SYBASE®

Reference

Application Integrator for CICS

Version 3.0

Document ID: 33123-01-0300-01

Last revised: February 1999

Sybase, Inc., 6475 Christie Avenue, Emeryville, CA 94608.

Contents

About This Book

Application Integrator is a Jaguar component development and deployment tool that allows application developers to quickly integrate legacy applications and transactions with Jaguar CTS.

This preface contains the following topics:

- Audience
- How to Use This Book
- Documentation
- Conventions
- If You Need Help
- If You Have Questions About This Book

Note The remaining chapters in this guide refer to Application Integrator as “AI for CICS,” and to Jaguar Component Transaction Server (CTS) as “Jaguar.”

Audience

Use this document if you are responsible for using AI for CICS to create components from COBOL COMMAREAs and then deploy them into Jaguar.

How to Use This Book

The following table describes the contents of this book.

Table 1: Application Integrator for CICS Reference contents

Chapter	Contents
Chapter 1, “Understanding Application Integrator for CICS”	Provides an overview of the functionality and features of Application Integrator.
Chapter 2, “Getting Ready to Use AI for CICS”	Describes the COMMAREA or data definition file that Application Integrator will use to create a component.
Chapter 3, “Working with the AI Component Builder”	Describes how to work with the AI for CICS Component Builder to create and deploy components.
Chapter 4, “Jaguar and Application Issues”	Describes guidelines for using Application Integrator components in Jaguar CTS.
Glossary	Defines technical terms used in this book.

Documentation

Sybase provides the following AI for CICS documentation:

- *Application Integrator for CICS Reference*
- Application Integrator for CICS online help

See the online documentation in the Sybase\Application Integrator 3.5 program group for Application Integrator samples, which include tutorials designed to give you hands-on experience with each step of creating and deploying components.

Related Documentation

This section lists documentation for the following products that can be used with AI for CICS:

- EAStudio
- EAServer
- Jaguar CTS
- Open ServerConnect
- DirectConnect for MVS

EAStudio

Building Internet and Enterprise Applications provides information about using Application Integrator and other EAStudio applications.

The *Enterprise Application Studio Installation Guide* describes how to install Application Integrator on the workstation and the mainframe.

EAServer

The *Enterprise Application Server version 3.0 Feature Guide* provides information about using Application Integrator and other EAServer applications.

Jaguar CTS

The *Jaguar CTS Programmer's Guide* is available with Jaguar CTS version 3.0.

Open ServerConnect

For LAN-to-mainframe communications *without* gateway software, you will use an Open ServerConnect application that is included with AI for CICS. The following relevant documentation is available with Open ServerConnect version 4.0:

- *Open ServerConnect Installation and Administration Guide for IBM CICS/MVS*
- *Open ClientConnect and Open ServerConnect Messages and Codes*
- Release Bulletin Open ServerConnect version 4.0 for IBM CICS/MVS

DirectConnect for MVS

For LAN-to-mainframe communications using gateway software, you must use DirectConnect for MVS. The following relevant documentation is available with DirectConnect for MVS version 11.07:

- *DirectConnect Server Administration Guide*
- *DirectConnect for MVS Connectivity Guide*
- *DirectConnect for MVS Installation Guide*
- *DirectConnect Transaction Router Service User's Guide*

- *MainframeConnect for DB2/MVS-CICS Installation and Administration Guide*
- *Release Bulletin for DirectConnect for MVS version 11.07 for AIX, HP-UX, Solaris, and Windows NT*

Conventions

This section describes the following:

- Style Conventions
- Syntax Conventions

Style Conventions

The following table shows some of the style conventions used in the documentation for this product.

Table 2: Style conventions

Item	Example
• Programs • Utilities • Procedures • Commands	create connection
• File names • Directory names • Properties	*install.cfg*
• Code examples • Screen text	01 DFHCOMMAREA
• User input • Command line input	*shutdown*
Variables (text that you replace with the appropriate value)	*host_name*

Syntax Conventions

The following example illustrates some of the syntax conventions used in this guide:

```
COMMAND [object_name, [ {TRUE | FALSE} ] ]
```

The following table explains the syntax conventions used in this guide.

Table 3: Syntax conventions

Symbol	Explanation	Example
( )	Include parentheses as part of the command.	`START DATABASE (`*`database_name`*`)`
|	A vertical bar indicates that you can select only one of the options shown. Do not type the bar in your command.	`{yes|no}`
,	A comma indicates that you can choose one or more of the options shown. Separate each choice by using a comma as part of the command.	`{long,short,none}`

{ }	Braces indicate that you must choose one or more of the enclosed options. Do not type the braces when you enter the option.	*Select only one:*
		`{datetime|datetime4}`
		Select at least one:
		`{char_iso,char_eur, char_jis}`

[ ]	Brackets indicate that you can choose one or more of the enclosed options, or none. Do not type the brackets when you enter the options.	`[truncate]`
...	An ellipsis indicates that you can repeat the previous item as many times as necessary.	`{datax,datay}...`

If You Need Help

Each Sybase installation that has purchased a support contract has one or more designated people who are authorized to contact Sybase Technical Support. If you cannot resolve a problem using the manuals or online help, please have the designated person contact Sybase Technical Support or the Sybase subsidiary in your area.

Sybase Professional Services

Sybase Professional Services offers on-site consulting and training programs to help you maximize the benefits of our products. For more information, call (800) 8SYBASE.

Note Outside the continental United States and Canada, obtain the correct telephone numbers from your sales representative and record them for future reference.

If You Have Questions About This Book

If you have questions, comments, or suggestions about this book, contact the Sybase documentation group directly by e-mail at:

```
icd_doc@sybase.com
```

Feel free to forward any information, comments, or questions about the following:

- Missing, incorrect, or unclear information
- Information you found particularly useful
- Organization or style

We will respond as promptly as possible by e-mail. Your feedback helps us provide more accurate, detailed, and easy-to-use documentation.

Note Please send comments about product features, functionality, or problems to your systems engineer or Sybase Technical Support.

CHAPTER 1

Understanding Application Integrator

This chapter describes the functionality and architecture of Application Integrator, and includes the following topics:

- What Is AI for CICS?
- Why Use AI for CICS?
- Jaguar and AI for CICS Functionality
- Basic Steps to Building a Web Application
- Using the AI for CICS Tutorial

Note The remaining chapters in this guide refer to Application Integrator as “AI for CICS” and to Jaguar Component Transaction Server (CTS) as “Jaguar.”

What Is AI for CICS?

AI for CICS is a Jaguar component development and deployment tool that allows application developers to quickly integrate legacy applications and transactions with Jaguar.

AI for CICS is included in Sybase EAServer, an integrated set of application servers you can use to deploy Web applications that support high-volume traffic, dynamic content, and intensive online transaction processing. EAServer consists of PowerDynamo, Jaguar, AI for CICS, AI for Stored Procedures, and Adaptive Server Anywhere (ASA).

This section covers the following topics:

- A Description of AI for CICS
- AI for CICS Architecture

A Description of AI for CICS

Using its Component Builder design tool, AI for CICS allows you to create components that are proxies (front-ends) for CICS programs written in COBOL, making them accessible to Internet or Intranet clients. These components, which represent the COBOL programs, are deployed in Jaguar. The end result is a LAN- or Web-based application that gives the user full access to existing CICS programs that can read and/or write mainframe data.

AI for CICS exposes mainframe business logic through two types of objects:

- A component: a self-contained, reusable, piece of software that represents business logic of the CICS program
- A connection: a specification of a path that a component uses to contact its source program

AI for CICS can run in a gateway-less environment or in a gateway environment using DirectConnect for MVS and DirectConnect Transaction Router Service (TRS).

AI for CICS Architecture

AI for CICS architecture consists of the following:

- The Component Builder, the design-time Graphical User Interface (GUI) portion of AI for CICS, which allows the user to perform these tasks:
  - Import data definition information that defines the component
  - Create a connection and a component
  - Deploy the component to produce Java source code
- A library of Java classes that components use to access the mainframe
- The AI for CICS mainframe run time that invokes COBOL programs on behalf of AI for CICS components

Figure 1-1 shows the design-time architecture of AI for CICS.

