Developer's Guide

iPlanet™ ECXpert

Version 3.6

December 2001

Contents

List of Figures

List of Tables

About this Book

This *Guide* describes the concepts, interface and underlying data organization of the ECXpert Software Developer's Kit and the Java Native Interface API Class Library.

This Preface discusses the intended audience, the organization of the *Guide*, and provides a listing of typographic conventions used in this document. If you spend a few minutes looking through the Preface before reading the rest of the *Guide*, you will be able to utilize the *Guide* more effectively.

Before You Begin

You only need to use this manual if you are running command line utilities or you are developing C++ programs that submit files to ECXpert or access the iPlanet ECXpert database.

If you need an overview of iPlanet ECXpert, read the *iPlanet ECXpert Administrator's Guide* first. Then, read "Introducing the ECXpert Software Developer's Kit" on page 21 in this manual to obtain a brief overview of the SDK components. If you are using the API in conjunction with TradingXpert, you should read *iPlanet TradingXpert Installation Guide* as well.

Before you begin, make sure you have the latest version of the *iPlanet ECXpert Release Note*. See the following section for instructions.

Downloading the Latest ECXpert Documentation

We continuously update iPlanet ECXpert documentation. You can download any of the ECXpert documents from:

http://docs.iplanet.com/docs/manuals/ecxpert.html

The ECXpert Documentation Set

You may wish to refer to other iPlanet ECXpert books for additional information. This section discusses each book in the ECXpert documentation set.

Release Note

After you receive the iPlanet ECXpert software, before you do anything else, you should download the *iPlanet ECXpert Release Note*. See "Downloading the Latest ECXpert Documentation" on page 15 for instructions.

The Release Note contains:

- A list of bugs fixed in the current release
- A list of all documentation corrections
- Warnings and workarounds for known problems
- Additional important information you should know before you install or use ECXpert

The Release Note is platform-specific, so make sure you have the right version for the platform you're using.

Installation Guide

The *iPlanet ECXpert Installation Guide* (available for Solaris and Windows NT operating systems) is the book you use to install ECXpert. It includes preinstallation tasks—including basic instructions for installing or upgrading to the required version of Oracle—ECXpert installation steps, and information on additional tasks you may wish to perform after you install ECXpert. Since the *iPlanet ECXpert Installation Guide* is platform-specific, make sure you have the right version for the platform you're using.

Administrator's Guide

The *iPlanet ECXpert Administrator's Guide* is written for the ECXpert System's site administrator. This book provides an overview of the ECXpert system and uses specific examples, or "scenarios," to illustrate the different ways in which ECXpert can be used most effectively in a wide variety of different business situations. It also covers the ECXpert System Administration Interface in depth, discusses the ECXpert commandline utilities, and explains how to integrate ECXpert with Oracle Financials, SAP, and MQSeries.

Operations Reference Guide

If you ever have difficulty using ECXpert, the *iPlanet ECXpert Operations Reference Guide* more than likely documents a quick resolution. This book contains basic troubleshooting guidelines for ECXpert, other iPlanet products, and Third-party products. It also includes a complete error message reference.

Audience

This manual is written for both a system administrator and developers. For example, the system administrator may wish to run the included command line utility programs. The developer will access the ECXpert SDK and perhaps the Java API class library information.

Organization

This manual is divided into the following chapters and appendices:

- Chapter 1, "Introducing the ECXpert Software Developer's Kit," identifies the command lines utilities and classes in the SDK. It also introduces custom services.
- Chapter 2, "Creating a Custom Service," describes how to create a custom service.
- Chapter 3, "Creating a User-defined Communications Service," describes how to write a program or script that you want to install as a user-defined communications service.

- Chapter 4, “Using the NAS ECXpert Submit Extension,” describes the NAS ECXpert extension and explains how to use it.
- Chapter 5, “The ECXpert XML SDK,” describes the the ECXpert XML software developer kit (SDK).
- Chapter 6, “The EcxBase Class,”, "The EcxBase Class", describes the base class for classes in the SDK.
- Chapter 7, “The EcxInit Class,” describes a class for initializing other objects.
- Chapter 8, “The EcxSubmit Class,” describes a class for submitting files to ECXpert for processing.
- Chapter 9, “The EcxLogin Class,” describes a class that represents a logged-in user.
- Chapter 10, “The EcxAddresses Class,” describes a class that represents member address records in an ECXpert database.
- Chapter 11, “Partnership-Related Classes,” describes a class that represents partnership-related records.
- Chapter 12, “The EcxMember Class,” describes a class that represents member records in an ECXpert database.
- Chapter 13, “Document-Related Classes,” describes a class that represents records in an ECXpert database for documents sent to the logged-in user via iPlanet ECXpert.
- Chapter 14, “The EcxTracking Class,” describes a class that represents records in an ECXpert database for documents sent from the logged-in user via iPlanet ECXpert.
- Chapter 15, “The EcxLog Class,” describes a class that represents log records in an ECXpert database.
- Chapter 16, “The EcxFtpClient Class,” describes a class that is an FTP client API which allows you to send and receive files via FTP.
- Chapter 17, “The EcxService Class,” describes a class that represents service records in an ECXpert database.
- Chapter 18, “The EcxServiceList Class,” describes a class that represents service list records in an ECXpert database.
- Chapter 19, “The ECXpert Java API Classes,” describes the classes comprising the Java API along with environment variables and implementation examples.

- Chapter 20, "Specifying Billing Data to be Captured by the Billing Utility," describes the guidelines used to capture billing data from the ECXpert database.
- Chapter 21, "Customizing Reports," describes how to use the Actuate Reporting System to create custom reports that access the ECXpert database.
- Appendix A, "ECXpert Database Schema," details the table structure of the database underlying the ECXpert System.

Conventions

A number of typographic conventions are used throughout this manual to help you recognize special terms and instructions. These conventions are summarized in the table below.

Convention	Meaning	Example
boldface	items on the screen	Click the **Submit** button to save your changes.
	names of keys	Press **Enter** to clear the message.
boldface numbered steps	higher level descriptions of tasks you perform (more detailed instructions follow)	1. Enter the group information. Enter the name in the **Group Name** field, and a short description in the **Description** field.
italics	key words, such as terms that are defined in the text	The notices posted on an electronic BBS are called *articles*.
	names of books	For more information, refer to the *iPlanet ECXpert Installation Guide*.
`courier font`	command line input or output	Enter the following command: `ls *.html`
	text file content, such as HTML templates and configuration files	`<TITLE>Password Check</TITLE>` `<IMG SRC="/ui/icons/hd_svcs.gif">`
	code samples	const char* getName() const;

Chapter 1

Introducing the ECXpert Software Developer's Kit

This chapter provides a description of the software development kit for iPlanet ECXpert. This description provides an overview of the command line utilities, custom services, and API.

This chapter contains the following sections:

Overview

The ECXpert Software Development Kit consists of the following parts:

- a protocol for implementing custom services
- a C++ API for accessing the database and for submitting files to ECXpert
- a Java API for extending application interface for accessing the database and file submission functions.

The following sections introduce these topics.

Command Line Utilities

ECXpert provides programs that you can embed in scripts or run from the command line. These programs allow you to bypass the user interface when importing records into the database and allow you to send documents to ECXpert and check for recently arrived documents. The following table lists the command line utilities.

Table 1-1 Command line utilities in ECXpert

Utility	Use
`submit`	Submit files to ECXpert
`poll`	Check for new documents
`import`	Import records for Members, Partnerships, or Service Lists
`importCertificate`	Import security certificates
`bdggenManifest and bdgrealpurge`	Purge aged data

Custom Services

A custom service is an application or program that is called by ECXpert to perform a specific task, such as moving a document to a directory outside of ECXpert's control, sending e-mail to a user when a document is sent or received, or preprocessing or translating a file in a custom way.

The chapter "Creating a Custom Service" on page 35 specifies language requirements and calling conventions for implementing a custom service. The chapter also includes examples, which are written in Perl.

User-Defined Communications Service

A user-defined communications service is an application or program that is called by ECXpert to deliver files after ECXpert has finished processing them.

The chapter "Creating a User-defined Communications Service" on page 49 explains how to implement a user-defined communications service.

Implementing Capture of Billing Data

Billing data can be collected and presented an XML file to determine the number of transactions that occur within a predefined billing period for particular sender, receiver, and document types.

The chapter "Specifying Billing Data to be Captured by the Billing Utility" on page 327 describes the billing data DTD and the perl script utility used to generate the XML data file.

ECXpert API

The iPlanet ECXpert API allows you to manipulate the database outside of iPlanet ECXpert. You can manipulate database records in the following ways:

- add, retrieve, delete, and update membership records
- add, retrieve, and delete address records
- add, retrieve, and change partnership-related records
- retrieve document records
- retrieve tracking records
- add log records

In addition, the API allows you to submit files for processing by iPlanet ECXpert.

The API is available for C++. The chapters that describe the classes in the API show C++ syntax and examples.

Use the SparkWorks C++ compiler, version 4.1 or higher, to compile the customer programs using ECXpert SDK shared libraries. If the compiler version is higher than SparkWorks C++ version 4.1 compiler, please use compiler compatibity option as 'compat=4' when compiling code.

Notes Regarding Legacy Versions of the ECXpert SDK

NOTE Changes to the ECXpert 2.0 SDK have made the classes noted below backwards incompatible.

- EcxDocument
- EcxLog
- EcxPartnership
- EcxSubmit
- EcxTracking

You must to rewrite any code you have written with these classes to reflect the changes that have been made since the ECXpert 1.1.1 SDK. If you do not rewrite your code, it will not compile.

Additionally, the following classes were added with the ECXpert 2.0 SDK:

- EcxFtpClient()
- EcxService()
- EcxServiceList()

Special LDAP Entry in ecx.ini File

If you have LDAP enabled with ECXpert, in the [LDAP] section of the *ecx.ini* file, set the cn parameter to ECX before you start using the ECXpert API. No harm is done if you fail to do this, but some false error messages may appear when listing members using the SDK API.

Class Library

Most classes descend from the `EcxBase` class, which defines the error-handling that is available in the class library.

The following table provides a brief description of the classes:

Table 1-2 ECXpert API Class Descriptions

Class	Defines	Page No.
`EcxAddresses`	Trading address records.	page 131
`EcxBase`	Base class for SDK.	page 105
`EcxDocID`	A document by its key.	page 225
`EcxDocument`	Documents sent to iPlanet ECXpert.	page 209
`EcxFtpClient`	FTP Client API to send and receive documents via FTP.	page 253
`EcxInit`	An initialization object.	page 109
`EcxLog`	Log records.	page 243
`EcxLogin`	User-login objects.	page 127
`EcxMember`	Membership records.	page 187
`EcxPartnerID`	A partnership by its key.	page 183
`EcxPartnership`	Partnership view-related records.	page 139
`EcxService`	Service records.	page 263
`EcxServiceList`	Service list records.	page 277
`EcxSubmit`	Submission objects.	page 113
`EcxTracking`	Documents sent from iPlanet ECXpert.	page 229

Relationship Between Objects and Database Records

The following table shows classes whose objects each represent a record (or records) in a database table (or tables):

Class	Record in Table	Page No.
`EcxAddresses`	MBAddresses	page 386
`EcxDocument`	TrkDoc	page 414
`EcxLog`	EventLog	page 424
`EcxMember`	Members	page 384
`EcxPartnership`	Partnership view-related records (real table names)	page 387
	Partnership by its key	page 389
	PNDocs	page 400
	PNGroup	page 401
	PNStd	
`EcxTracking`	TrkDoc	page 414

Note that objects of the `EcxDocument` and `EcxTracking` classes represent the same kind of records. Objects of the `EcxDocument` class represent documents that have been sent to iPlanet ECXpert. Objects of the `EcxTracking` class represent documents that you have sent using iPlanet ECXpert.

