by reading the message content and acting accordingly. If appropriate, the receiving application can send a response message back to the sending application. MSMQ protects these transactions by safely storing the messages in queues, which prevents message loss and make messages available to an application whenever it is prepared to receive them.

Applications using MSMQ are able to send and process messages, even when the receiving application is not available. Upon connection, the receiver can accept or pick up the messages. Applications using MSMQ's transactional delivery mode can be assured that messages are delivered exactly one time only, and that they are delivered in the order that they were sent.

About the MSMQ Adapter

The Sun Adapter for MSMQ allows the Sun Java Composite Application Platform Suite to exchange data with Microsoft's MSMQ. The MSMQ Adapter transparently integrates existing systems with MSMQ.

The MSMQ Adapter enables the Sun Java Composite Application Platform Suite to use business logic within Collaborations or Business Processes to perform data identification, manipulation, and transformation. Messages are tailored to meet the communication requirements of specific applications or protocols.

About Sun Adapter for SNA

This topic provides conceptual information about SNA and its Sun Java CAPS Adapter.

About SNA

SNA is a data communications architecture developed by IBM to specify common conventions for communication between various IBM hardware and software products. It is specifically designed to address issues of reliability and flexibility of sharing data between components and their peripherals. Many vendors other than IBM also support SNA, allowing their products to interact with SNA networks.

An addressable unit on an SNA network is called a node, and is made up of four functional components forming a hierarchy.

To establish a communications session, SNA uses Logical Units (LUs) as entry points into the network. There are several types of LUs, currently type 0 through type 6.2. Most of the LU types are specific to IBM operating environments, but type 6 is intended for use in a distributed data processing environment.

Generally, an LU can communicate only with another LU of the same type, but specific exceptions to this rule exist with type 6.2. LU6.2 is the least-restrictive of the various LU types, and also supports multiple concurrent sessions. As a result, it is the LU most widely supported by other system vendors.

Like the OSI model, SNA functions are divided into seven hierarchical layers, but the layers are not identical. The Transport Network handles the lower three layers, while the Network Accessible Units (NAU) implement the upper four layers by using the services of the Transport Network to establish communication between nodes.

SNA defines formats and protocols between these layers that allow equivalent layers in different nodes to communicate with each other. Also, each layer provides services to the layer above, and requests services from the layer below.

SNA uses a standard method for the exchange of data within a network. This standard method defines how to establish a route between components, how to send and receive data reliably, how to recover from errors, and how to prevent flow problems.

Originally designed for networks in which a mainframe computer controls the communications relationships, SNA has since evolved to incorporate protocols and implementations to allow two user processes to communicate with each other directly. These two different networking models, or roles, are referred to as hierarchical and peer-oriented, respectively. The peer-oriented model is designed to allow distributed control of the communications process independent of the mainframe.

The peer-to-peer connection between two user processes is known as a *conversation*, while the peer-to-peer connection between two LUs is known as a *session*. A session is generally a long-term connection between two LUs, while a conversation is generally of shorter duration.

A *User Process* is also known as a *Transaction Program* (TP). Also, the interface between a User Process and an LU is known as *Presentation Services*.

Supported Logical Unit Types

SNA LU6.2

LU 6.2, also known as APPC (Advanced Program-to-Program Communication), is used for Transaction Programs communicating with each other in a distributed data processing environment. In a CPIC (Common Programming Interface for Communications) implementation, CPIC provides the API that contains the commands, known as verbs, that are used by LU 6.2 to establish communication sessions.

Two types of Presentation Service interfaces are possible with LU6.2: mapped conversations and unmapped, or basic, conversations. The following table summarizes the set of LU6.2 commands for basic conversations. Equivalent commands for mapped conversations have the prefix <MC_> added to the command name. Note that “control operator verbs” are not listed.