Figure 1-1: AI for CICS design-time architecture

During design time, the Component Builder imports the COMMAREA from the CICS program (in this example, from a copybook), then deploys the component with its definitions into Jaguar, where it will be used by the client application.

Figure 1-2 shows the run-time architecture of AI for CICS.

Figure 1-2: AI for CICS run-time architecture

During run time, a client application attempts to invoke a component in Jaguar, which in turn accesses the CICS program on the mainframe. Note that if you are using SNA connectivity protocol, you need to provide connectivity from the Jaguar machine through DirectConnect TRS to CICS; this is the gateway option for connectivity.

Why Use AI for CICS?

To understand how AI for CICS can provide solutions for business problems, consider the following scenario.

Problem

A company needs to attract new prospects and keep them as customers, and at the same time, lower the cost of sales.

The company wants to allow customers to self-service their accounts through a new application over the Internet. Therefore, the customer can easily renew an account, and by deploying the application over the Internet, the company can expand its target market without increasing its direct sales force.

However, the company faces the following constraints:

- Existing operational systems were not designed for direct exposure to customers over the Internet.
- The company cannot afford to rewrite all the applications or consolidate all the data on a common platform. The business logic and security infrastructure is spread across LAN database stored procedures, mainframe CICS applications, and C++ DLLs.
- The new application must be developed using HTML and Java (for broad Internet accessibility), but none of the existing systems are “Web-enabled” today.

Solution

For its solution, the company uses Sybase Enterprise Application Server (EAServer) as a platform to integrate their internal applications and make them accessible through the Internet. Using AI for CICS and AI for Stored Procedures, they can access business logic in their mainframe applications and database stored procedures quickly and easily from the Jaguar middle tier for new application development.

Jaguar and AI for CICS Functionality

The following sections describe how Jaguar functions and how it relates to AI for CICS:

- How Jaguar Works
- When Jaguar Uses AI for CICS Components

How Jaguar Works

Jaguar implements a “multi-tier” distributed computing architecture. Figure 1-3 shows the typical Jaguar environment, in which three distinct elements, or tiers, work together to allow users to access data:

- The client, which contains the applet or application code that manages the presentation and interaction with the end user.
- Middle-tier components, which run in Jaguar and are typically coded by the middle-tier developer. They contain the executable business logic that handles much of the application processing.
- The back-end database, which is the transaction target accessed by the Jaguar component. It stores, manages, and processes data.

Figure 1-3: Typical Jaguar environment

In most cases, interaction between the client application and Jaguar occurs as follows:

1 When a client application needs to use a component, it issues a request to the Jaguar server.

2 The request from the client environment prompts Jaguar to instantiate the component.

3 The client application invokes a method on a component. The method executes, accesses a database, and possibly returns data to the client.

Note For detailed information about how Jaguar works, see the Jaguar documentation.

When Jaguar Uses AI for CICS Components

When Jaguar uses AI for CICS components, the components in the middle tier are “proxies” that represent the CICS application program. The Jaguar environment is composed of the following elements, or tiers:

- The client, which contains the applet or application code that manages the presentation and interaction with the end user.
- Middle-tier AI for CICS components, which provide direct access to the CICS application program. No middle-tier coding is required.
- The CICS program, which contains business logic for an application.

Figure 1-4 shows the Jaguar client communicating with a CICS program through the AI for CICS component in the middle tier.

Figure 1-4: Jaguar environment with AI for CICS

In most cases, interaction between the client application and Jaguar occurs as follows:

1 The Jaguar client application requests the component to be instantiated one of two ways.
   - If the Jaguar client is an applet, users can find and launch applications from traditional HTML pages. Instead of simply loading a static page, Jaguar downloads an executable applet to the user's browser.
   - If the Jaguar client is an installed application, users can launch the application from their machines.

2 Jaguar instantiates the component.

3 The client calls a method in the component.

4 The AI for CICS component invokes the CICS program and returns any result sets to the client application.

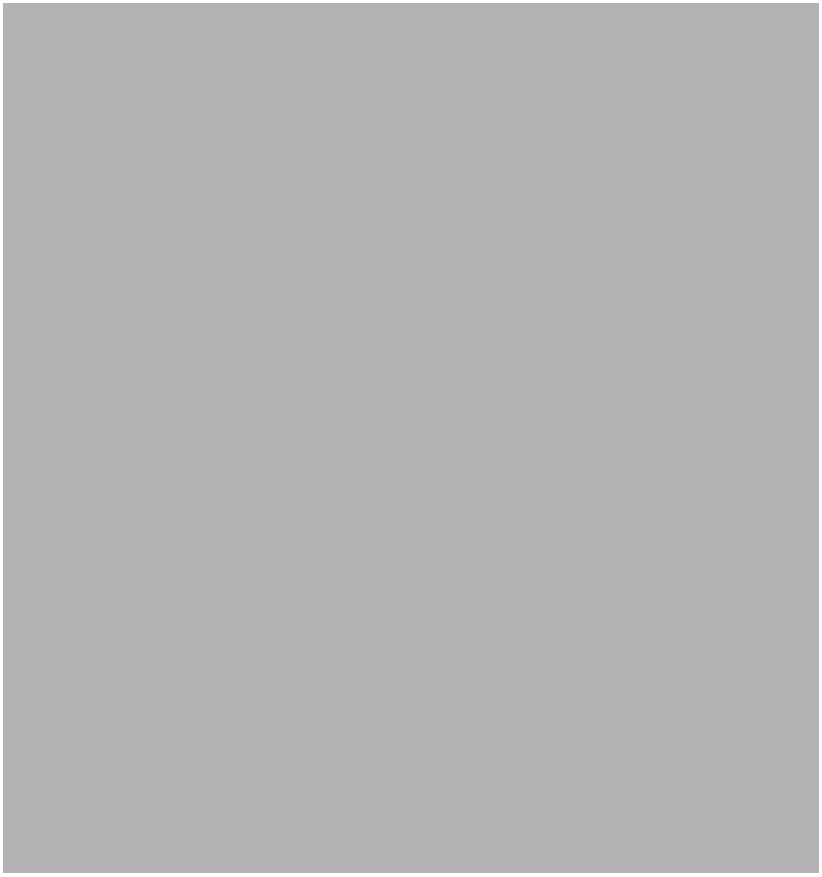
Basic Steps to Building a Web Application

Following are the basic steps required to build a Web- or Internet-based application:

- Step 1: Prepare the COMMAREA Definition
- Step 2: Create the Component
- Step 3: Deploy the Component
- Step 4: Use the Component

Figure 1-5 shows the process flow by task, the environment in which each task is performed, and the skills needed to perform the tasks in each step.

Figure 1-5: Four-step process flow and required skills



The following sections describe each step.

Step 1: Prepare the COMMAREA Definition

You must research the CICS program you plan to use so you are familiar with how it is defined and how it works.

The COMMAREA contains data definitions of input and output for CICS programs that AI for CICS components will call. You can find the COMMAREA for a CICS program in copybooks, in separate files, or in main code.

For more information about how to prepare the COMMAREA for importing, see Chapter 2, “Getting Ready to Use AI for CICS.”

Step 2: Create the Component

In this step, you use the Component Builder to create a connection and a component. The connection provides access to a CICS program. The component contains a collection of methods, and each method invokes a single CICS program.

After you have created the component, the Component Builder adds methods by importing the COMMAREA definitions.

Step 3: Deploy the Component

When you deploy the component, the Component Builder defines it in Jaguar, and creates Java classes that implement the component. After the component is deployed, it is ready to run.

Step 4: Use the Component

Once a component is deployed, you can use it in applications. AI for CICS components may be used to build Jaguar client applications, or to build other Jaguar components.

Note For information about how to build your client application, see the Jaguar documentation.

Using the AI for CICS Tutorial

Sybase provides tutorials to give you hands-on experience and guide you through each detailed step of the process just described.

Tutorials are installed with AI for CICS. See the online documentation in the Sybase\Application Integrator 3.0 program group for information on running the AI for CICS tutorial.

CHAPTER 2

Getting Ready to Use AI for CICS

This chapter describes how to determine whether AI for CICS can uses a CICS program directly. It also describes the COMMAREA data definition that AI for CICS uses to create a component and provides a worksheet for gathering the information that AI for CICS requests.