Database Access

Before you can use an object of any SDK class, you must create a single `EcxInit` object. Typically, you create the `EcxInit` object in your program's `main()` function.

To access a record in the database or to add a record to the database, you must

1. create an object that corresponds to the kind of record you want to manipulate and
2. associate the object with an `EcxLogin` object.

The `EcxLogin` object specifies the user who is allowed to access the record. In most cases, only users who are also administrators can add, change, or delete records. Non-administrators can retrieve their own records; administrators can retrieve any record.

When you access an object's field, you are only accessing the in-memory value for the field. The record in the database remains unchanged.

Using Lists

Most classes provide a `List()` method that you can use to retrieve records that match a specific criteria. When you call the `List()` method, the first record that matches the criteria is associated with the object and the record's fields populate the object. You call the `Next()` method to retrieve the next record in the list; the newly retrieved record's fields replace the previous values in the object. You call the `More()` method to determine if there are more records in the list and, if desired, to count the remaining records. You can call the `Clear()` method to reset the list. Calling the `Clear()` method also disassociates the object with all records.

Error Handling

Methods that access the database may set result codes that you can access by calling the object's `Errnum()` method. You can also call the object's `ErrMsg()` method to determine and, perhaps, display the cause. The following codes are defined by the SDK:

Result	Value	Description
`noError`	0	No error occurred.
`unknownError`	1	An unspecified error occurred.
`logicError`	2	Internal error.
`notImplemented`	3	Internal error.
`invalidArgument`	4	A required argument is missing or an argument contains invalid data or improperly formatted data.
`outOfMem`	5	Insufficient memory to create an object.
`argumentOutOfRange`	6	An argument contains data that is not within the allowable range of values.

Result	Value	Description
`uninitializedData`	12	An object has not been completely set up. For example, this error occurs if you attempt to use an `EcxLogin` object that is not associated with a valid user.
`invalidValue`	13	Invalid value.
`invalidData`	17	Invalid date.
`notFoundErr`	21	Record not found.
`invalidRequest`	22	An action was attempted for which you do not have permission; for example, when a non-administrator attempts an action that can only be performed by an administrator.
`missingData`	27	Missing data.
`securityException`	60000	An action was attempted for which you do not have permission; for example, when a non-administrator attempts an action that can only be performed by an administrator.
`invalidLogin`	60001	Invalid login.

In addition to the error codes defined by the SDK, additional errors codes can be returned from the underlying database access functions. Database error codes are in the range of 501 to 606, inclusive.

Oracle Warnings When Compiling the ECXpert SDK

If you are using an earlier ECXpert version with Oracle8, release 8.0.4 or Oracle7, release 7.3.3.5, you will see a series of warning messages when you compile the ECXpert SDK. These warning messages appear to have no affect on the resulting compiled executable.

NOTE If you are using Oracle7, release 7.3.4, you will not see these warning messages.

If you are using Oracle8, release 8.0.4, when you compile the ECXpert SDK you will see a series of warning messages, of which the first three should be similar to the following:

```
ld: warning: symbol ‘osnttc’ has differing sizes:
(file /export2/actraadm/NS-apps/ECXpert/lib/libecxsdkdb10.so    value=0x4c;
file
/export2/oracle/product/8.0.4/lib/libclntsh.so value=0x74);
/export2/actraadm/NS-apps/ECXpert/lib/libecxsdkdb10.so definition    taken
ld: warning: symbol ‘nstrcarray’ has differing sizes:
(file /export2/actraadm/NS-apps/ECXpert/lib/libecxsdkdb10.so    value=0xce4;
file
/export2/oracle/product/8.0.4/lib/libclntsh.so value=0xde0);
/export2/actraadm/NS-apps/ECXpert/lib/libecxsdkdb10.so definition    taken
ld: warning: symbol ‘nnfgtable’ has differing sizes:
```

If you are using Oracle7, release 7.3.3.5, when you compile the ECXpert SDK you will see a series of warning messages, of which the first three should be similar to the following:

```
ld: warning: symbol ‘nnfgtable’ has differing sizes:
(file /disk1/actraadm/install1/NS-apps/ECXpert/lib/libecxsdkdb10.so value=0x30;
file /disk1/oracle7/wg7322/lib/libclntsh.so value=0x40);
/disk1/actraadm/install1/NS-apps/ECXpert/lib/libecxsdkdb10.so definition taken
ld: warning: symbol ‘nls_global_lock’ has differing sizes:
(file /disk1/actraadm/install1/NS-apps/ECXpert/lib/libecxsdkdb10.so value=0x20;
file /disk1/oracle7/wg7322/lib/libclntsh.so value=0x28);
/disk1/oracle7/wg7322/lib/libclntsh.so definition taken
ld: warning: symbol ‘nlstdgbl’ has differing sizes:
(file /disk1/actraadm/install1/NS-apps/ECXpert/lib/libecxsdkdb10.so value=0x148;
file /disk1/oracle7/wg7322/lib/libclntsh.so value=0x178);
/disk1/oracle7/wg7322/lib/libclntsh.so definition taken
```

ECXpert Java API

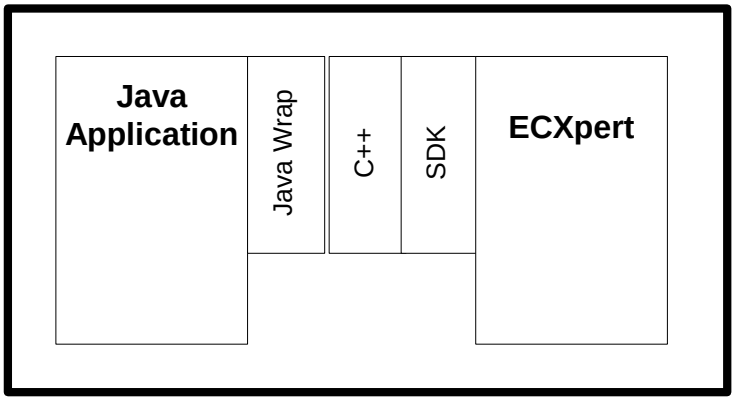
The ECXpert Java API contains a set of Java classes that correspond to ECXpert objects, such as member, address, partnership, document, tracking, log, service, service list, and submission. This Java API is a JNI (Java Native Interface) wrapper which provides a developer the library to design and implement a Java application that utilizes the classes contained in the traditional ECXpert software developer’s kit written in C++.

The ECXpert Java API can be used with an application server, a RMI and a standalone Java application. The use of the ECXpert Java API in servlets is not recommended.

NOTE Running the ECXpert Java API requires the JDK, version 1.1.6 or higher.

The interaction between the Java API with ECXpert is shown in Figure 1-1 below.

Figure 1-1 Java API Interaction with ECXpert



The individual classes in the Java API have almost a one-to-one mapping to the classes and methods in the ECXpert SDK.

Java API Class Library

Most classes descend from the `JEcxBase` class, which defines the error-handling that is available in the class library.

The following table provides a brief description of the classes:

Table 1-3 Java API Classes

Class	Defines	Page No.
`JEcxAddresses`	Trading address records.	page 304
`JEcxBase`	Base class for SDK.	page 295
`JEcxDocID`	A document by its key.	page 315
`JEcxDocument`	Documents sent to iPlanet ECXpert.	page 312
`JEcxFtpClient`	FTP Client API to send and receive documents via FTP	page 319
`JEcxInit`	An initialization object.	page 296
`JEcxLog`	Log records.	page 318
`JEcxLogin`	User-login objects.	page 299
`JEcxMember`	Membership records	page 300
`JEcxPartnerID`	A partnership by its key.	page 311
`JEcxPartnership`	Partnership view-related records.	page 305
`JEcxService`	Service records.	page 320
`JEcxServiceList`	Service list records.	page 322
`JEcxSubmit`	Submission objects.	page 297
`JEcxTracking`	Documents sent from iPlanet ECXpert.	page 316
`JEcxFTPClient`	Methods to send and receive files by FTP	page 319

Through the Java API, most of the ECXpert functionality is exposed to any developer who wishes to design applications using ECXpert as a platform. For example, you can administer user and partnership profiles, define services and service lists, submit documents into ECXpert, and track its workflow.

Relationship Between Java Objects and Database Records

The following table shows classes whose objects each represent a record (or records) in a database table (or tables):

Table 1-4 Relationship Between Java Objects and Database Records

Class	Record in Table	Page No.
`JEcxAddresses`	MBAddresses	page 386
`JEcxDocument`	TrkDoc	page 414
`JEcxLog`	EventLog	page 424
`JEcxMember`	Members	page 384
`JEcxPartnership`	Partnership view-related records (real table names)	page 387
`JEcxPartnerID`	Partnership by its key	page 387
`JEcxTracking`	TrkDoc	page 414

Note that objects of the `JEcxDocument` and `JEcxTracking` classes represent the same kind of records. Objects of the `JEcxDocument` class represent documents that have been sent to iPlanet ECXpert. Objects of the `JEcxTracking` class represent documents that you have sent using iPlanet ECXpert.

Database Access

Before you can use an object of any Java API class, you must create a single `JEcxInit` object. Typically, you create the `JEcxInit` object in your program's `main()` function.

To access a record in the database or to add a record to the database, you must

1. create an object that corresponds to the kind of record you want to manipulate and
2. associate the object with an `JEcxLogin` object.

The `JEcxLogin` object specifies the user who is allowed to access the record. In most cases, only users who are also administrators can add, change, or delete records. Non-administrators can retrieve their own records; administrators can retrieve any record.

When you access an object's field, you are only accessing the in-memory value for the field. The record in the database remains unchanged.

Using Lists

Most classes provide a list() method that you can use to retrieve records that match a specific criteria. When you call the `list()` method, the first record that matches the criteria is associated with the object and the record's fields populate the object. You call the `next()` method to retrieve the next record in the list; the newly retrieved record's fields replace the previous values in the object. You call the `more()` method to determine if there are more records in the list and, if desired, to count the remaining records. You can call the `clear()` method to reset the list. Calling the `clear()` method also disassociates the object with all records.

Error Handling for Java API Class Methods

For all mutator (set), accessor (get), and action (list, more, next, add, delete, change) methods used in the Java API, an exception is thrown when an error is encountered. This represents a departure from the traditional error processing model in the C++ SDK. Instead of using `Errnum()` to check the success of individual operation, statements should be placed within a "try, catch" block. Appropriate use of the printStack`Trace()` and `getMessage()` methods of java.lang.Throwable (which is inherited by Exception) will produce verbose error information from ECXpert. In addition, users can still get error numbers and error messages using `getErrnum()` and `getErrmsg()` methods defined in `JEcxBase()` class.

Custom Reports

ECXpert includes the Actuate Reporting System, which you can use create custom reports. These reports access the ECXpert database directly using the Select statement you specify to select records for your report. For information about how to create custom reports, see "Customizing Reports" on page 331. For information about the database schema that you use to specify the selection criteria, see "ECXpert Database Schema" on page 369.