TABLE 1 LU6.2 Commands

Name	Description
ALLOCATE	Allocates a conversation with another program.
CONFIRM	Sends a confirmation request to the remote process and waits for a reply.
CONFIRMED	Sends a confirmation reply to the remote process.
DEALLOCATE	De-allocates a conversation.
FLUSH	Forces the transmission of the local SEND buffer to the other LU.
GET_ATTRIBUTES	Obtains information about a conversation.
PREPARE_TO_RECEIVE	Changes the conversation state from SEND to RECEIVE.
RECEIVE_AND_WAIT	Waits for information (either data or confirmation request) to be received from the partner process.
RECEIVE_IMMEDIATE	Receives any information that is available in the local LU’s buffer, but does not wait for information to arrive.
REQUEST_TO_SEND	Notifies the partner process that the local process wants to send data. When a “send” indication is received from the partner process, the conversation state changes.

TABLE 1 LU6.2 Commands *(Continued)*

Name	Description
SEND_DATA	Sends one data record to the partner process.
SEND_ERROR	Informs the partner process that the local process has detected an application error.

About the SNA Adapter

The SNA Adapter enables the Sun Java CAPS ESB system to access an SNA network environment to drive entire transactions, including conversational transactions.

The SNA Adapter is an interface that makes calls to an SNA Server. The SNA Server acts as a high-speed gateway between distributed SNA Clients and the SNA network having a mainframe host system.

In a typical data exchange using the SNA Adapter, the Adapter invokes the LU6.2 protocol--through the invocation of CPI-C calls--to enable the SNA client to send requests to the SNA server. For outbound Adapters, the Adapter can be triggered by any incoming message. For inbound Adapters, the Adapter is triggered by established conversation activity.

About Sun Adapter for TCP/IP

The Sun Adapter for TCP/IP (referred to as the TCP/IP Adapter throughout this document) enables the Sun Java CAPS ESB to communicate with client applications using TCP/IP, and provides real-time, reliable data transfer for systems that support TCP/IP.

For details on operating and using Sun Java CAPS ESB and its user interface, see the Java CAPS topics for ESB.

The TCP/IP Adapter allows you to create a client interface to a server or implement a TCP/IP server in eGate, using a Collaboration framework created using the eGate Enterprise Designer. This interface allows your system to communicate via TCP/IP.

The TCP/IP Object Type Definition (OTD) enables the creation of any messaging protocol capable of running over TCP/IP, and also utilizes the common Adapter services available in ESB.

The Adapter also allows you to select a desired message envelope, using predefined envelope types. If these types do not meet your needs, you can customize your own envelope, using specialized Adapter interfaces designed for this purpose.

About Sun Adapter for TCP/IP HL7

The Sun Adapter for TCP/IP HL7 enables the Sun Java CAPS ESB system to exchange data with an external TCP/IP application, using the HL7 data protocol. The Sun Java CAPS ESB with the TCP/IP HL7 Adapter provides:

- Macro functionality, providing ease of use and productivity.
- Prebuilt Standards-compliant Inbound and Outbound Template Collaborations that work "as is" or can be modified for your specific needs.
- Customization of functionality, by configuring the Adapter properties and/or modifying the Collaborations.
- Journaling and error messaging to JMS queues and topics. This is in addition to Sun Java CAPS ESB's standard alert and debug logging.
- Support for HL7 Standard versions 2.1, 2.2, 2.3, 2.3.1, 2.4, and 2.5.

Note – Throughout this document the term "JMS queue" is used in the generic sense and actually denotes JMS queues or topics.

TCP/IP HL7 Features

The TCP/IP HL7 Adapter includes the following features:

- Bi-directional, including Client or Server mode in either direction (to or from Sun Java CAPS ESB).
- Handles both HL7 HLLP and MLLP protocols and envelopes.
- Provides a wide variety of recourse action configurations.
- Non-blocking I/O.
- Recovery and retry logic.
- Debug levels.
- Error logging.
- Journaling of HL7 messages and associated ACKs
- HL7 Acknowledgement levels
- Fully supports the HL7 sequence numbering protocol.
- Full support for HL7 ACK, NAK generation and validation.
- Supports Delayed ACK in both directions.