The following topics are covered in this section:

- Selecting a CICS Program
- Understanding COMMAREAs
- AI for CICS Data Definition Worksheet

Note The information in this chapter is primarily for use by the CICS developer.

Selecting a CICS Program

Before you use AI for CICS, be sure that the CICS program you plan to use in the component meets the criteria listed in this section

This section covers the following topics:

- CICS Program Requirements
- Data Definition Restrictions
- If Your CICS Program Does Not Meet All Criteria

CICS Program Requirements

Following are requirements for the CICS program that you select:

- The program can be called using a CICS Link call.
- The program does not perform any 3270 screen or printer I/O directly through an LU.
- The program has a communications area (COMMAREA).
- All input and output data is handled in the communications area.

Data Definition Restrictions

Following are restrictions for the data definition file (COMMAREA):

- It cannot contain nested **OCCURS** clauses.
- It cannot have a variable length **OCCURS** clause, unless it is the last definition in the COMMAREA.
- It cannot contain a **REDEFINES** clause.

If Your CICS Program Does Not Meet All Criteria

If the CICS programs you want to integrate with AI for CICS do not meet all of the previous criteria, you can:

- Modify an existing CICS program.

- Write an Open ServerConnect application and call it as a stored procedure. (For more information, see the *Application Integrator for Stored Procedures Reference*.)
- Write a CICS program that meets the criteria.

Understanding COMMAREAs

This section covers the following topics:

- Defining the COMMAREA
- How CICS Programs Use the COMMAREA
- How AI for CICS Components Map from the COMMAREA
- Making the COMMAREA Available to AI for CICS

Defining the COMMAREA

Modern CICS programs store presentation logic and business logic in separate programs. When the two programs need to communicate with each other, they must pass data back and forth in a COMMAREA, which is a block of contiguous memory.

Because the CICS programs must agree on the format and use of the data definitions in the COMMAREA, this area is usually mapped by a single COBOL data definition that is included in both participating programs.

The COMMAREA (data definition file) can be stored as follows:

- Embedded in CICS program source code
- In a copybook member that is copied in at compile time
- In a separate file, copied from the source code

Note See 28 for more information about how to import the COMMAREA from each of these stored locations.

Files imported into the AI for CICS Component Builder must contain a COMMAREA for the CICS program you want and *nothing else*, as shown in the following example:

```
01 DFHCOMMAREA.
     02  CA-RETCODE                  PIC 9(8) COMP.
     02  CA-SWSECI1-COMMAREA.
         05  CA-NUMBER-OF-ROWS       PIC 9(4) COMP.
         05  CA-ERROR-MESSAGE        PIC X(10).
         05  CA-CURRENT-DATE         PIC X(8).
         05  CA-CURRENT-TIME         PIC X(8).
         05  CA-CICS-ABSTIME         PIC S9(15) COMP-3.
```

```
05  CA-ROW-DATA  OCCURS 1  TO 1818 TIMES
                 DEPENDING ON CA-NUMBER-OF-ROWS.
    10  CA-ROW-NUMBER       PIC S9(4) COMP.
    10  CA-ROW-NUM-AS-CHAR  PIC X(6).
    10  CA-DATA             PIC X(10).
```

Note When you move a COMMAREA to the workstation, be sure you remove all **REDEFINES** clauses and field declarations following **REDEFINES** clauses.

How CICS Programs Use the COMMAREA

To communicate input and output information, the "calling" program typically sets input fields in the COMMAREA and calls another CICS program, passing the COMMAREA to it. The "called" program then puts the results of this operation back into fields in the COMMAREA and returns execution to the calling program. This makes the output for the called program available to the calling program through the COMMAREA.

Figure 2-1 shows this process: PROGA (the presentation logic) invokes PROGB (the business logic) and passes the COMMAREA to it, thus defining the input and output definitions that they will share.

Figure 2-1: CICS programs using the COMMAREA

How AI for CICS Components Map from the COMMAREA

The following figure shows how AI for CICS maps the data definitions from the COMMAREA.

Figure 2-2: How AI for CICS maps from the COMMAREA

Following are the basic steps in this process:

1 When you build an AI for CICS component, AI for CICS parses the COMMAREA in the copybook and stores the data definitions.

2 When you generate the component, AI for CICS creates a Jaguar component and maps the COMMAREA for PROGB business logic into structured parameters that the Jaguar component can understand.

3 When the method in the Jaguar component calls the CICS program, the structured parameters are mapped to the CICS fields, as shown in the **invoke** call. This presents a “component-like” interface to client applications.

Making the COMMAREA Available to AI for CICS

This section describes how to make data definitions in the COMMAREA available on the workstation so that AI for CICS can use them. Following are descriptions based on how you import the COMMAREA:

- From COBOL Source Code
- From a Copybook Member
- From a Separate File

From COBOL Source Code

If the COMMAREA is embedded in source code, follow these steps to make it available to AI for CICS:

1 Cut and paste the source code portions of data definitions into a file.

2 Use **FTP**, a screen capture, **IND$FILE**, or a similar utility to copy the file and move it to the workstation where AI for CICS is installed.

If necessary, be sure that the transferred files receive EBCDIC-to-ASCII translation so they are readable characters on the workstation.

3 Record the name of the copied file for future reference.

From a Copybook Member

In mainframe applications, many sections of program code are common to multiple programs, such as file layouts, communication area layouts, common data definitions, and file descriptions. Most compilers are able to include code fragments from a library at compile time.

In COBOL, libraries of these code fragments are called copybooks, and the individual members within them are called copybook members.

Follow these steps to make the copybook member available to AI for CICS:

1 Find the source code of the program you want to make available as a component.

Note If your site uses code management utilities such as **CA-LIBRARIAN** or **PANVALET**, contact your mainframe systems administrator for information on locating copybooks in your environment.

---

2 Find the **COPY** statements that look like this:

```
COPY abcd
```

3 Locate the relevant members in your copybook library. If you are not sure which copybooks to use, look at the SYSLIB DDNAME in the JCL used to compile the program. One or more of the data sets allocated to this DDNAME will be the copybook for this compilation.

4 Find the member in the copybooks that describes the COMMAREA for the CICS program you will use to build a component. This is the copybook member you import into AI for CICS.

5 Using **FTP**, a screen capture, **IND$FILE**, or a similar utility, copy the copybook member and move it to the workstation where AI for CICS is installed.

   If necessary, be sure that the transferred files receive EBCDIC-to-ASCII translation so they are readable characters on the workstation.

6 Record the CICS program name, copybook name, and copybook member names for future reference.

Note If the copybook member contains more than the COMMAREA definitions, edit the downloaded file so that it contains only the COMMAREA.

From a Separate File

Follow these steps to make the COMMAREA available to AI for CICS:

1 Using **FTP**, a screen capture, **IND$FILE**, or a similar utility, copy the file.

   If necessary, be sure that the transferred file receives EBCDIC-to-ASCII translation so it is displayed as readable characters on the workstation.

2 Move it to the workstation where AI for CICS is installed.

3 Record the filename for future reference.

Note If the file contains more than the COMMAREA definitions, edit the downloaded file so that it contains only the COMMAREA.

AI for CICS Data Definition Worksheet

Record the following information for each COMMAREA data definition.

Table 2-1: AI for CICS worksheet for each method

Information needed	Record information here
CICS program name (for example, SWSECI1)	
Copybook library or PDS name (for example, COBOL.COPYLIB.COB)	
Copybook member name (for example, FILE02)	
PC filename and location for the COMMAREA (for example, C:\AICICS\SWECI1.COB)	

CHAPTER 3

Working with the AI Component Builder

This chapter contains information about using the AI for CICS Component Builder, including the following topics:

- Working with the Component Builder
- Understanding AI Connections
- Understanding AI Components
- Supported Datatypes
- Understanding AI Component Deployment
- Mapping CICS Components to AI Components

Note Sybase provides samples with the Application Integrator software. These samples include tutorials designed to help you learn to use the Component Builder to create components and connections. See the online documentation in the Sybase\Application Integrator 3.0 program group for information on running the tutorials.

Working with the Component Builder

The AI for CICS Component Builder lets you easily create Jaguar components that invoke existing CICS programs. Like all AI Component Builders, the AI for CICS Component Builder runs in Sybase Central, a system management and design tool. Sybase Central is also used with Sybase servers such as Jaguar and Adaptive Server Anywhere (ASA).