Chapter 2

Creating a Custom Service

This chapter describes how to write a program or script that you want to install as a custom service. The following topics are covered:

Overview

A service list may include custom services. A **custom service** is an application or program that is called by ECXpert to perform a specific task. Examples of these tasks include:

- moving a document to a directory outside of ECXpert's control
- sending e-mail to a user when a document is sent or received
- preprocessing or translating a file in a custom way

The following sections specify languages that you can use to implement a custom service, the conventions that ECXpert follows to call your service, and the conventions your service must follow when it returns. Several examples, written in Perl, are provided to show how your program can receive and use parameters passed to your service from ECXpert.

Language Requirements

Most custom services are written in compiled languages, such as C or C++, or in scripting languages, such as csh, sh, or Perl. You can use any language that has the following capabilities:

- accepts arguments from the command line
- supports file I/O

Because many languages provide these capabilities, your choice of the language is most likely determined by the language's suitablity to the task, its ease of use, and site standards.

Windows NT Under Windows NT 4.0, the custom service may not be a batch file. A simple workaround is to use a Perl script. This is not an ECXpert limitation; NT 4.0 does not allow a background process like the ECXpert Dispatcher to start up an executable that opens a foreground window. Starting up a batch file momentarily opens a DOS window.

Call and Return Conventions

A program that implements a custom service must follow ECXpert's conventions for argument passing when the program is invoked. It must also follow ECXpert's conventions for returning from the program on termination.

When ECXpert calls a custom service it passes three arguments to the service. The first argument is the full path name of a file that contains parameters that control the operation of the service. This file is called the **parameter -specification file**. The second argument is the full path name of a file that contains the files on which the service executes. This file is called the **data-specification file**. The third argument is the full path name of a file that contains data to be passed from each custom service in a service list to subsequent custom services in the same service list. This file is called the **custom parameter file**. The parameter-specification and data-specification files are discussed in the following sections.

When the service returns, it must return a value of 0 if it performed all operations successfully. The service may return any non-zero value to indicate that one or more operations did not succeed.

CAUTION If your custom service returns a non-zero value, ECXpert stops processing the service list. You can view the status of the service and the service list by checking the Event log in Activity Tracking.

The Parameter-specification File

When ECXpert calls your custom service, it passes the service a parameter-specification file as the service's first argument. This file contains the parameters that may be used by the service. These parameters include the sender and receiver's member IDs, the file type and path name of the document file. Each parameter is identified by a two-letter keyword. Table 2-1 shows the keywords and their descriptions

Table 2-1 Parameter-specification File Keyword and Usage Description

Keyword	Usage
SE	Member ID of the sender.
RE	Member ID of the sender recipient.
FN	The full path name of the file.
FT	The type of the file.
TI	The file's tracking ID.

The keyword is separated from the parameter's value by an equal sign. Only one keyword-value pair can appear on a single line in the parameter-specification file. Keyword-value pairs can appear in any order within the file. Your service must be able to handle all pairs, in any order, even if your program just ignores the parameter. It must also be robust enough to handle missing keyword-value pairs.

The following example shows the contents of a parameter-specification file.

```
TI=20
SE=Dante
RE=Dash
FN=/export/home/actraadm/actra-home/Actra-apps/ECXpert/tmp/track/    trk20
FT=EDI
```

In this example, the tracking ID is 20, the sender is Dante, the receiver is Dash, the file name is `/export/home/actraadm/actra-home/Actra-apps/ECXpert/tmp/track/trk20`, and the file type is EDI.

The Data-specification File

The data-specification file contains the files that ECXpert generates as part of its translation process. For example, the data-specification file may contain files such as these:

```
/export/home/actraadm/.../Actra-apps/ECXpert/data/output/20-1-1-2.    out2
/export/home/actraadm/.../Actra-apps/ECXpert/data/output/20-1-1-2.    out3
/export/home/actraadm/.../Actra-apps/ECXpert/data/output/20-1-1-1.    out2
...
/export/home/actraadm/.../Actra-apps/ECXpert/data/output/20-1-1-8.    out3
/export/home/actraadm/.../Actra-apps/ECXpert/data/output/20-1-1.997
```

The Custom Parameter File

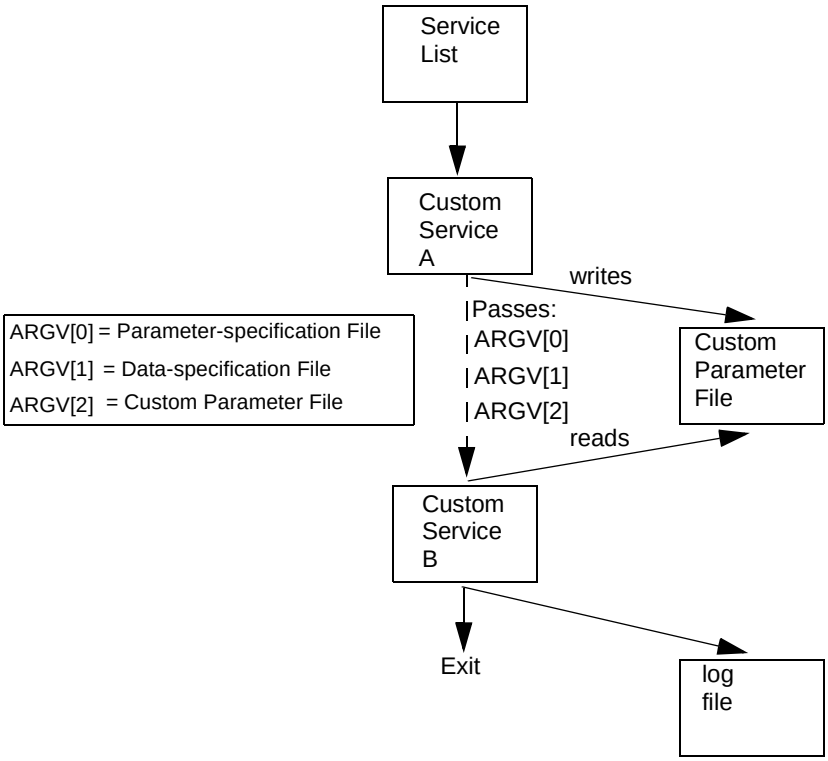
Prior to ECXpert Version 2.0, when ECXpert called a custom service it passed only the parameter-specification file and the data-specification file.

Since Version 2.0, a third argument is passed—the full path name of a file that contains data to be passed from any custom service in a service list to subsequent custom services in the same service list. This file is called the custom parameter file. As the custom parameter file passes through the service list, it can be edited by any custom service in the service list.

The custom parameter file is automatically deleted upon completion of the service list.

Example In Figure 2-1, a service list calls Custom Service A. Custom Service A then passes the three parameters—the Parameter-specification filename, the Data-specification filename, and the Custom Parameter filename—to Custom Service B, and writes information to the Custom Service Parameter File. Custom Service B reads the information from the Custom Service Parameter File, and then exits the service list. When the service list is exited, a log file is created.

Figure 2-1 Custom Parameter File Diagram



Example Write Script Following is an example of a script that writes information to the Custom Parameter File. In the Figure 2-1, this script would be used by Custom Service A to write information to the Custom Parameter File:

```
#!/usr/local/tools/bin/perl
#
# File: customSvr_Write.pl
#
```

```
# Custom service for ECXpert
#
# Description:
#    This program tests out the custom service for ECXpert.  It
#    basically prints out to a log file the parameters received
#    from ECXpert when the custom sevice is invoked.
#
#    In addition to printing out the parameters, it also writes
#    a few lines to the custom service file (argv 3) being
#    passed in by dispatcher.  In v2.0 of ECXpert, it supports
#    a 3rd parameter file to allow passing of information between
#    2 custom services.
#
#
# $logFIle = pathname of the log file for this service to output
#
$logFile = "/tmp/customSvr.dbg";

$paramFile = $ARGV[0];
$dataFile  = $ARGV[1];
$customFile = $ARGV[2];

open(LOGF, ">>$logFile") || die "Can't open log file\n";
print LOGF "--------- Custom_Write Service Start ---------\n";
printTimeToLog();
# print basic file argument
print LOGF "Parameter File (0): $paramFile\n";
print LOGF "Data File      (1):  $dataFile\n";
print LOGF "Custome File   (2): $customFile\n";

# print additional argument if exist
foreach $i (3 .. $#ARGV) {


print LOGF "\n";

print LOGF ">>> Parameter File Content:\n";
printAsciiFile($paramFile);
print LOGF ">>> Data File Content:\n";
printAsciiFile($dataFile);
print LOGF ">>> Custom File Content:\n";
printAsciiFile($customFile);
print LOGF ">>> End File Content\n";

print LOGF "\n";
print LOGF ">>> Writing to Custom File\n";
writeAsciiData($customFile, 1, 2);
print LOGF ">>> Finish Write\n";

print LOGF ">>> New Custom File Content:\n";
printAsciiFile($customFile);
print LOGF ">>> End File Content\n";

print LOGF "--------- Custom_Write Service End ---------\n";
close(LOGF);
0;
```

```
############################
# Subroutines
############################
sub printTimeToLog {
local($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst) =
localtime(time());
$mon += 1;
print LOGF "Time : $mon/$mday/$year $hour:$min:$sec\n";
}


sub printAsciiFile {









}
sub writeAsciiData{




}Output File
```

Example Read Script Following is an example of a script that reads information from the Custom Parameter File. In the Figure 2-1, this script is would be used by Custom Service B to read the Custom Parameter File:

```
#!/usr/local/tools/bin/perl
#
# File: customSvr.pl
#
# Custom service for ECXpert
#
# Description:
#    This program tests out the custom service for ECXpert.  It
#    basically prints outs to a log file the parameters received
#    from ECXpert when the custom service is invoked.
#
# $logFile = pathname of the log file for this service to output
#
$logFile = "/tmp/customSvr.dbg";

$paramFile = $ARGV[0];
$dataFile  = $ARGV[1];
```

```
$customFile = $ARGV[2];

open(LOGF, ">>$logFile") || die "Can't open log file\n";
print LOGF "--------- Custom Service Start ---------\n";
printTimeToLog();

# print basic file argument
print LOGF "Parameter File (0): $paramFile\n";
print LOGF "Data File      (1):  $dataFile\n";
print LOGF "Custome File   (2): $customFile\n";

# print additional argument if exist
foreach $i (3 .. $#ARGV) {


print LOGF "\n";

print LOGF ">>> Parameter File Content:\n";
printAsciiFile($paramFile);
print LOGF ">>> Data File Content:\n";
printAsciiFile($dataFile);
print LOGF ">>> Custom File Content:\n";
printAsciiFile($customFile);
print LOGF ">>> End File Content\n";

print LOGF "--------- Custom Service End ---------\n";
close(LOGF);
0;

############################
# Subroutines
############################
sub printTimeToLog {
local($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst) =
localtime(time());
$mon += 1;
print LOGF "Time : $mon/$mday/$year $hour:$min:$sec\n";
}

sub printAsciiFile {



#




}
```

Example Log File Following is an example of the log file—`/tmp/customSvr.dbg`—that would be created. In Figure 2-1, this file would be generated when the service list is exited.

```
--------- Custom_Write Service Start ---------
Time : 10/23/98 13:5:7
Parameter File (0): /disk1/actraadm/install1/NS-apps/ECXpert/data/work
ENVAAAa006zq-28657-0
Data File      (1):  /disk1/actraadm/install1/NS-apps/ECXpert/data/work
LSTBAAa006zq-28657-0
Custome File   (2): /disk1/actraadm/install1/NS-apps/ECXpert/data/work
ARGCAAa006zq-28657-946

>>> Parameter File Content:
TI=946
SE=rav4
RE=escort
FN=/disk1/actraadm/install1/NS-apps/ECXpert/data/work/trk/trk946
FT=custom
RV=0
>>> Data File Content:
>>> Custom File Content:
>>> End File Content

>>> Writing to Custom File
>>> Finish Write
>>> New Custom File Content:
argument 1 = 1
argument 2 = 2
>>> End File Content
--------- Custom_Write Service End ---------
--------- Custom Service Start ---------
Time : 10/23/98 13:5:8
Parameter File (0): /disk1/actraadm/install1/NS-apps/ECXpert/data/work
ENVDAAa006zr-28657-0
Data File      (1):  /disk1/actraadm/install1/NS-apps/ECXpert/data/work
LSTEAAa006zr-28657-0
Custome File   (2): /disk1/actraadm/install1/NS-apps/ECXpert/data/work
ARGCAAa006zq-28657-946

>>> Parameter File Content:
TI=946
SE=rav4
RE=escort
FN=/disk1/actraadm/install1/NS-apps/ECXpert/data/work/trk/trk946
FT=custom
RV=0
>>> Data File Content:
>>> Custom File Content:
argument 1 = 1
argument 2 = 2
>>> End File Content
--------- Custom Service End ---------
```

Custom Service Examples

Your custom service can be divided into a function that parses command line arguments and functions that perform the logic you want to implement. The following examples show a Perl function that handles the command line and Perl scripts that implement two services, a file-copy service and a submission service.