TCP/IP HL7 Adapter Components

The TCP/IP HL7 Adapter incorporates three components:

- The HL7 TCP/IP Resource Adapter that implements the lower layer HL7 protocol over TCP/IP.
- Default inbound and outbound Collaborations that implement the HL7 messaging protocol, sequence numbering and recourse actions.
- Generic HL7 OTDs that provide the structures necessary to parse and create the data messages and ACKs used by the protocol.

The TCP/IP HL7 Object Type Definition (OTD) enables the creation of HL7 interfaces capable of running over TCP/IP, and also utilizes the common Adapter services available in Sun Java CAPS ESB. The TCP/IP HL7 Adapter works hand in hand with the Sun Java CAPS HL7 OTD Libraries, versions 2.1 through 2.5.

The TCP/IP HL7 Adapter properties allow the user to easily configure the operation of the TCP/IP HL7 Adapter. These properties are adopted into the OTD's functions.

The OTD handles all of the lower-layer protocol. The OTD's behavior is customized using the Adapter configuration properties. These Adapter properties are used by the resource adapter, but are also accessed and used by the prebuilt Collaborations.

About Sun Adapter for HTTPS

This topic provides conceptual information about HTTPS and its Sun Java CAPS Adapter.

About HTTP and HTTPS

HTTP (hypertext transfer protocol) is the set of rules used for transferring files (text, graphic images, sound, and video) over the Web. When a user opens a Web browser, the user is indirectly making use of HTTP. HTTP is an application protocol that runs on top of the TCP/IP suite of protocols.

In addition to the files that it serves, every Web server contains an HTTP daemon—a program that waits for HTTP requests and handles them when they arrive. A Web browser is an HTTP *client*, sending requests to *server* machines. When the user enters a URL or clicks on a hypertext link, the browser builds an HTTP request and sends it to the IP address indicated by the URL. The HTTP daemon in the destination server machine receives the request and sends back the requested file or files associated with the request.

HTTPS (hypertext transfer protocol over secure socket layer—or HTTP over SSL) is a Web protocol that encrypts and decrypts user page requests as well as the pages that are returned by

the Web server. HTTP uses port 443 instead of HTTP port 80 in its interactions with the lower layer TCP/IP. SSL uses a 40-bit encryption key algorithm, which is considered an adequate level of encryption for commercial exchange.

When an HTTPS request is sent by a browser—usually by clicking a link that begins with **https://**—the client browser encrypts the request and sends it to the Web server. The acknowledgement sent by the Web server is also sent using encryption, and is decrypted by the client browser.

About the HTTPS Adapter

The HTTPS Adapter enables Sun Java CAPS ESB to communicate with client and server applications over the Internet using HTTP, either with or without SSL.

HTTP Messages

An HTTP message has two parts: a request and a response. The message header is composed of a header line, header fields, a blank line, and an optional body (or data payload). The response is made up of a header line, header fields, a blank line, and an optional body (or data payload). HTTP is a synchronous protocol, that is, a client makes a request to a server and the server returns the response on the same socket.

Web Browser Cookies

A cookie is an HTTP header, which is a key-value pair in the header fields section of an HTTP message.

The **Set-Cookie** and **Cookie** headers are used with cookies. The **Cookie-request** header is sent from the server in request for cookies on the client side. An example of a **Cookie-request** header is:

```
Set-Cookie: sessauth=44c46a10; expires=Wednesday, 27-Sep-2006
03:59:59 GMT
```

In this example, the server requests that the client store the following cookie:

```
sessauth=44c46a10
```

Everything after the first semi-colon contains additional information about the cookie, such as the expiration date. When the Adapter sees this header, it extracts the cookie `sessauth=44c46a10` and returns it to the server on subsequent requests. The Adapter prepends a cookie header to the HTTP request, for example:

```
Cookie: sessauth=44c46a10
```

Each time the Adapter sends a request to the same server during a session, the cookie is sent along with the request.