You use the AI for CICS Component Builder to perform the following tasks:

- Create, test, and edit connections
- Create and edit components
- Deploy components to Jaguar

This section covers the following topics:

- Starting the Component Builder
- Working with Project Files

Starting the Component Builder

To start the Component Builder, from the Windows Start menu, select AI Component Builder from the Sybase\Application Integrator 3.0 program group.

This starts Sybase Central and loads any AI Component Builders you have installed. The AI Component Builders appear in the tree view in the Sybase Central window.

Working with Project Files

Project files provide a way to store component and connection definitions in the Component Builder. Project files store definitions and information about logically-related components and connections. For example, you can group all components created for a specific department into a single project file, such as a Human Resources or Payroll file. Or, if you are creating components from CICS programs that reside in different CICS regions, you can save all components and connections created for a specific region in a single project file.

The Component Builder assigns the *.aip* extension to new, untitled project files. We recommend that you continue to use the *.aip* extension for each project file you save.

This section covers these topics:

- Creating, Opening, and Closing Projects
- Viewing Project Contents
- Saving Project Files

Creating, Opening, and Closing Projects

To create a new project, right-click the Component Builder and click New Project.

To open a project file, right-click the Component Builder and click Open Project.

To close a project file, right-click the project and click Close.

Viewing Project Contents

Each project contains a collection of components and connections. These objects are displayed hierarchically in the Sybase Central tree view. You view the contents of a project by expanding the objects that appear beneath a project in the tree view. To expand an object in the tree, you can either left-click the + button that appears next to the object, or double-click the object.

The following figure shows a project file named *new_cics* in the CICS Component Builder.

Figure 3-1: Expanded project file

In the *new_cics* project, the component named *Calculator* contains two methods called *multiply* and *divide*. The connection associated with this component is *Sungard*.

The right-hand window in Sybase Central window contains different views of the object currently selected in the tree. The Details view displays children of the object selected in the tree. Some objects include other views that provide additional information. For example, when you select a component in the tree, you can see both Details and Interface views for the component:

- The *Details* view displays the methods in the component.
- The *Interface* view displays the CORBA IDL (Interface Definition Language) interface for the component.

Saving Project Files

To save your project, right-click on it and click Save. This saves the project to a file, so you can use it again later.

Understanding AI Connections

An Application Integrator connection is a named set of properties that are used to access a CICS program. Every deployed component is associated with a connection definition by name.

A connection identifies how the component and its methods can locate the AI for CICS server, which must be installed and configured in the CICS region where the CICS program resides. (For information about installing AI for CICS on the workstation and the mainframe, see the *Enterprise Application Studio Installation Guide.)*

Connections are used in the Component Builder at design time, and in Jaguar at run time. In the Component Builder, connections can be tested to verify end-to-end connectivity. In Jaguar, connections are used by component methods to invoke CICS programs.

You use the Component Builder to define connection properties, associate them with components, and deploy them to Jaguar. Every component is associated with exactly one connection.

This section covers the following topics:

- Determining Connectivity
- Connection Properties
- Testing Connections
- Connection Caching
- Connections and Deployed Components

Determining Connectivity

There are two ways to connect to the CICS region, as shown in Figure 3-2:

Figure 3-2: Gateway versus gateway-less connectivity

- *Gateway connectivity:*

  If you are using the SNA protocol to access the AI for CICS server, provide connectivity from the Jaguar server to CICS through DirectConnect TRS. For this connectivity method, you need to know the TRS Service Name when prompted.

- *Gateway-less connectivity:*

  If you are using TCP/IP to access the AI for CICS server, you can connect directly to the CICS server from the Jaguar server.

Connection Properties

To create a connection, you must supply the following information in the Component Builder:

- CICS user ID and password needed to access the target CICS program
- Properties needed to connect to the target CICS program

The following table shows all of the requested connection properties:

Table 3-1: Connection properties

Dialog box	Field	Description
Create Connection	Host name	TCP/IP server name. Name of the machine, either gateway or mainframe.
	Port number	TCP/IP port that the gateway or mainframe is listening on. Valid value is any that matches a listener on the Host name.
	TRS service name	Name assigned to DirectConnect Transaction Router Service. Required for gateway connectivity.
	User name	Valid CICS user ID.
	User password	Password for CICS user ID.
General	Name	Name of connection.
	Description	Optional short description of the connection. This description will appear in the Details area of the project file tree view on the Sybase Central window.

Dialog box	Field	Description
Connectivity	RPC name	Name assigned to the RPC for the CICS run time.
	Code page	Number that identifies a set of characters with an encoding scheme that uniquely defines each character. Also called code set or character set. Click Select to display a list of code page options.

Testing Connections

The Component Builder allows you to test a connection to the target CICS region where the AI for CICS server run time resides. This makes it easy to verify that the region is running, and that you have defined the connection correctly.

Common reasons that connection tests fail are:

- The network or CICS region is down.
- Invalid values, such as host name, port number, user name, or password, were specified for the connection.
- AI run time (CICS) is not installed.

Note Because connectivity problems are difficult to identify at run time, we recommend that you test connections in the Component Builder before you deploy components and methods that use them.

Connection Caching

AI components do not use Jaguar's connection caching feature by default. Components can be configured to use the connection caching feature in Jaguar. See Chapter 4, "Jaguar and Application Issues," for details.

Connections and Deployed Components

When you deploy a component into Jaguar, the Component Builder uses the connection definition that was associated with the component in the Component Builder.

The information for the component's connection is sent to Jaguar in the form of a serialized file, which is located in the Jaguar installation directory structure (in the *java\classes\SybAIConnections* directory). If you need to change connection information for a component, you must edit the connection in the Component Builder, and then redeploy the component and connection.

Understanding AI Components

AI for CICS components are Jaguar components that provide access to one or more CICS programs. A component contains one method for each CICS program it exposes, and each method invokes exactly one CICS program.

Every component is associated with exactly one connection. The choice of connection determines which CICS programs the component's methods can invoke. All CICS programs exposed in a component must reside in a single CICS region.

After creating the component and its methods, the AI Component Builder can be used to automatically deploy it to Jaguar, where it is used to build LAN- or Web-based applications. The result is that, using the component, the client application can access several existing CICS programs that read and/or write mainframe data.

Once you have created a component and added one or more methods, you can deploy the component to Jaguar, where it can then be used by Jaguar client applications or other Jaguar components.

The person responsible for using AI for CICS components to build applications should perform the steps of the component creation process.

The rest of this section covers the following topics:

- Component Properties
- Method Properties
- Viewing Method Information

Component Properties

Note Before you create a component, be sure you know how COBOL datatypes map to Java datatypes. See "Supported Datatypes" on page 44.

The Component Builder allows you to associate multiple methods with a single component.

For each component that you create, you must supply the following information in the Component Builder:

Table 3-2: Component properties

Tab	Field	Description
General	Name	Name used for the component when it is deployed to Jaguar.
	Java package name	Components are implemented as Java classes. Java package names are used to organize Java classes hierarchically. Typically, Java package names are in reverse Internet domain name format (as in the example *com.xyzcorp.app_name*, where app_name is the name of the application you are building). **Note** Java package names are not the same as Jaguar package names.
Connection	Connection	Each component requires a connection. When components are created in the Component Builder, you can select an existing connection, or create a new one.

Method Properties

The Component Builder requests the following properties on the Method Properties dialog box.

Table 3-3: Method properties

Tab	Field	Description
General	Name	Name that Jaguar clients will use when invoking the method.
	Description	Description of the method. An optional field.
	Parameters:	
	Name	Java name derived from the mapped COBOL field.
	Mode	Defaults to inout. To edit, click the current mode and select from the list.
	Type	Jaguar datatype of each mapped COBOL field. To edit, click the current type and select from the list. See “Supported Datatypes” on page 44.
	COMMAREA Definition	Name of each field in the target COMMAREA definition. This field is not editable.

Tab	Field	Description
Result Sets	Name	Name of the result set for the OCCURS clause in the target COMMAREA definition.
	Type	Jaguar datatype that maps to the COBOL field.
	COMMAREA Field	Name of the field in the COMMAREA of the target COBOL program.
Target	COBOL Program	Name of the target COBOL program for this method. This field is not editable.
	COMMAREA	Displays the COMMAREA for this method. This field is not editable.