Parsing Command Line Arguments

The following function parses the command line arguments. The function opens the file specified in the first argument and decodes the keyword arguments. It then opens the file specified in the second argument and creates a list of file names. This function is called by the scripts that implement services.

```
#!/usr/local/bin/perl

# This function is designed to be called by a script acting as
# a service within a service list of ECXpert. It takes two parameters.
# The first parameter is expected to be a filename pointing to a file
# that contains submission information. The second parameter is a
# filename that points to a file containing a list of filenames
# that have been unbundled (possibly via translation) by ECXpert.
#
# The function places the parsed values from the first file into an
# associative array called svcArgs. The keys into the array are:
#
#                sender     - the sender of the document
#                receiver   - the receiver of the document
#                trackingID - the tracking id assigned to this
                    document
#                fileName   - the filename
#                fileType   - the type of the file
#
# The function places the file names from the second argument into
# an array called svcFiles.

sub ServiceArgsParse() {
  local(@argv) = @_;

  # This section of code opens the file pointed to by the first
  # parameter and parses out the information.
  open(META, $argv[0]) || die "\nError opening file $argv[0]\n";

  while (<META>) {
    chop($_);
```

```
  close(META);

  # This section of code opens the file pointed to by the second
  # parameter and places each file as an element in an array.

  open(FILELIST, $argv[1]) || die "\nError opening file $argv[1]\n";

  @svcFiles = <FILELIST>;
  chop(@svcFiles);

  close(FILELIST);
}
1;
```

Note that Perl requires a non-zero return as the last line of a file that is required by, meaning included in, another file.

Implementing a File-copy Service

The following script implements a file-copy service. The contents of files in the data-specification file are appended together and their output is separated by a delimiter. The first argument is not used except for printing the keyword values as the first line of the output file.

```
#!/usr/local/bin/perl

# This script copies files from ECXpert to a directory. It may be
# customized by modifying the following variables:
#
#  $targetDirectory  - full path to the directory where the files should be
copied.
#  $delimeter        - the delimiter to be used between concatenated files
#  $additionalInfo   - if defined, will place the value of the variable as the
first line of the file.

$ACTRA_HOME = “/export/home/actraadm/actra-home”;
$ECX_HOME = “$ACTRA_HOME/Actra-apps/ECXpert”;

require “$ECX_HOME/custom-services/ServiceArgsParse.pl”;

&ServiceArgsParse(@ARGV);

#######################################
```

```
#  begin user customizable variables #
#####################################

$targetDirectory = “/tmp”;
$delimeter = “--ECXpert--”;
$additionalInfo = “<SE>$svcArgs{sender}</SE>
     <RE>$svcArgs{receiver}</RE>”
     “<TI>$svcArgs{trackingID}</TI>”;

###################################
# end user customizable variables #
###################################

$targetFile = $targetDirectory . “/ECX-$svcArgs{trackingID}.dat”;
open(COPYFILE, “>$targetFile”) || die “\nError opening $targetFile\n”;

if ($additionalInfo) {
  print COPYFILE “$additionalInfo\n”;
}

$arrayLength = scalar(@svcFiles);
$i = 0;
foreach $file (@svcFiles) {
  $i++;
  open(EACHFILE, $file) || die “\nError opening $file\n”;
  print COPYFILE <EACHFILE>;
  close(EACHFILE);

  print COPYFILE “$delimeter\n” if ($i < $arrayLength);
}

close(COPYFILE);

exit 0;
```

Note that a custom service must return 0 to indicate that it succeeded.

Implementing a Submission Service

The following section implements a submission service. For example, if a file has been submitted to ECXpert, this custom service resumbmits it, effectively forwarding it to another recipient. In this example, all submissions are resubmitted to member ID “Dart.”

```
#!/usr/local/bin/perl

# This script kicks off another submission using information passed in
# from ECXpert and the variables defined below that should be customized
# for specific recipients:
#
#  $targetRecipient    - member id where the document should be
#   forwarded to

$ACTRA_HOME = "/export/home/actraadm/actra-home";
$ECX_HOME = "$ACTRA_HOME/Actra-apps/ECXpert";

require "$ECX_HOME/custom-services/ServiceArgsParse.pl";

&ServiceArgsParse(@ARGV);

#######################################
#  begin user customizable variables #
#######################################

$targetRecipient = "Dart";

#####################################
#  end user customizable variables #
#####################################

$command = "$ECX_HOME/bin/submit -se $svcArgs{receiver} ";
$command .= "-re $targetRecipient -fn $svcArgs{fileName} ";
$command .= "-ft $svcArgs{fileType} -in $ECX_HOME/config/bdg.ini";

system($command);

exit 0;
```

Chapter 3

Creating a User-defined Communications Service

This chapter describes how to write a program or script that you want to install as a user-defined communications service. The following topics are covered:

Overview

A **user-defined communications service** is an application or program that is called by ECXpert to deliver files after ECXpert has finished processing them. ECXpert provides the following delivery methods for data:

- SMTP
- FTP
- GEIS FTP
- HTTP

You can provide a user-defined communications service to implement other kinds of disposition methods.

Typically, a user-defined communications service operates either on documents that have been bundled into an interchange and are ready for delivery to an external system, or on application data ready to be transmitted to another internal host. Examples of tasks performed by a user-defined communications service include sending files via an in-house file transfer utility or submitting the output from ECXpert into a PeopleSoft system.

You implement a user-defined communications service in two parts:

- Modify the configuration file to specify the location of the executable file, titles for the service and its parameters, and to specify other configuration information. ECXpert uses this specification to allow an administrator to set up the service on the Trading Partnership Protocol screen.
- Write the service using a compiled language, such as C or C++, or a scripting language, such as csh, sh, or Perl. The language must accept arguments from the command line and support file I/O.

Windows NT Under Windows NT 4.0, the user-defined communications service may not be a batch file. A simple workaround is to use a Perl script. This is not an ECXpert limitation; NT 4.0 does not allow a background process like the ECXpert Dispatcher to start up an executable that opens a foreground window. Starting up a batch file momentarily opens a DOS window.

The following sections show you how to modify your configuration file and write the service.

Modifying the Configuration File (ecx.ini)

The *ecx.ini* configuration file defines how ECXpert initiates the communications service. You must set up a user-defined communications section, as discussed in "User-defined communications sections" on page 249. Below is a sample user-defined communications section in the *ecx.ini* file.

```
[user-defined-1]
section_type = network
type = process
cmd_and_args = /var/tmp/CopyToServer.sh
append_data_file = 1
prefix_data_file =
cmd_type = script
operation = send
data_type = Both
is_comm_agent = yes
```

```
internal_name = USER DEFINED 1
visible_name = Copy To Server
parameter_name_1 = Destination Directory:
parameter_name_2 = Destination File Pattern:
parameter_name_3 = User:
parameter_name_4 = Host:
```

In this example, ECXpert calls *CopyToServer* in the */var/tmp/* directory to copy application data. ECXpert appends the bundle file's full path name when it calls the script. Table 3-1 explains each line in the configuration file:

Table 3-1 Description of Each Line in User-Defined section of ecx.ini File

Line	Description
[user-defined-1]	A section name. The default is *user-defined-1*.
section_type=network	Type of section; must be *network*.
type=process	Type of executable; must be *process*.
cmd_and_args=/var/tmp/CopyToServer.sh	Full path to the executable for the user-defined communication service and arguments, entered exactly as you would enter them from the OS command line. In this example, *CopyToServer* is not invoked with arguments other than those passed as parameters. The syntax for this line is as follows: (note that this should appear all on one line in the *ecx.ini* file) *cmd_and_args=<pathname> <static_arguments> <data_filename> <partnership_defined_arguments>* Static arguments are hard-coded arguments, and partnership-defined arguments are arguments you can set up via the partnership pages.
append_data_file=1	Whether to append the name of the data file to the end of the *cmd_and_args* line and the trading partnership parameters. Valid values: *0* (do not append), or *1* (append). In this case, *CopyToServer* expects the data file name to be appended.

Table 3-1 Description of Each Line in User-Defined section of ecx.ini File *(Continued)*

Line	Description
prefix_data_file=	Prefix to add to the file name passed to the user-defined communications service, for example *fname*=usercomm1. The bundle file name will be concatenated with the prefix. In this example, the prefix is specified as usercomm1.
cmd_type = script	Type of command, valid values: *script* (default), or *executable*. In this case, *CopyToServer* is a script.
operation = send	Type of communications operation involved. In this example, the service sends data.
data_type = Both	Format of the data in the bundle. In this example, the service sends data in an both EDI and application-specific format.
is_comm_agent = yes	Whether the protocol can be selected as a communications agent; must be *yes*.
internal_name = USER DEFINED 1	The internal name that identifies the service. **Do not** change this value. If you do, the service will not work.
visible_name = Copy To Server	Title that appears as a Primary Outgoing Protocol on the Trading Partnership Protocol screen. In this example, it is "Copy to Server."
parameter_name_1 = Destination Directory:	Title for the first parameter. In this example, the first parameter specifies the name of the destination directory on the server.
parameter_name_2 = Destination File Pattern:	Title for the second parameter. In this example, the second parameter specifies the destination file pattern on the server.
parameter_name_3 = User:	Title for the third parameter. In this example, the third parameter specifies the name of the user on the server.
parameter_name_4 = Host:	Title for the fourth parameter. In this example, the fourth parameter specifies the server.