Cookie Expiration Date Checking

The HTTPS Adapter checks time-limited cookies with expiration dates to ensure that they have not expired. If they have expired, the cookie is removed and is not resent to the originating server. As a result, the session state is removed.

The following standard expiration date formats are recognized by the HTTPS Adapter:

```
"Sun, 06 Nov 1994 08:49:37 GMT"            ;RFC 822, updated by RFC 1123
"Sunday, 06-Nov-94 08:49:37 GMT"         ;RFC 850, obsoleted by RFC 1036
"Sunday, 06-Nov-1994 08:49:37 GMT"                 ;RFC 1036
"Sun Nov  6 08:49:37 1994"                   ;ANSI C's asctime()
```

If the expiration date is in another format, the Adapter does not recognize the expiration date. Instead, it treats the cookie as if it does not have an expiration date.

GET and POST Methods

The GET method can be used in client mode to retrieve a page specified by the URL or to retrieve information from a form-based Web page by submitting URL-encoded key and name value pairs. In the latter case, the page must support the GET method.

The following example shows a URL-encoded query string:

```
http://.../bin/query?p=seebeyond+integrator
```

The URL specifies the search page and the name-value pair for the search. The question mark (?) indicates the beginning of the name-value pair encoding. In the previous example, the name portion of the query is "p," and the value to search is "seebeyond integrator." A query can consist of one or more of these name-value pairs.

Note – See the official HTTP Specification for complete information.

The POST method is more versatile, in that it supports form-based requests, as well as sending large amounts of data. The POST method does not have the size-limitation maximum of 255 or 1024 characters (depending on the Web server), which the GET method has. As with GET, the Web page must support the POST method in order to use POST.

Taking the previous URL as an example, if you specify the following URL:

```
http://.../bin/query
```

Then, you can specify the name-value pair separately. The HTTP client allows for the specification of the URL and n-number of value pairs via its methods.

Sample HTTP Exchange in Client Mode

To retrieve the file at the following URL:

```
http://www.myhost.com/path/file.html
```

First open a socket to the host `www.myhost.com`, port 80 (use the default port of 80 because none is specified in the URL). You can then send a request through a socket that looks like the following example:

```
GET /path/file.html HTTP/1.0                (Request Header Line)
User-Agent: HTTP(S)Adapter                  (Request Header field)
```

The server sends a response back through the same socket. The response could look like the following example:

```
HTTP/1.0 200 OK               (Response Header Line)
Date: Fri, 31 Dec 1999 23:59:59 GMT                (Response Header Field)
Content-Type: text/html             (Response Header Field)
Content-Length: 1354                (Response Header Field)
[blank line here]
<html>                  (Response payload)
<body>
<h1>Happy New Millennium!</h1>
(more file contents)
  .
  .
  .
</body>
</html>
```

After sending the response, the server closes the socket.

Sample HTTP Exchange in Server Mode

To listen for a request from an HTTP client, the HTTPS Adapter in server mode listens on the port configured for your Integration Server (18001 by default). The HTTPS Adapter receives the request and processes it according to the logic you create in your Collaboration or Business Process.

In a simple example, the HTTPS Adapter receives a request from the following form:

```
<HTML><HEAD><TITLE>HTTP Server JCE Test Page</TITLE></HEAD>
<BODY>
<FORM ACTION="http://localhost:18001/
    Deployment1_servlet__MyServlet/
    _MyServlet" METHOD=POST>
<TABLE>
```

```
<TR><TD>What's your name?</TD><TD><INPUT NAME=fname></TD></TR>
<TR><TD></TD><TD></TD></TR>
</TABLE>
<BR>
<CENTER><INPUT TYPE=submit VALUE="Submit"></CENTER>
</FORM>
</BODY>
</HTML>
```

When the client enters a name in a browser and clicks Submit, the HTTPS Adapter server returns a simple response (according to the logic in the Collaboration or Business Process).