Viewing Method Information

When you add a method to a component, you can select the method name and click the following tabs for information:

- *Details*, which includes the name of each method in the component (if you have added methods), the target name, and an optional description
- *Interface*, which shows the representation of this component using CORBA IDL (Interface Definition Language)

Supported Datatypes

This section covers the following topics:

- COBOL FILLER and Group Level Items
- COBOL Datatype Support
- Additional Datatype Information

COBOL *FILLER* and Group Level Items

Fields declared as FILLER in the COBOL COMMAREA definition are not displayed as fields in the component, and the corresponding storage is allocated but inaccessible to the component user.

Also, AI for CICS only displays information about elementary fields in the component. Group-level items are not displayed.

COBOL Datatype Support

The AI for CICS Component Builder supports the following datatypes in the COMMAREA.

Table 3-4: Supported COBOL datatypes in the COMMAREA

COBOL Datatype	Picture Clause	Java Datatype	Jaguar IDL Datatype
Character Data	PIC X(n) or PIC A(n)	java.lang.string	string
Numeric Display	PIC S9(n), or PIC S9(n)V9(n)	*See* Table 3-6 on page 46.	
Numeric Binary	PIC S9(n) COMP	*See*Table 3-6 on page 46.	
Numeric Packed	PIC S9(n) COMP-3, or PIC S9(n)V9(n) COMP-3	*See* Table 3-6 on page 46.	

Note The AI for CICS Component Builder does not support COBOL fields defined as *USAGE* COMP-1, *USAGE* COMP-2, or PIC G.

Additional Datatype Information

This section describes the default datatypes that AI for CICS maps based on the COBOL datatype.

This section covers the following topics:

- Alphanumeric Datatypes
- Alphabetic Datatypes
- Numeric Datatypes
- Input and Output Parameter Information

When the Component Builder parses a COBOL COMMAREA data definition, it maps the COBOL datatypes to datatypes used by AI for CICS and Jaguar.

In the Component Builder, you can change the datatype that the parser determined to *binary*. When binary is selected as the Jaguar datatype, the AI Adapter does not perform conversion to or from the format of the underlying COBOL storage datatype. Instead, this field is treated as raw data that the client application programmer must interpret and format correctly. However, the AI Adapter will throw an exception if the length of the byte array passed as an input parameter exceeds the length of the underlying COBOL storage. If the array length of an input parameter is less than the underlying COBOL storage, only the number of bytes in the array are placed into the storage for that field.

Alphanumeric Datatypes

Alphanumeric datatypes (PICTURE X(n)) are subject to character set conversion, such as ASCII to EBCDIC, using the code page indicated on the component's associated connection definition. Keep the following in mind:

- Input values shorter than the actual COBOL definition are padded with spaces at the end.
- Input values longer than the COBOL definition cause an exception to be thrown.
- Output values are exactly the same length as defined in the COBOL definition.

Alphabetic Datatypes

Alphabetic datatypes (PICTURE A(n)) are treated the same as alphanumeric datatypes. AI for CICS does not verify that valid alphabetic characters are stored in these fields.

Keep the following in mind:

- Input values shorter than the actual COBOL definition are padded with spaces at the end.
- Input values longer than the COBOL definition cause an exception to be thrown.
- Output values are exactly the same length as defined in the COBOL definition.

Numeric Datatypes

AI for CICS supports numeric datatypes as follows:

Table 3-5: Numeric datatypes and COBOL USAGE clauses

Numeric Datatype	COBOL USAGE Clauses
Binary	• COMP • COMPUTATIONAL • BINARY
Packed Decimal	• COMP-3 • COMPUTATIONAL-3 • PACKED-DECIMAL
Zoned Decimal	DISPLAY

As shown in Table 3-6, all formats of default and allowable Java datatypes depend on precision (total number of digits in PIC clause) and scale (number of digits to the right of the decimal).

Table 3-6: Numeric datatype mappings according to precision and scale

Precision	Scale	Default Java Datatype	Jaguar IDL Datatypes
1 - 4	0	short	integer<16>
5 - 9	0	int	integer<32>
10 - 18	0	long	integer<64>
any	> 0	java.math.BigDecimal	decimal

Input and Output Parameter Information

Following are additional points to consider:

- If input parameters passed as BigDecimal have a scale or precision that is greater than defined for the field in the COBOL definition, an exception will be thrown.
- If input parameters passed as BigDecimal have a scale that is less than the COBOL definition, the scale will be padded with zeros for the remaining least significant digits.
- If input parameters have a precision less than the COBOL definition, the value will be padded with zeros for the remaining most significant digits.
- Output parameters specified as datatype BigDecimal will have the same precision and scale identified in the COBOL definition.

Understanding AI Component Deployment

Once you have defined a component that has one or more methods, you can deploy it to Jaguar, where it can then be used in Jaguar client applications.

This section covers the following topics:

- Understanding the Deployment Wizard
- Understanding Deployed Components in Jaguar
- Understanding Deployed Connections in Jaguar

Understanding the Deployment Wizard

The Component Builder provides a wizard that makes it easy to deploy components. The wizard performs the following functions:

- Defines the component in Jaguar
- Copies the connection and Java source code for the component to the Jaguar *java\classes* directory
- Compiles the Java source code for the component on the Jaguar server

When you deploy a component, you must provide the information on the dialog boxes described in the following table:

Table 3-7: Information needed by the Deploy Components wizard

Wizard Dialog Box	Field	Description
Jaguar connection information	User name	Valid Jaguar user ID.
	Password	Valid password for Jaguar user ID.
	Host name	Name of the machine on which Jaguar is running. **Note** Be sure that Jaguar is running.
	Port number	Number of the port on which Jaguar is listening.

Wizard Dialog Box	Field	Description
Jaguar package information	Package name	The package name currently selected for this component. Enter a new name, accept the selected name, or select a different name from the Existing Packages field.
	Existing packages	The list of package names currently available. Click a package name to select the package.

After you provide the information needed by the wizard, a progress window appears and provides you with status as the components are deployed. If deployment is not successful, use the messages that appear in the progress window to troubleshoot errors.

Note For each Jaguar server you deploy components into, the first time you deploy a component, a message indicates that the feature for deploying components is not yet installed in the Jaguar server. Click OK to install this feature.

Understanding Deployed Components in Jaguar

Each AI for CICS component built and deployed to Jaguar consists of two or more Java classes that work with the AI for CICS Adapter, which allows those CICS programs to be viewed in Jaguar as components.

The Adapter consists of a set of Application Integrator classes that are also referred to as the Application Integrator run-time classes. Internally, the Adapter provides access to CICS programs; it also provides security, transaction control, datatype conversion, and error handling integration.

For each component that is deployed, AI for CICS creates (or modifies, if one already exists) the following items on the Jaguar server:

- A Jaguar package (modified, if it already exists)
- A Jaguar component definition
- An IDL module (modified, if it already exists)
- An IDL interface
- Java source files for the component

- Java class files for the component
- A file that contains the component's connection

When components are deployed into Jaguar, the following steps occur:

1 The Jaguar package and IDL module are created only if they do not already exist:

   - If the package already exists, AI for CICS adds the new component to the existing package.
   - If the module already exists, AI for CICS adds the new IDL interface to the existing module.

2 The Application Integrator deployment feature creates implementation files and stores them in the *<jaguar_install_dir>\java\classes* directory.

3 The component connection is created in a file named *connection_name.ser* in the Jaguar *java\classes\SybAiConnections* directory.

4 The Application Integrator component definition is added to the Jaguar repository in addition to standard Jaguar component properties. Application Integrator components also include a collection of user-defined Jaguar properties, which allow you to configure additional options for the component, such as debugging and connection caching.

   If the deployed component properties need to be changed, you can edit them in Jaguar Manager. See Chapter 4, "Jaguar and Application Issues," for details.

Understanding Deployed Connections in Jaguar

When you deploy a component into Jaguar, the Component Builder also deploys the connection that is associated with the component.

When a component is deployed into Jaguar, the information for the component's connection is sent to Jaguar in the form of a serialized file, which is located in the Jaguar installation directory structure (in the *java\classes\SybAIConnections* directory). If you need to change connection information for a component, you must edit the connection in the Component Builder, and then redeploy the component and connection.