The *CopyToServer.sh* script that is executed by this sample user-defined communications service is shown below.

```
#! /bin/sh

#
# Copy the file
#

retval=0
directory=${1}
pattern=${2}
remoteuser=${3}
remotehost=${4}
bundlefile=${5}

suffix="‘echo $bundlefile | sed -e ’s/^.*bndl//’‘"

/bin/rcp ${bundlefile}
${remoteuser}@${remotehost}:${directory}/${pattern}.${suffix} \
             2>> /tmp/edi/id.log

if [ "$?" != "0" ]
then
        retval=‘expr ${retval} +1‘
fi

## done
#
exit ${retval}
```

Important Notes Keep the following in mind when using this example to implement a user-defined communications service:

- Do not add spaces between the variable names and their assignment values. For example, this assignment works:

  `fname=${1}`

  while the one below does not:

  `fname = ${1}`

- Any recipient user must have a file named *.rhosts* (e.g.,*. /u/member2/.rhosts*) containing the following information:

  hostname user

If *actraadm* is the user, quasar is the host, and *member2* is a recipient user for the *CopyToServer.sh* script, then *member2* would need to have a file named */u/member2/.rhosts* containing the following:

```
quasar     actraadm
```

Remember to include a domain suffix with the *hostname* (e.g., *quasar.actracorp.com*) if the recipient's machine is in a different domain.

- ECXpert, when running your script, will source the *.cshrc* file in the remote directory, not in the local directory. It is necessary to have lines similar to the following near the beginning of the *.cshrc* file in the remote directory to ensure proper execution.

```
#
# Generic .cshrc
#

# Set up a basic path here in case the script bombs out
setenv PATH /bin:/sbin:/usr/bin:/usr/sbin

# Set umask
umask 022

# Skip rest of setup if not an interactive shell
if ( $?prompt == 0 ) exit
if ( "$prompt" == "" ) exit
```

Writing a User-defined Communications Service

The user-defined communications service accepts values for the parameters and performs the specified task. The parameters are identified by their position as they are passed to the service. This order is defined as follows:

1. parameters specified in the *cmd_and_args* entry in its section of the configuration file, in the order that they are listed in the entry
2. parameters specified in the configuration file, in order from *parameter_name_1* to *parameter_name_n*, where *n* is the last parameter in its section of the configuration file
3. the bundle file name if the *append_data_file* entry in its section of the configuration file is set to 1

The following shell script is an example of a user-defined communications service. It sets the return value to 0 to indicate success, retrieves the parameters that the administrator specified when setting up the protocol, and performs the copy operation. The parameters are

1. destination file
2. user ID
3. host name
4. full path name of the bundled file

```
#!/bin/sh

#
## Copy the file.
#
retval= 0
fname = ${1}
remoteuser = ${2}
remotehost = ${3}
bundlefile = ${4}
rdist -b -c ${bundlefile} ${remoteuser}@${remotehost}:/tmp/${fname} \
  /tmp/edi/id.log
if [ "$?" != "0" ]
then
  retval=‘expr ${retval} + 1‘
fi
## done.
#
exit ${retval}
```

When the service returns, it must return a value of 0 if it performed all operations successfully. The service may return any non-zero value to indicate that one or more operations did not succeed. If an error occurs, check the Event log; the error number is in the log.

CAUTION If your custom service returns a non-zero value, ECXpert stops processing the service list.

Chapter 4

Using the NAS ECXpert Submit Extension

This chapter describes the NAS ECXpert extension and explains how to use it.

This chapter contains the following sections:

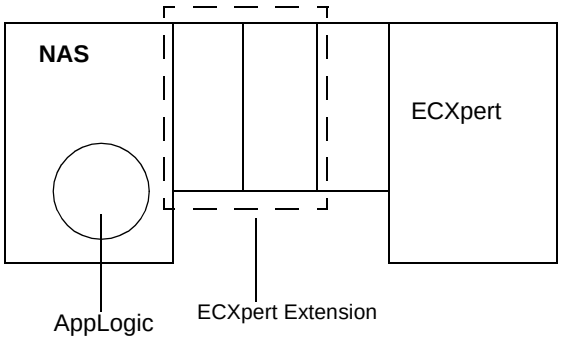
About the NAS ECXpert Extension

The NAS extension of ECXpert contains Java interfaces to ECXpert objects such as member, member address, partnership, document, tracking, log, service, service list, and submission. The extension is written in C++ with a Java wrapper, so that a developer may design Java application logic to directly make use of the extension to interface with the APIs in the ECXpert Software Developer's Kit (SDK).

NOTE An *interface* in Java functions exactly as a *class* in C++.

The interfaces and methods in the extension have an almost one-to-one mapping to the classes and methods in the ECXpert SDK. Through this extension, most of the ECXpert functionalitiy is exposed to any developer who wishes to design applications using ECXpert as a platform. For example, it is possible to use the NAS ECXpert extension to administer user and partnership profiles, define services and service lists, submit documents into ECXpert and track its workflow.

Figure 4-1 Interaction with ECXpert



NOTE The Java classes wrap around the C++ interfaces.

NAS ECXpert Extension Interfaces

The following fourteen NAS ECXpert extension interfaces are available:

IEcxAddress	IEcxMgr
IEcxBase	IEcxPartnerId
IEcxDocID	IEcxPartnership
IEcxDocument	IEcxService
IEcxLog	IEcxServiceList
IEcxLogin	IEcxSubmit
IEcxMember	IEcxTracking

This document explains only the NAS ECXpert submit extension in detail. For more information about other functionality available via the NAS ECXpert API, refer to the *iPlanet TradingXpert Installation Guide*.

Using the NAS ECXpert Submit Extension

The `IEcxSubmit` Interface defines methods that you use to submit a file to ECXpert. You may use these methods to provide a file submission capability within your application instead of requiring the user to execute a command or use ECXpert's HTML interface to submit an object.

You may create objects from the `IEcxSubmit` Interface and use them, directly or you may define a subinterface of the `IEcxSubmit` Interface and create objects from the derived interface. For example, you might define a subinterface that handles much of the application logic associated with files to be submitted to ECXpert. Objects derived from your subinterface would inherit the ability to submit files to ECXpert.

You call methods to specify this information. For example, you call the object's `setSender()` method to specify the sender's member ID. You must specify the files that you wish to submit to ECXpert. You build a submission list by calling the object's `addFile()` method to add a file to the list. You specify the following information when you add a file:

- Document name
- Document type, such as EDIFACT or EDIX12, or a non-EDI type

You may add as many files as you want. If you add more than one file, the files become part of a single multi-part file. When you finish adding the files to the submission list, you may call the object's `Submit()` method to submit the files.

By default, ECXpert moves the files being submitted to the directory specified by the `repository` entry in the configuration file's `tcpip-connector` section. Moving a file (copying the file and deleting the source file after copying) is the most efficient way to submit files if your application executes on the same server as ECXpert.

You may also submit files to ECXpert using a TCP/IP connection. You specify whether or not to use a TCP/IP connection when you call the object's `submit()` method. Using a TCP/IP connection causes ECXpert to stream the contents of the files through a socket to the server. This is a useful technique if your application runs on a remote computer and the files being submitted are relatively small. If you want to submit large files from a remote computer, you should consider using a protocol such as FTP to copy the files to the server and then submit them from the server.

If you wish to submit files to ECXpert from a remote system, you must:

- Edit the ECXpert system's `ecx.ini` file `[tcpip-connector]` section as follows:
  - `port_location=static`
  - `admin_port_type=manual`
  - `admin_port=6001`
  - `listener_port_type=manual`
  - `listener_port=6002`

NOTE For more information on the `ecx.ini` file, see the *iPlanet ECXpert Administrator's Guide*

- Copy the edited `ecx.ini` file to the `/bin` directory in the NAS base directory on the remote machine. This is the same directory where any NAS start-up scripts are located.

NOTE If you stream data through a TCP/IP connection, the source file is not deleted after the data has been streamed to the server.

After you submit a file, you should check for errors. If no error occurred, you may call the object's `getFirstTrackingID()` method to determine the tracking ID of the first file submitted and the object's `getNextTrackingID()` method to determine the tracking ID for each additional file in the list.

CAUTION If the `submit()` method fails, the value returned by calling the `getFirstTrackingID()` or `getNextTrackingID()` method is undefined. When you no longer need references to these files, you may call the object's `clearFileList()` method to remove the files from the list.

Syntax and Methods

Name IEcxSubmit

Syntax `public interface IEcxSubmit extens com.kivasoft.IObject`

Methods Following is a list of methods in the `IEcxSubmit` interface. For additional details about each method, refer to Chapter 8, "The EcxSubmit Class."

Methods
`public int addFile(java.lang.String pFile,java.lang.String pFileType)`
`public int clearFileList()`
`public java.lang.String getDeliveryMethod()`
`public java.lang.String getEcxIniFileName()`
`public int getFirstTrackingID()`
`public java.lang.String getMapName()`
`public java.lang.String getPassword()`
`public java.lang.String getRecipient()`
`public java.lang.String getSender()`
`public int getNextTrackingID()`
`public int setDeliveryMethod(java.lang.String pDeliveryMethod)`
`public int setEcxIniFileName(java.lang.String pIniFileName)`
`public int setMapName(java.lang.String pMapName)`
`public int setPassword(java.lang.String pPassword)`
`public int setRecipient(java.lang.String pRecipient)`
`public int setSender(java.lang.String pSender)`
`public int submit(boolean bDataStreaming)` **Note**: The `bDataStreaming` parameter should be "true" if submitting to a remote ECXpert system.