Note Sybase provides samples with the Application Integrator software. These samples include tutorials designed to help you learn to use the Component Builder to create components and connections. See the online documentation in the Sybase\Application Integrator 3.0 program group for information on running the tutorials.

Mapping CICS Components to AI Components

This section contains information about the relationships between target CICS programs and corresponding components in AI for CICS and Jaguar.

This section includes the following topics:

- Target CICS Programs and Method Names
- Result Sets and Method Return Value
- Parameter Information
- An Illustration of CICS Program-to-Component Mapping

Target CICS Programs and Method Names

Each Application Integrator component deployed to Jaguar contains one or more methods, each of which corresponds to a target CICS program. When you add a method to a component, the Component Builder uses the name of the target CICS program, with the first character in upper case and the subsequent characters in lower case.

Result Sets and Method Return Value

When the Component Builder parser detects an **OCCURS** clause in the CICS program, it maps the **OCCURS** clause to a result set, establishing one result set field for each field in the **OCCURS** clause in the COMMAREA definition.

The following table shows how the number of **OCCURS** clauses encountered are mapped in Jaguar:

Table 3-8: Result sets and method return values

Number of OCCURS Clauses	Jaguar IDL Mapping
0	void
1	TabularResults::ResultSet
More than 1	TabularResults::ResultSets

See Chapter 2, “Getting Ready to Use AI for CICS,” for restrictions on **OCCURS** clauses.

Parameter Information

This section covers the following topics:

- Parameter Names
- Parameter Modes

Parameter Names

For each method in an AI for CICS component, each parameter corresponds to the fields from the COMMAREA that are described by the COBOL data definition. Parameter names are derived from the corresponding COBOL fields and are altered to conform to valid Java field names. For example, the Component Builder keeps the first character in upper case, makes characters immediately following dashes ("-") in upper case, and removes the dashes.

You can also alter parameter names in the Component Builder.

Parameter Modes

Because the COBOL data definition does not indicate which fields the CICS program uses in input, input/output, or output, the Component Builder sets the defaults to **inout** for all fields. You can change the mode to **in**, **out**, or **inout**.

However, you must set the mode correctly for all fields based on how the CICS program uses the field. For example:

- If the CICS program uses a field for a database query, or if it uses the field in a calculation but never alters the value, define the field in the Component Builder with a mode of **in**.
- If a field contains a newly-calculated or derived value, define it in the Component Builder as **out**.
- If a field used by the CICS program has a value that can change, define the field as **inout**.

An Illustration of CICS Program-to-Component Mapping

The example in this section shows how AI for CICS maps COBOL programs to components.

A CICS COBOL program called *ZXR150R* is designed to get a list of customer names, addresses, and phone numbers of all customers for a specific account representative, in descending order of total sales amount. The program caller can limit the number of customers that the program will return, for example, up to ten.

The following COBOL data definition describes the COMMAREA for the CICS program:

```
01 COMMAREA.
   05 ACCOUNT-REP-ID          PIC X(5).
   05 NUMBER-OF-CUSTOMERS     PIC 9(4) COMP-3.
   05 ACCOUNT-REP-QUOTA       PIC 9(8)V9(2) COMP-3.
   05 CUSTOMER-DATA OCCURS 1 TO 500 DEPENDING ON
    NUMBER-OF-CUSTOMERS.
     10 CUST-NAME             PIC X(25).
     10 CUST-ADDR             PIC X(25).
     10 CUST-CITY             PIC X(15).
     10 CUST-STATE            PIC C(2).
     10 CUST-ZIP              PIC X(10).
```

The preceding data definition does not clearly indicate whether the parameters are used for input, output, or both.

The Component Builder uses this definition to set the modes for the parameters as follows:

Table 3-9: Parameter definitions

Parameter Name	Type	Mode
accountRepId	*string*	inout
numberOfCustomers	*integer <16>*	inout
accountRepQuota	*decimal*	inout

Assume that the logic for the CICS program is as follows:

- The *ACCOUNT-REP-ID* field is used to query for customers serviced by that account rep, in descending order of the customers year-to-date sales.
- The *NUMBER-OF-CUSTOMER*S field will be used to limit the number of customers.
- The data from the query populates the fields in the *CUSTOMER-DATA* group item that contains the **OCCURS** clause.
- If the actual number of the account representative's customers is less than the value in the *NUMBER-OF-CUSTOMERS* field, set *NUMBER-OF-CUSTOMERS* to the actual number of customers.

- Query the database to find this account representative's sales quota and insert this value in the *ACCOUNT-REP-QUOTA* field.
- You may want to change the method name that is defaulted from the CICS program (altered by AI for CICS to *zxr150r*) to getTopCustomerList, which is more meaningful.

You must set parameter modes as follows:

- Because the CICS program only uses the *ACCOUNT-REP-ID* field to query information and does not alter its value, change the mode of the *accountRepId* parameter from **inout** to **in**.
- Because the caller can use the *NUMBER-OF-CUSTOMERS* field to indicate the maximum number of customers to return, and the CICS program itself can alter the field to contain a different value, allow the mode of the *numberOfCustomers* parameter to remain as **inout**.
- Because the *ACCOUNT-REP-QUOTA* field is used exclusively to pass a value back from the CICS program, change the mode of the *accountRepQuota* parameter to **out**.

The Resulting Component Definition

If you customize the method name, parameter names, and parameter modes of this component as described, it will accurately represent the CICS program.

In Jaguar, this component will be called *AccountRep* (the same name you assigned the component using the Component Builder), and it will have a method named *getTopCustomerList* that returns a *ResultSet*. The parameters for that method will be defined appropriately, based on their functions in the program.

CHAPTER 4

Jaguar and Application Issues

This chapter describes the characteristics of Application Integrator components in Jaguar that affect programming client applications in Jaguar. This chapter contains the following topics:

- AI Components in Jaguar
- Connections and Connection Caching
- Client Application Development

To help you understand this material, we recommend that you familiarize yourself with the following Jaguar-related topics in the *Jaguar CTS Programmer's Guide*:

- Java components in Jaguar
- Transactions and component life cycles
- Connection management

AI Components in Jaguar

When the Component Builder deploys a component into Jaguar, a new component is created in the specified Jaguar package using the Jaguar Java component model. The new component's interface (its methods and their arguments) is described using CORBA IDL.

Application Integrator components are designed to take advantage of the Jaguar instance pooling feature. This section covers the following topics:

- Instance Creation
- Instance Activation
- Method Invocation
- Instance Deactivation
- Instance Destruction

Instance Creation

Upon receiving a client request for a method, Jaguar creates a new instance of a component if no instances are available in the pool for reuse. If an unused instance already exists in the pool, Jaguar can initialize that instance by calling the **activate** method and allowing a client application to call a method on the component. During component instantiation, a one-time initialization routine retrieves component property information.

Instance Activation

Jaguar activates an Application Integrator component instance as a result of a client method invocation request. When activated, the component attempts to connect to its target server using the connection properties that you deployed along with the component. Depending on its configuration, the component can obtain its connection from a connection cache defined in Jaguar, or it can open a connection directly to the server.

If an exception occurs during component activation, the exception information is held and returned on the first method call to that instance. The most common reasons for exceptions during component activation result from problems connecting to the server, such as invalid security values, server not available, and invalid connection properties.

Method Invocation

When a client executes methods on a proxy object for an Application Integrator component, Jaguar invokes the methods that correspond to a target COBOL program. Method invocations prompt the Application Integrator component to execute the COBOL program. The method's input parameters (parameters defined with the modes **in** or **inout**) are passed to the program. After the program executes, any result sets are returned to the caller as the return value of the method call, and any output parameters (parameters defined as **inout** or **out**) are returned to the caller.

Instance Deactivation

Jaguar deactivates each Application Integrator component instance when it no longer needs to keep that instance associated with the client that caused its activation. During instance deactivation, the connection that was obtained during activation is returned.

Instance Destruction

Jaguar destroys pooled component instances in the following circumstances:

- A nonrecoverable error occurs during construction, activation, or deactivation. (Messages about these errors are recorded in the Jaguar server log.)
- The Jaguar server shuts down.
- A component is refreshed.
- After the component method completes, if the client invoked the method using the Jaguar MASP feature.

Connections and Connection Caching

This section discusses how Application Integrator components use connections to target servers, including the following topics:

- Component Properties: Connections and Security
- Acquiring and Disposing of Connections
- Defining Connection Caches in Jaguar Manager
- AI Component Properties in Jaguar Components

Component Properties: Connections and Security

When the Component Builder deploys a component into Jaguar, it sets the following properties for the component. The following are properties that relate to connection handling and security:

com.sybase.ai.connection_name

This property is referred to as *connection_name*. It specifies the name of the connection associated with the component.

com.sybase.ai.use_client_security

This property is referred to as *use_client_security*. It affects the user ID and password that are used to obtain connections. Valid values are:

- **false**

  A value of **false** tells the Application Integrator component to use the user ID and password that were supplied in the connection object from the Component Builder. The default value is **false**.

- **true**

  A value of **true** tells the component to use the user ID and password from the client that invoked the component method. When the value is **true**, the caller's user ID and password are obtained from Jaguar and are subsequently used to override the user ID and password from the connection definition.

com.sybase.ai.use_connection_cache

This property is referred to as *use_connection_cache*. It specifies how the Application Integrator component acquires and disposes of connections. Valid values are:

- **false**

  A value of **false** (the default) tells the Application Integrator component to open and close connections to a server directly.

- **true**

  A value of **true** tells the component to get and release connections from a Jaguar connection cache.

com.sybase.ai.connection_cache_unavail_action

This property is referred to as connection_cache_unavail_action. It controls the action that Jaguar takes if there are no connections available in the cache. Valid values correspond to the possible values for the flag parameter on the **JCMCache.getConnection()** method. (See the connection cache section of the *Jaguar CTS Programmer's Guide* for a more detailed discussion). Valid values are:

- **nowait**

  Indicates that if no connections are available, the request results in an exception (error condition) that is then propagated back to the caller. The default is **nowait**.

- **force**

  Indicates that Jaguar should attempt to open a new connection (that will not be placed back in the cache after use).

- **wait**

  Indicates that the component waits indefinitely until a connection becomes available. We do not recommend that you use this option because this indefinite wait can cause component instances to tie up resources and can cause the client application to wait indefinitely.

Acquiring and Disposing of Connections

Connections created in the Component Builder give an Application Integrator component the information it needs to connect to a CICS server at execution time. The CICS server is an Application Integrator-supplied CICS program that Application Integrator components communicate with to execute a CICS application program.

The process of acquiring and disposing of connections differs depending on whether the Application Integrator component is configured to use connection caching.

When connection caching is ***not*** being used:

- **Acquiring connections** refers to the process of opening a new connection directly to the CICS server. The Application Integrator component uses the values supplied in the connection definition with the name that matches the *connection_name* property. These values include such information as the TCP/IP host name, a port number, an Open ServerConnect RPC name, and a Code Page.
- **Disposing of connections** refers to closing connections.

When connection caching ***is*** being used:

- **Acquiring connections** refers to the process of retrieving a connection from the Jaguar connection cache mechanism. When the Application Integrator component needs to get a connection to the target server, it uses the *connection_name* property to locate a Jaguar connection cache by name.
- **Disposing of connections** refers to releasing a connection back to the Jaguar connection cache for other components to use.

Defining Connection Caches in Jaguar Manager

Defining connection caches for Application Integrator components is similar to defining them for other components that use JDBC connection caches.

You must set up connection cache definitions in Jaguar Manager using the following procedure. In general, use the same information that was used to create the connection in the Component Builder. To define a connection cache:

1 On the Driver tab of the Connection Cache Properties dialog box, select the JDBC 1.1 radio button.

2 Enter the JDBC driver class name in the DLL or Class Name as follows:

com.sybase.ai.common.adapter.AIJagDriver

3 On the General tab:

a Set the Server Name property in the following format:

jdbc:sybase_ai:ai:<connection_name>

where *<connection_name>* is the connection name associated with any AI for CICS components that should use this connection.

b Enter a valid user name and password.

4 Click the Enable cache-by-name access check box to turn this property **on** (it is **off** by default). This allows the Application Integrator component to look up connection caches by name.

Note Setting the Jaguar Enable cache-by-name access property requires Jaguar Administrator authority.

5 Restart the Jaguar server to make the changes to connection cache definitions take effect.

Note Make sure the connection cache is installed in a running Jaguar server before you use it in a component to connect to a target.

AI Component Properties in Jaguar Components

In addition to the properties that all Java-style Jaguar components have, Application Integrator adds some properties to the component in the Jaguar component repository. Application Integrator-specific properties begin with the *com.sybase.ai* prefix. You must refresh the component in Jaguar Manager for property changes to take effect.

To edit these properties in Jaguar Manager, use the All Properties tab on the Component Properties dialog box. Click on the column heading once to sort the properties alphabetically by name in ascending order; click on the column heading again to sort the properties alphabetically in descending order.

Table 4-1: Component properties and where they are set

Component property name	Where property is set
com.sybase.ai.connection.name	During Application Integrator Component Builder deployment
com.sybase.ai.use_connection_cache	Jaguar Manager
com.sybase.ai.connection_cache_unavail_action	Jaguar Manager
com.sybase.ai.use_client_security	Jaguar Manager
com.sybase.ai.debug	Jaguar Manager

Do not modify any of the following component properties unless instructed to do so by Sybase Technical Support:

- *com.sybase.jaguar.component.files*

- *com.sybase.jaguar.component.ids*
- *com.sybase.jaguar.component.java.classes*
- *com.sybase.jaguar.component.interfaces*
- *com.sybase.jaguar.component.java.class*
- *com.sybase.jaguar.component.type*
- *com.sybase.jaguar.component.name*
- *com.sybase.jaguar.component.bind.object*
- *com.sybase.jaguar.component.bind.thread*

Component Execution Debugging

The *com.sybase.ai.debug* component property can be used to cause the component to write detailed trace information to the Jaguar server log. The property can have a value of **true** to indicate that the component should write trace information to the Jaguar server log. A value of **false** indicates that the component should not write information to the log. The default value is **false**.

Warning! Turning on debugging can cause a large amount of data to be logged, which can adversely affect performance.

In general, the information produced by this logging is intended for Sybase Technical Support to aid in problem determination.

To turn on component debugging:

1 In Jaguar Manager:

   a Set the *com.sybase.ai.debug* property to a value of **true**.

   b Refresh the component.

2 Run the component using your Jaguar client application.

3 Check the Jaguar server log to view the debug output produced by component execution.

Client Application Development

If you are familiar with Jaguar, writing client applications that use Application Integrator components is a straightforward task. Because an Application Integrator component is defined in Jaguar as are other Java components, the process of writing client applications will be similar to writing applications for non-Application Integrator components.

For example, building client applications in Jaguar primarily consists of the following steps:

1 Generate the stub classes.

2 Write the application logic. To write the application logic, you must do the following:

   a Instantiate proxy instances.

   b Execute component methods.

   c Process results.

   d Handle errors.

3 Deploy the application.

   See the *Jaguar CTS Programmer's Guide* for details on how to perform these tasks.

Note All IDL modules for Application Integrator components are defined with the module property *com.sybase.jaguar.module.java.package* set to the Java package name that was specified for this component in the Component Builder. Therefore, this will be the Java package name for Java stubs generated for Application Integrator components.

When you write client applications in Jaguar, be sure you understand the following characteristics of Application Integrator components:

- Components can contain one or more methods.
- Each method corresponds to a target COBOL program.
- The input and output parameters for each method are mapped to parameters in the program.
- The return type of the method is based on whether result sets are returned from the program (in the form of an **OCCURS** clause). The number of result sets maps to IDL statements as follows:

Number of result sets	IDL mapping
0	void
1	TabularResults::ResultSet
Variable (0, 1, or many)	TabularResults::ResultSets

- Exceptions caught by the component are propagated to the client stub as a user-defined exception type of *AIUserException*. These exceptions contain a single field called *Message* that contains the error message text for the exception.