Example

```
WARNING:  This is a machine generated list, do not modify below
** WizardDictionaryValues={
**   CodeTemplate="/kiva/templates/DoInputWizard.javatmpl",
**   CodeFiles="*.java;Session:SessionAccessorInsert.java",
**   CodeProject="Input",
**   CodeDir="/kiva/APPS/ecx_demo/",
**   CodeLanguage="Java",
**   SessionOut=[
**     "sender"
**     "password"
**     "recipient"
**     "fileName"
**     "fileType"
**     "ecxIniFileName"
**   ],
**   BaseAgent="ecx_demo.BaseAppLogic",
**   CodeWizard="com.kivasoft.wizard.DoInputWizardFactory",
**   CodeFile="/kiva/APPS/ecx_demo/Input.java",
**   Input_filename="/kiva/APPS/DevXpert/web/ecx_demo/index.html",
**   CodeGUID="{588779da-f69c-15e5-e4e3-080020794ab3}",
**   Project="/kiva/APPS/ecx_demo/ecx_demo.gxm",
**   ValIn={com.kivasoft.tools.KSVectorHash
**     ValIn=[
**       "sender"
**       "password"
**       "recipient"
**       "fileName"
**       "fileType"
**       "ecxIniFileName"
**       "remoteSubmission"
**     ],
**     ValIn_NotNull=[
**       "true"
**       "true"
**       "true"
**       "true"
**       "true"
**       "true"
**       "true"
**     ],
**   },
** }
** WARNING:  This is a machine generated list, do not modify above
*/
package ecx_demo;

import java.util.*;

import com.kivasoft.*;
import com.kivasoft.applogic.*;
import com.kivasoft.session.*;
import com.kivasoft.types.*;
```

```
import com.kivasoft.util.*;

import ecx_demo.Session;

import ecx_demo.BaseAppLogic;

import ecx.*;

public class Input extends ecx_demo.BaseAppLogic
{

    public String guid()
    {
        return "{588779da-f69c-15e5-e4e3-080020794ab3}";
    }

    public int execute()
    {
        ecx_demo.Session session = getSessionProxy();

        if (session == null) {
            return result("<HTML>Call to getSessionProxy() failed in
            Input</HTML>");
        }

        //
        // Verify correctness of valIn criteria
        //
        String sender = valIn.getValString("sender");

        if ( null == sender ||
             0 == sender.trim().length() )
        {
            log("Input error on sender");
            return result("<HTML><BODY>sender should not be
               null!</BODY></HTML>");
        }
        String password = valIn.getValString("password");

        if ( null == password ||
             0 == password.trim().length() )
        {
            log("Input error on password");
            return result("<HTML><BODY>password should not be
               null!</BODY></HTML>");
        }
        String recipient = valIn.getValString("recipient");

        if ( null == recipient ||
             0 == recipient.trim().length() )
        {
            log("Input error on recipient");
            return result("<HTML><BODY>recipient should not be
              null!</BODY></HTML>");
        }
```

```
        String fileName = valIn.getValString("fileName");

        if ( null == fileName ||
             0 == fileName.trim().length() )
        {
            log("Input error on fileName");
            return result("<HTML><BODY>fileName should not be
               null!</BODY></HTML>");
        }
        String fileType = valIn.getValString("fileType");

        if ( null == fileType ||
             0 == fileType.trim().length() )
        {
            log("Input error on fileType");
            return result("<HTML><BODY>fileType should not be
              null!</BODY></HTML>");
        }
        String ecxIniFileName =
           valIn.getValString("ecxIniFileName");

        if ( null == ecxIniFileName ||
             0 == ecxIniFileName.trim().length() )
        {
            log("Input error on ecxIniFileName");
            return result("<HTML><BODY>ecxIniFileName should not be
              null!</BODY></HTML>");
        }
        String remoteSubmission =
            valIn.getValString("remoteSubmission");

        if ( null == remoteSubmission ||
             0 == remoteSubmission.trim().length() )
        {
            log("Input error on remoteSubmission");
            return result("<HTML><BODY>remoteSubmission should not be
            null!</BODY></HTML>");
        }

        //
        // Save login criteria into the session.
        //
        session.setsender(valIn.getValString("sender"));
        session.setpassword(valIn.getValString("password"));
        session.setrecipient(valIn.getValString("recipient"));
        session.setfileName(valIn.getValString("fileName"));
        session.setfileType(valIn.getValString("fileType"));
        session.setecxIniFileName(valIn.getValString("ecxIniFileName"));
        session.saveSession();


    // Get the extension
    IEcxMgr ecxMgr = access_cECX.getcECX(context,null,this);
    IEcxSubmit ecxSubmit = ecxMgr.createSubmit();
```

```
    System.out.println("Got the extension...");

    ecxSubmit.setSender(sender);
    ecxSubmit.setRecipient(recipient);
    ecxSubmit.setPassword(password);
    ecxSubmit.addFile(fileName, fileType);
    ecxSubmit.setEcxIniFileName(ecxIniFileName);

    System.out.println("Set all parameters...");

    boolean remote;
    if (remoteSubmission.equals("yes"))
      remote = true;
    else
      remote = false;
    ecxSubmit.submit(remote);

    // Return screens
    if (((IEcxBase)ecxSubmit).errnum() == 0)
    {
        String successString = "Submission successful, the file's ECXpert
tracking ID is " + ecxSubmit.getFirstTrackingID() + ".";
        return streamResult(successString);
    }
    else
    {
        String errorString = "Submission failed, error number " +
((IEcxBase)ecxSubmit).errnum() + ".";
        return streamResult(errorString);
    }

  } // execute

} // class
```

Chapter 5

The ECXpert XML SDK

This chapter describes the ECXpert XML software developer kit (SDK). The following topics are covered:

Overview

The ECXpert XML SDK provides a set of C++ Class APIs for users to build applications communicating with eXML-Connector through XML-formatted messages. The SDK library also includes APIs to allow user applications to listen to a port and/or connect to a (host, port) for message exchanges. There are some samples that illustrate how to build a simple server and client programs. There is also a utility that allows easy submission of document to the eXML-Connector.

The eXML-Connector works as another (generic) communications agent in the ECXpert architecture. In the outbound (with respect to ECXpert) process at the transportation/Gateway stage, the document details are propagated to the eXML-Connector by a NSPkt. The eXML-Connector determines what to do with the document based on the information contained in NSPkt. It translates NSPkt info. into an XML-formatted message (XFM), and passes it to the specified service. This service can reside anywhere on the network, and the eXML-Connector interacts with it using XFM.

In the inbound transaction, any XML-based application can send a submission or service request to ECXpert, based on XFM. This request is intercepted by the eXML-Connector, which passes on the ECXpert internals.

Directory Structure and Source Files

The XML SDK directory (*$NSBASE/NS-apps*/`ECXpert/xmlsdk`) in the ECXpert directory tree includes required libraries, header files, and some sample programs. These are listed in Table 5-1.

Table 5-1 XML SDK File Structure Description

Subdirectory or File	Description of Contents
$NSBASE/NS-apps/ECXpert/bin/xmlsbmt	The utility for use to submit a document to eXML Connector
xmlsdk	The XML SDK root directory
xmlsdk/bin	The SDK binary directory
xmlsdk/config	The XML SDK configuration directory
xmlsdk/config/xmlserver.ini	the configuration file for sample program xmlserver
xmlsdk/example	The XML SDK example directory
xmlsdk/example/Makefile.{platform}	The Makefile sample file
xmlsdk/example/xmlclient.cpp	The xmlclient sample program
xmlsdk/example/xmlserver.cpp	The xmlserver sample program
xmlsdk/example/xmlsubmit.cpp	The xmlsubmit sample program
xmlsdk/include	The XML SDK include directory
xmlsdk/include/cxbase.h	XML SDK header file

Table 5-1 XML SDK File Structure Description *(Continued)*

Subdirectory or File	Description of Contents
xmlsdk/include/cxipconn.h	XML SDK header file for CXIPConnection class
xmlsdk/include/cxipinit.h	XML SDK header file for CXIPInit class
xmlsdk/include/cxiplsnr.h	XML SDK header file for CXIPListener class
xmlsdk/include/cxipmsg.h	XML SDK header file for CXIP_MSG class
xmlsdk/include/cxsbmt.h	XML SDK header file for CXSubmit class
xmlsdk/include/cxtypes.h	XML SDK header file
xmlsdk/include/cxxsdom.h	XML SDK header file for CXxsDOM class
xmlsdk/include/cxxsmsg.h	XML SDK header file for CXxsMSG class
xmlsdk/include/xmlparser	The XML SDK xmlparser include directory
xmlsdk/include/xmlparser/xmlparse.h	The XML SDK xmlparser header file
xmlsdk/lib	The XML SDK library directory
xmlsdk/lib/libecxmlcxbase.a	XML SDK library file
xmlsdk/lib/libecxmlcxcs.a	XML SDK library file
xmlsdk/lib/libecxmlcxsdk.a	XML SDK library file
xmlsdk/lib/libecxmlcxus.a	XML SDK library file
xmlsdk/lib/libecxmlcxxs.a	XML SDK library file
xmlsdk/lib/libecxmlxml.a	XML SDK xmlparser library file

CXIP_MSG Class Reference

Interface `cxipmsg.h`

Superclasses `CXxsMSG, CXxsDOM`

Subclasses None

Syntax `class CXIP_MSG : public CXxsMSG { ... };`

Constructor and Destructor

CXIP_MSG( )

Creates a `CXIP_MSG` object.

Syntax `CXIP_MSG::CXIP_MSG();`

Parameters None.

Creates a `CXIP_MSG` object undefined content.

Syntax `CXIP_MSG::CXIP_MSG(const char *doc, const char *dtd = CXIP_MSG_DTD);`

Parameters The `CXIP_MSG()` method hras the following parameters:

`doc`	the document
`dtd`	the dtd, pass "" (empty string) if dtd is already embedded in document

Creates a `CXIP_MSG` object given the document and DTD.

Syntax `CXIP_MSG::CXIP_MSG(const char *doc, int opt = 0);`

Parameters The `CXIP_MSG()` method has the following parameters:

`doc`	the XML document including needed DTD
`opt`	the option, which can be OR'ed from the following option: • CXXS_OPT_DELETEDOC - delete the XML message once the internal DOM object tree is formatted after the parsing

Creates a `CXIP_MSG` object given the content from the given object.

Syntax `CXIP_MSG::CXIP_MSG(CXIP_MSG& obj);`

Parameters The `CXIP_MSG( )` method has the following parameter:

`obj`	the object to copy from

~CXIP_MSG( )

Destroys a `CXIP_MSG` object.

Syntax `virtual ~CXIP_MSG();`

CXxsMSG Class Reference

Interface `cxxsmsg.h`

Superclasses `CXxsDOM`

Subclasses `CXIP_MSG`

Syntax `class CXxsMSG : public CXxsDOM { ... };`

Constructor and Destructor

CXxsMSG( )

Creates a CXxsMSG object.

Syntax Not intended to be used directly.

~CXxsMSG( )

Destroys a CXxsMSG object.

Syntax Not intended to be used directly.

Methods

This section lists the methods of the `CXxsMSG` class.

GetMSGTYPE( )

Gets the MSGTYPE attribute from the CONTROL section in the CXIP message.

Syntax `int CXxsMSG::GetMSGTYPE(char **v, int allocstr = 0);`

Parameters The `GetMSGTYPE()` method has the following parameters:

`v`	pointer to the MSGTYPE string pointer
`allocstr`	flag indicating whether to allocate a space for the returned value, or simply point to the object private data

Returns 0 when successful; -1 otherwise.

GetSERVICE( )

Gets the SERVICE attribute from the CONTROL section in the CXIP message.

Syntax `int CXxsMSG::GetSERVICE(char **v, int allocstr = 0);`

Parameters The `GetSERVICE()` method has the following parameters:

`v`	pointer to the SERVICE string pointer
`allocstr`	flag indicating whether to allocate a space for the returned value, or simply point to the object private data

Returns 0 when successful; -1 otherwise.

GetTIMEOUT( )

Gets the TIMEOUT attribute from the CONTROL section in the CXIP message.

Syntax `int CXxsMSG::GetTIMEOUT(long *v);`

Parameters The `GetTIMEOUT( )` method has the following parameters:

`v`	pointer to the TIMEOUT value
`allocstr`	flag indicating whether to allocate a space for the returned value, or simply point to the object private data

Returns 0 when successful; -1 otherwise.

GetRETRIES( )

Gets the RETRIES attribute from the CONTROL section in the CXIP message.

Syntax `int CXxsMSG::GetRETRIES(long *v);`

Parameters The `GetRETRIES( )` method has the following parameter:

`v`	pointer to the RETRIES value

Returns 0 when successful; -1 otherwise.

GetSTATUS( )

Gets the STATUS attribute the CONTROL section in from the CXIP message.

Syntax `int CXxsMSG::GetSTATUS(long *v);`

Parameters The `GetSTATUS( )` method has the following parameter:

`v`	pointer to the STATUS value

Returns 0 when successful; -1 otherwise.

GetSENDER( )

Gets the SENDER attribute from the PREDEFINED MONITOR section in the CXIP message.

Syntax `int CXxsMSG::GetSENDER(char **v, int allocstr = 0) ;`

Parameters The `GetSENDER( )` method has the following parameters:

`v`	pointer to the SENDER string pointer
`allocstr`	flag indicating whether to allocate a space for the returned value, or simply point to the object private data

Returns 0 when successful; -1 otherwise.

GetRECEIVER( )

Gets the RECEIVER attribute from the PREDEFINED MONITOR section in the CXIP message.

Syntax `int CXxsMSG::GetRECEIVER(char **v, int allocstr = 0);`

Parameters The `GetRECEIVER( )` method has the following parameters:

`v`	pointer to the RECEIVER string pointer
`allocstr`	flag indicating whether to allocate a space for the returned value, or simply point to the object private data

Returns 0 when successful; -1 otherwise.

GetTIMESTAMP( )

Gets the TIMESTAMP attribute from the PREDEFINED MONITOR section in the CXIP message.

Syntax `int CXxsMSG::GetTIMESTAMP(char **v, int allocstr = 0);`

Parameters The `GetTIMESTAMP( )` method has the following parameters:

`v`	pointer to the TIMESTAMP string pointer
`allocstr`	flag indicating whether to allocate a space for the returned value, or simply point to the object private data

Returns 0 when successful; -1 otherwise.

GetCONTROL( )

Gets the CONTROL section object from the CXIP message.

Syntax `int CXxsMSG::GetCONTROL(CXxsObj *obj);`

Parameters The `GetCONTROL( )` method has the following parameter:

`obj`	the found control object

Returns 0 when successful; -1 otherwise.

GetMONITOR( )

Gets the PREDEFINED MONITOR section object from the CXIP message.

Syntax `int CXxsMSG::GetMONITOR(CXxsObj *obj);`

Parameters The `GetMONITOR( )` method has the following parameter:

`obj`	the found predefined monitor object

Returns 0 when successful; -1 otherwise.

Gets the USRDEFINED MONITOR section object from the CXIP message.

Syntax `int CXxsMSG::GetMONITOR(const char *n, CXxsObj *obj);`

Parameters The `GetMONITOR( )` method has the following parameters:

`n`	the name of the (user-defined) monitor section
`obj`	the found monitor object

Returns 0 when successful; -1 otherwise.

GetPredefinedMONITOR( )

Gets the PREDEFINED MONITOR section object from the CXIP message.

Syntax `int CXxsMSG::GetPredefinedMONITOR(CXxsObj *obj);`

Parameters The `GetPredefinedMONITOR()` method has the following parameter:

`obj`	the found predefined monitor object

Returns 0 when successful; -1 otherwise.

GetUsrDefinedMONITOR( )

Syntax `int CXxsMSG::GetUsrDefinedMONITOR(CXxsObj *obj);`

Gets the first USRDEFINED MONITOR section object from the CXIP message.

Parameters The `GetUsrDefinedMONITOR()` method has the following parameter:

`obj`	the found monitor objec

Syntax `int CXxsMSG::GetUsrDefinedMONITOR(CXxsObj pobj, CXxsObj *obj);`

Gets the next USRDEFINED MONITOR section object from the CXIP message.

Parameters The `CXxsMSG()` method has the following parameters:

`pobj`	the current monitor object
`obj`	the found monitor object

Syntax `int CXxsMSG::GetUsrDefinedMONITOR(const char *n, CXxsObj *obj);`

Gets the named USRDEFINED MONITOR section object from the CXIP message.

Parameters The `GetUsrDefinedMONITOR()` method has the following parameters:

`n`	the name of the monitor object
`obj`	the found monitor object

Returns 0 when successful; -1 otherwise.

GetINPUT( )

Gets the first INPUT object from the DATA section in the CXIP message.

Syntax `int CXxsMSG::GetINPUT(CXxsObj *obj) ;`

Parameters The `GetINPUT()` method has the following parameter:

`obj`	the found input object

Gets the next INPUT object from the data section in the CXIP message.

Syntax `int CXxsMSG::GetINPUT(CXxsObj pobj, CXxsObj *obj);`

Parameters The `GetINPUT()` method has the following parameters:

`pobj`	the current input object
`obj`	the found input object

Gets the named INPUT object from the data section in the CXIP message.

Syntax `int CXxsMSG::GetINPUT(const char *n, CXxsObj *obj);`

Parameters The `GetINPUT()` method has the following parameters:

`n`	the name of the input object
`obj`	the found input object

Gets the named INPUT value from the data section in the CXIP message.

Syntax `int CXxsMSG::GetINPUT(const char *n, int *v);`

Parameters The `GetINPUT( )` method has the following parameters:

`n`	name of the input object
`v`	pointer to the input value

Gets the named INPUT value from the data section in the CXIP message.

Syntax `int CXxsMSG::GetINPUT(const char *n, char **v, int allocstr = 0);`

Parameters The `GetINPUT( )` method has the following parameters:

`n`	name of the input object
`v`	pointer to the pointer of input value string
`allocstr`	flag indicating whether to allocate a space for the returned value, or simply point to the object private data

Returns 0 when successful; -1 otherwise.

GetOUTPUT( )

Gets the first OUTPUT object from the DATA section in the CXIP message.

Syntax `int CXxsMSG::GetOUTPUT(CXxsObj *obj);`

Parameters The `GetOUTPUT( )` method has the following parameter:

`obj`	the found output object

Gets the next OUTPUT object from the data section in the CXIP message.

Syntax `int CXxsMSG::GetOUTPUT(CXxsObj pobj, CXxsObj *obj);`

Parameters The `GetOUTPUT( )` method has the following parameters:

`pobj`	the current output object
`obj`	the found output object

Gets the named OUTPUT object from the data section in the CXIP message.

Syntax `int CXxsMSG::GetOUTPUT(const char *n, CXxsObj *obj);`

Parameters The `GetOUTPUT( )` method has the following parameters:

`n`	the name of the output object
`obj`	the found output object

Gets the named OUTPUT value from the data section in the CXIP message.

Syntax `int CXxsMSG::GetOUTPUT(const char *n, int *v);`

Parameters The `GetOUTPUT( )` method has the following parameters:

`n`	name of the output object
`v`	pointer to the output value

Gets the named OUTPUT value from the data section in the CXIP message.

Syntax `int CXxsMSG::GetOUTPUT(const char *n, char **v, int allocstr = 0);`

Parameters The `GetOUTPUT( )` method has the following parameters:

`n`	name of the output object
`v`	pointer to the pointer of output value string
`allocstr`	flag indicating whether to allocate a space for the returned value, or simply point to the object private data

Returns 0 when successful; -1 otherwise.

SetCONTROL( )

Sets the specified attribute in the CONTROL section for a CXIP message.

Syntax `int CXxsMSG::SetCONTROL(const char *n, long v);`

Parameters The `SetCONTROL()` method has the following parameters:

`n`	the name of the attribute
`v`	the attribute value

Sets the specified attribute in the CONTROL section for a CXIP message.

Syntax `int CXxsMSG::SetCONTROL(const char *n, const char *v);`

Parameters The `SetCONTROL()` method has the following parameters:

`n`	the name of the attribute
`v`	the attribute value

Returns 0 when successful; -1 otherwise.

SetMSGTYPE( )

Sets the MSGTYPE attribute in the CONTROL section for a CXIP message.

Syntax `int CXxsMSG::SetMSGTYPE(const char *v);`

Parameters The `SetMSGTYPE()` method has the following parameter:

`v`	the MSGTYPE value

Returns 0 when successful; -1 otherwise.

SetSERVICE( )

Sets the SERVICE attribute in the CONTROL section for a CXIP message.

Syntax `int CXxsMSG::SetSERVICE(const char *v);`

Parameters The `SetSERVICE( )` method has the following parameter:

v	the SERVICE value

Returns 0 when successful; -1 otherwise.

SetTIMEOUT( )

Sets the TIMEOUT attribute in the CONTROL section for a CXIP message.

Syntax `int CXxsMSG::SetTIMEOUT(long v);`

Parameters The `SetTIMEOUT( )` method has the following parameter:

v	the TIMEOUT value

Returns 0 when successful; -1 otherwise.

SetRETRIES( )

Sets the RETRIES attribute in the CONTROL section for a CXIP message.

Syntax `int CXxsMSG::SetRETRIES(long v);`

Parameters The `SetRETRIES( )` method has the following parameter:

v	the RETRIES value

Returns 0 when successful; -1 otherwise.

SetSTATUS( )

Sets the STATUS attribute in the CONTROL section for a CXIP message.

Syntax `int CXxsMSG::SetSTATUS(long v);`

Parameters The `SetSTATUS( )` method has the following parameter:

v	the STATUS value

Returns 0 when successful; -1 otherwise.

SetPreDefinedMONITOR( )

Sets the specified attribute in the PREDEFINED MONITOR section for a CXIP message.

Syntax `int CXxsMSG::SetPreDefinedMONITOR(const char *n, const char *v);`

Parameters The `SetPreDefinedMONITOR( )` method has the following parameters:

n	the name of the attribute
v	the attribute value

Returns 0 when successful; -1 otherwise.

SetSENDER( )

Sets the SENDER attribute in the PREDEFINED MONITOR section for a CXIP message.

Syntax `int CXxsMSG::SetSENDER(const char *v);`

Parameters The `SetSENDER( )` method has the following parameter:

v	the SENDER attribute value

Returns 0 when successful; -1 otherwise.

SetRECEIVER( )

Sets the RECEIVER attribute in the PREDEFINED MONITOR section for a CXIP message.

Syntax `int CXxsMSG::SetRECEIVER(const char *v);`

Parameters The `SetRECEIVER( )` method has the following parameter:

v	the RECEIVER attribute value

Returns 0 when successful; -1 otherwise.

SetTIMESTAMP( )

Sets the TIMESTAMP attribute in the PREDEFINED MONITOR section for a CXIP message.

Syntax `int CXxsMSG::SetTIMESTAMP(const char *v);`

Parameters The `SetTIMESTAMP( )` method has the following parameter:

v	the TIMESTAMP attribute value

Returns 0 when successful; -1 otherwise.

SetUsrDefinedMONITOR( )

Sets the specified attribute in the USRDEFINED MONITOR section for a CXIP message.

Syntax `int CXxsMSG::SetUsrDefinedMONITOR(const char *n, const char *v);`

Parameters The `SetUsrDefinedMONITOR( )` method has the following parameters:

n	the name of the attribute
v	the attribute value

Returns 0 when successful; -1 otherwise.

SetINPUT( )

Sets the specified input variable in the DATA section for a CXIP message.

Syntax `int CXxsMSG::SetINPUT(const char *n, const char *v);`

Parameters The `SetINPUT()` method has the following parameters:

`n`	the name of the input variable
`v`	the variable value

Sets the specified input attribute in the DATA section for a CXIP message.

Syntax `int CXxsMSG::SetINPUT(const char *n, const char *g, long v);`

Parameters The `SetINPUT()` method has the following parameters:

`n`	the name of the input variable
`g`	the attribute name
`v`	the attribute value

Sets the specified input attribute in the DATA section for a CXIP message.

Syntax `int CXxsMSG::SetINPUT(const char *n, const char *g, const char *v);`

Parameters The f method has the following parameters:

`n`	the name of the input variable
`g`	the attribute name
`v`	the attribute value

Returns 0 when successful; -1 otherwise.

SetOUTPUT( )

Sets the specified output attribute in the DATA section for a CXIP message.

Syntax `int CXxsMSG::SetOUTPUT(const char *n, const char *g, long v);`

Parameters The `SetOUTPUT( )` method has the following parameters:

`n`	the name of the output variable
`g`	the attribute name
`v`	the attribute value

Sets the specified output attribute in the DATA section for a CXIP message.

Syntax `int CXxsMSG::SetOUTPUT(const char *n, const char *g, const char *v);`

Parameters The `SetOUTPUT( )` method has the following parameters:

`n`	the name of the output variable
`g`	the attribute name
`v`	the attribute value

Returns 0 when successful; -1 otherwise.

CreateMSG( )

Starts creating a CXIP message.