Glossary

adapter
A set of Application Integrator classes that are also referred to as the Application Integrator run-time classes. Internally, the Adapter provides access to CICS programs; it also provides security, transaction control, datatype conversion, and error handling integration.

applet
An application program, written in the Java programming language, that can be retrieved from a Web server and executed by a Web browser. A reference to an applet appears in the markup for a Web page in the same way that a reference to a graphics file appears: a browser retrieves an applet in the same way that it retrieves a graphics file. For security reasons, an applet's access rights are limited in two ways: the applet cannot access the file system of the client upon which it is executing; and the applet's communication across the network is limited to the server from which it was downloaded.

builder
See *Component Builder*.

byte code
The output of a Java compiler. Byte code is suitable for interpretive execution by a Java Virtual Machine.

catalog schema
A persistent object in the database that consists of the collection of objects associated with a particular schema name and user authorization identifier. The objects include tables, views, domains, constraints, assertions, and privileges.

catch
Java code that begins error handling logic in exception handling. *See also throw*.

child process
In the UNIX operating system, a process started by a parent process that shares the resources of the parent process.

class
In object-oriented programming, a model or template that can be instantiated to create objects with a common definition and therefore, common properties, operations and behavior.

An object is an instance of a class.

code page
An assignment of graphic characters and control function meanings to all code points.

code set Characters in a code page.

COMMAREA Common communications area where mainframe application logic and presentation logic that are stored in separate regions can communicate with each other.

Common Object Request Broker Architecture (CORBA) Specification produced by the Object Management Group (OMG) that presents standards for various types of object request brokers. Implementation of CORBA standards allows object request brokers from different software vendors to interoperate.

component In Jaguar, an "application object" that consists of one or more methods. Jaguar components typically execute business logic, access data sources, and return results to the client. Clients (Java applications) create an instance of a component and execute methods associated with that component.

Application Integrator creates a component that can contain one or more methods for use in a client application.

Component Builder Application Integrator graphical design tool used to build component and connection information into a component.

connection Network between two systems:

- For SNA, the path connects an LU on one machine to an LU on another machine.
- For TCP/IP, the path connects TCP modules on separate machines.

In Application Integrator, the user creates a connection as part of the component.

container Visual user-interface component that holds objects.

copybook Library of CICS code fragments that usually contain common data definitions and file descriptions. Copybooks are language-specific.

datatype Characteristics of stored information on a computer.

deploy Act of sending components to target container, such as Jaguar CTS and PowerDynamo.

deployment target Deployment container, such as Jaguar CTS and PowerDynamo.

design time Environment in which the Component Builder is used to build connections and components for deployment into Jaguar CTS. *Compare with run time.*

exception Abnormal condition, such as an unknown communications error.

execution time	*See run time*.
host	Mainframe or other machine on which a database, an application, or a program resides. In TCP/IP, this is any system that is associated with at least one Internet address. See also *Transmission Control Protocol*.
hypertext	Way of presenting information online with connections (hypertext links) between one piece of information and another.
import	Process of moving a copybook from its data source into the Application Integrator tool.
instance	In object-oriented programming, an object created by instantiating a class.
instantiate	In object-oriented programming, to represent a class abstraction with a concrete instance of the class. A way to replicate a class file into several new components.
Interface Definition Language (IDL)	In CORBA, a declarative language that is used to describe object interfaces without regard to object implementation.
Jaguar CTS	Jaguar Component Transaction Server.
jar file	File composed of other files, such as class files and serialized files, that acts as a repository or library of classes.
Java	An object-oriented programming language for portable interpretive code that supports interaction among remote objects. Java was developed and specified by Sun Microsystems, Inc.
JavaBeans	The platform-independent component architecture for the Java programming language. JavaBeans enables software developers to assemble pieces of Java code in a drag-and-drop development environment.
Java Database Connectivity (JDBC)	Application programming interface that has the same characteristics as Open Database Connectivity (ODBC) but is specifically designed for use by Java database applications. For databases that do not have a JDBC driver, JDBC includes a JDBC-to-ODBC bridge.
JavaScript	Scripting language that resembles Java and was developed by Netscape for use with the Netscape browser.
Java Virtual Machine (JVM)	An execution time environment for running Java programs. JVMs may be stand alone or embedded in Web browsers, transaction servers, database management systems, and so on.

jConnect	Sybase JDBC driver that is 100% Java and can be used by applets. jConnect can also be used with Sun and Microsoft Java virtual machines.
marshal	To copy data into a form suitable for use by another object. Stubs perform marshalling.
method	In object-oriented programming, software that implements the behavior specified by an operation.
null indicator	A boolean value (**true/false**) that indicates whether the first parameter value of a method is **NULL**.
object	In object-oriented programming, a concrete realization of a class that consists of data and the operations associated with that data.
object request broker (ORB)	In object-oriented programming, software that serves as an intermediary by transparently enabling objects to exchange requests and responses.
Open Database Connectivity (ODBC)	A standard application programming (API) interface for accessing data in both relational and nonrelational database management systems. Using this API, database applications can access data stored in database management systems on a variety of computers even if each database management system uses a different data storage format and programming interface. ODBC is based on the Call Level Interface specification of the X/Open SQL Access Group and was developed by DEC, Lotus, Microsoft and Sybase.
Open ServerConnect	Sybase product that provides capability for programmatic access to mainframe data. It allows workstation-based clients to execute customer-written mainframe transactions remotely.
package	In Jaguar, a collection of components that work together to provide a service or some aspect of an application's business logic. Each package acts as a unit of distribution, grouping together application resources for ease of deployment and management.
package name	In Application Integrator, name used to provide unique identifiers for components and associate them with a specific piece of an application program. In the absence of a package name, Application Integrator assigns a default name.
partner certification reports	Sybase publications that certify third-party query and development tools to work with Sybase products.
port number	In the Internet suite of protocols, the identifier for a logical connector between an application entity and the transport service.

project file	Files that store definitions and information about logically-related components and connections.
proxy	A local representation of a remote user object in a distributed application.
remote stored procedure (RSP)	A customer-written CICS program that resides on the mainframe and communicates with MainframeConnect for DB2/MVS-CICS.
result set	Rows of data retrieved from the database when a SQL **SELECT** statement is executed. The return value for a method call.
run time	Instant at which a particular computer program executes. In Application Integrator, the environment in which a client application tries to invoke a component in Jaguar CTS. Synonymous with *execution time*. *Compare with design time*.
scalable	Pertaining to the capability of a system to adapt readily to a greater or lesser intensity of use, volume, or demand.
schema	Set of statements, expressed in a data definition language, that completely describe the structure of a database.
server	A functional unit that provides shared services to one or more clients or workstations over a network.
skeleton	A *skeleton* acts as the interface between the Jaguar run-time environment and the user code that implements the method. Skeletons are compiled and linked with each of the components, and at run time they enable Jaguar to locate and invoke an appropriate method.
stateful	In code set conversion, the interpretation of a byte depends on the bytes before it. In Application Integrator, component instances that remain dedicated to a single client application while the connection to the server remains open until the component is deactivated. *Compare with stateless*.
stateless	Application Integrator component instances that are removed from the instance pool and dedicated to a client application only for the duration of the method call. *Compare with stateful*.
stub	Small module called from an application that performs marshalling and transfer of control to a larger body of related code.

	In Jaguar, a *stub* is a Java class generated by Jaguar Manager and acts as a proxy object for a Jaguar component. Compiled and linked with a Java client application, a stub communicates with Jaguar to instantiate and invoke a method on a component in the middle tier. Stubs make remote Jaguar components appear local to the client.
Sybase Central	A graphical user interface (GUI) that monitors, configures, and controls databases and related products. Written in Java, it can run on any platform that supports a Java Virtual Machine.
Systems Network Architecture (SNA)	A network plan for transmitting information units through networks and controlling network configuration and operation.
target	Source that contains the data definitions used to create an Application Integrator component.
throw	Java code that deals with exception handling, specifically sending the exception that triggers code for error handling. *See also catch*.
Transmission Control Protocol (TCP)	A communications protocol used in the Internet. TCP uses Internet Protocol (IP) as the underlying protocol.
Transaction Router Service (TRS)	DirectConnect program that accepts requests from workstation-based clients and routes them to Open ServerConnect.
Web browser	A client program that initiates requests to a Web server and displays the information that the server returns.
Web server	A server that is connected to the internet and is capable of serving Web pages.
wizard	A series of dialog boxes within an application that guide a user through the completion of a task.

Index

D

E

F

G

I

J

M

N

O

P

R

S

T

U

V

W

Z