Syntax `int CXxsMSG::CreateMSG(const char *n, const char *v);`

Parameters The `CreateMSG( )` method has the following parameters:

`n`	the name of the message; must be `CXIP_MSG`
`v`	the version of the message; must be `1.0`

Returns 0 when successful; -1 otherwise.

CreateCONTROL( )

Creates the CONTROL section for a CXIP message.

Syntax `int CXxsMSG::CreateCONTROL(const char *m, const char *s);`

Parameters The `CreateCONTROL()` method has the following parameters:

`m`	the value of MSGTYPE attribute
`s`	the value of SERVICE attribute

Returns 0 when successful; -1 otherwise.

CreateTIMEOUT( )

Creates the TIMEOUT attribute value in the CONTROL section for a CXIP message.

Syntax `int CXxsMSG::CreateTIMEOUT(long v);`

Parameters The `CreateTIMEOUT()` method has the following parameter:

`v`	the value of TIMEOUT attribute

Returns 0 when successful; -1 otherwise.

CreateRETRIES( )

Creates the RETRIES attribute value in the CONTROL section for a CXIP message.

Syntax `int CXxsMSG::CreateRETRIES(long v);`

Parameters The `CreateRETRIES()` method has the following parameter:

`v`	the value of RETRIES attribute

Returns 0 when successful; -1 otherwise.

CreateSTATUS( )

Creates the STATUS attribute value in the CONTROL section for a CXIP message.

Syntax `int CXxsMSG::CreateSTATUS(long v);`

Parameters The `CreateSTATUS()` method has the following parameter:

`v`	the value of STATUS attribute

Returns 0 when successful; -1 otherwise.

CreatePreDefinedMONITOR( )

Creates the PREDEFINED MONITOR section for a CXIP message.

Syntax `int CXxsMSG::CreatePreDefinedMONITOR(const char *s, const char *r, const char *t = 0);`

Parameters The `CreatePreDefinedMONITOR()` method has the following parameters:

`s`	the value of SENDER attribute
`r`	the value of RECEIVER attribute
`t`	the value of TIMESTAMP attribute; passing zero value causes it to be created internally using current time

Returns 0 when successful; -1 otherwise.

CreateUsrDefinedMONITOR( )

Creates create the USRDEFINED MONITOR section for a CXIP message.

Syntax `int CXxsMSG::CreateUsrDefinedMONITOR(const char *n, const char *t, const char *v);`

Parameters The `CreateUsrDefinedMONITOR()` method has the following parameters:

`n`	the value of NAME attribute
`t`	the value of TYPE attribute
`v`	the value of the data

Returns 0 when successful; -1 otherwise.

CreateINPUT( )

Creates the INPUT variable in the DATA section for a CXIP message.

Syntax `int CXxsMSG::CreateINPUT(const char *n, const char *t, const char *v, int opt = 0);`

Parameters The `CreateINPUT()` method has the following parameters:

`n`	the value of NAME attribute
`t`	the value of TYPE attribute
`v`	the value of the data
`opt`	the option, which can be OR'ed from the following options: • CXXS_OPT_KEEPIT - use the string directly; do not duplicate another copy internally • CXXS_OPT_FREE - use `free()` instead of `delete` to release the string

Returns 0 when successful; -1 otherwise.

Creates the INPUT variable in the DATA section for a CXIP message.

Syntax `int CXxsMSG::CreateINPUT(const char *n, const char *t, const char *v, const char *charset, const char *encoding, *v, int opt = 0);`

Parameters The `CreateINPUT()` method has the following parameters:

n	the value of NAME attribute
t	the value of TYPE attribute
`v`	the value of the data
`charset`	the value of CHARSET attribute
`encoding`	the value of ENCODING attribute

opt	the option, which can be OR'ed from the following options: • CXXS_OPT_KEEPIT - use the string directly; do not duplicate another copy internally • CXXS_OPT_FREE - use `free()` instead of `delete` to release the string

Returns 0 when successful; -1 otherwise.

CreateOUTPUT( )

Creates the OUTPUT variable in the DATA section for a CXIP message.

Syntax `int CXxsMSG::CreateOUTPUT(const char *n, const char *t, const char *v, int opt = 0);`

Parameters The `CreateOUTPUT()` method has the following parameters:

n	the value of NAME attribute
t	the value of TYPE attribute
v	the value of the data
opt	the option, which can be OR'ed from the following options: • CXXS_OPT_KEEPIT - use the string directly; do not duplicate another copy internally • CXXS_OPT_FREE - use `free()` instead of `delete` to release the string

Returns 0 when successful; -1 otherwise.

Creates the OUTPUT variable in the DATA section for a CXIP message.

Syntax `int CXxsMSG::CreateOUTPUT(const char *n, const char *t, const char *v, const char *charset, const char *encoding, int opt = 0);`

Parameters The `CreateOUTPUT()` method has the following parameters:

n	the value of NAME attribute
t	the value of TYPE attribute
v	the value of the data

`charset`	the value of CHARSET attribute
`encoding`	the value of ENCODING attribute
`opt`	the option, which can be OR'ed from the following options: • CXXS_OPT_KEEPIT - use the string directly; do not duplicate another copy internally • CXXS_OPT_FREE - use `free()` instead of `delete` to release the string

Returns 0 when successful; -1 otherwise.

CXxsDOM Class Reference

Interface `cxxsdom.h`

Superclasses Not applicable.

Subclasses `CXxsMSG, CXIP_MSG`

Syntax `class CXxsDOM { ... };`

Constructor and Destructor

CXxsDOM( )

Creates a CXxsDOM object.

Syntax Not intended to be used directly.

~CXxsDOM( )

Destroys a CXxsDOM object.

Syntax Not intended to be used directly.

Methods

This section lists the methods of the `CXxsDOM` class.

Parse( )

Parses an XML-formatted message, which is passed to this object from the constructor.

Syntax `int CXxsDOM::Parse(int opt = 0);`

Parameters The `Parse()` method has the following parameter:

`opt`	the option, which can be OR'ed from the following options: • CXXS_OPT_DELETEDOC - delete the XML message once the internal DOM object tree is formatted after the parsing

Returns 0 when successful; -1 otherwise.

Format( )

Formats an XML-formatted message from the internal DOM object tree created previously by the Create methods.

Syntax `int CXxsDOM::Format(int opt = 0);`

Parameters The `Format()` method has the following parameter:

`opt`	the option, which can be OR'ed from the following options: • CXXS_OPT_DELETEDOM - delete the internal DOM object tree once the XML message is formatted/constructed

Returns 0 when successful; -1 otherwise.

GetErrors( )

Retrieves information about the parsing error.

Syntax `const char *CXxsDOM::GetErrors(int *ecode, int *eline, int *ecol);`

Parameters The `GetErrors()` method has the following parameters:

`ecode`	the error code
`eline`	the line number where error is detected
`ecol`	the column number where error is detected

Returns The error message, if available.

GetDTD( )

Gets the XML DTD from this object.

Syntax `inline const char *CXxsDOM::GetDTD();`

Parameters None.

Returns The DTD string pointer.

GetDocument( )

Gets the XML document from this object.

Syntax `inline const char *CXxsDOM::GetDocument();`

Parameters None.

Returns The document string pointer.

GetObjectName( )

Gets the object name from a CXxsDOM object.

Syntax `int CXxsDOM::GetObjectName(CXxsObj obj, char **v);`

Parameters The `GetObjectName()` method has the following parameters:

`obj`	the object
`v`	the value of the object name

Returns 0 when successful; -1 otherwise.

GetObjectData( )

Gets the object data from a CXxsDOM object.

Syntax `int CXxsDOM::GetObjectData(CXxsObj obj, char **v);`

Parameters The `GetObjectData()` method has the following parameters:

`obj`	the object
`v`	the value of the object data

Returns 0 when successful; -1 otherwise.

GetObjectAttribute( )

Gets the object attribute from a CXxsDOM object.

Syntax `int CXxsDOM::GetObjectAttribute(CXxsObj obj, const char *n, int *v);`

Parameters The `GetObjectAttribute()` method has the following parameters:

`obj`	the object
`n`	the name of the object attribute
`v`	the value of the object attribute

Syntax `int CXxsDOM::GetObjectAttribute(CXxsObj obj, const char *n, char **v);`

Parameters The `GetObjectAttribute()` method has the following parameters:

`obj`	the object
`n`	the name of the object attribute
`v`	the value of the object attribute

Returns 0 when successful; -1 otherwise.

CXIPInit Class Reference

Interface `cxipinit.h`

Superclasses Not applicable.

Subclasses None.

Syntax `class CXIPInit { ... };`

Constructor and Destructor

CXIPInit( )

Creates a CXIPInit object.

Syntax `CXIPInit::CXIPInit();`

Parameters None.

~CXIPInit( )

Destroys a CXIPInit object.

Syntax `virtual ~CXIPInit();`

Methods

This section lists the methods of the CXIPInit class.

Init( )

Initializes the XML SDK application.

Syntax `int CXIPInit::Init();`

Parameters None.

Returns 0 when successful; -1 otherwise.

SetDebugMode( )

Sets debug mode of the application.

Syntax `void CXIPInit::SetDebugMode(int d);`

Parameters The `SetDebugMode()` method has the following parameter:

`d`	the debug mode - 1 when on, 0 when off

SetLogFiles( )

Sets output files for debug messages.

Syntax `void CXIPInit::SetLogFiles(const char *o, const char *e);`

Parameters The `SetLogFiles()` method has the following parameters:

`o`	the output file for stdout messages - stdout when nil
`e`	the output file for stderr messages - stderr when nil

Base64Decode( )

Performs a Base64 decoding.

Syntax `static void *CXIPInit::Base64Decode(char *src, long& srclen, long &declen);`

Parameters The `Base64Decode()` method has the following parameters:

`src`	the source of the (encoded) string to be decoded
`srclen`	the length of the (encoded) source length
`declen`	the length of the decoded string length

Returns The encoded string when successful; 0 otherwise.

Base64Encode( )

Performs a Base64 decoding.

Syntax `static char *CXIPInit::Base64Encode(void *src, long& srclen, long &enclen);`

Parameters The `Base64Encode()` method has the following parameters:

`src`	the source of the string to be encoded
`srclen`	the length of the source length
`enclen`	the length of the encoded string length

Returns The encoded string when successful; 0 otherwise.

CXIPConnection Class Reference

Interface `cxipconn.h`

Superclasses Not applicable.

Subclasses None.

Syntax `class CXIPConnection { ... };`

Constructor and Destructor

CXIPConnection( )

Creates a CXIPConnection object.

Syntax `CXIPConnection::CXIPConnection();`

Parameters None.

~CXIPConnection( )

Destroys a CXIPConnection object.

Syntax `virtual ~CXIPConnection();`

Methods

This section lists the methods of the CXIPConnection class.

Connect( )

Connects to a specified host and port.

Syntax `int CXIPConnection::Connect(const char *host, int port);`

Parameters The `Connect()` method has the following parameters:

`host`	the host name or IP address
`port`	the port number

Returns 0 when successful; -1 otherwise.

SendMsg( )

Sends a message through the connection.

Syntax `int CXIPConnection::SendMsg(const char *m);`

Parameters The `SendMsg()` method has the following parameter:

`m`	the null-terminated message string

Returns The number of bytes sent when successful; -1 otherwise.

ReceiveMsg( )

Receives a message from the connection.

Syntax `int CXIPConnection::ReceiveMsg(char **m);`

Parameters The `ReceiveMsg()` method has the following parameter:

`m`	the null-terminated message string - it is allocated inside the object and expected to be released by the caller

Returns The number of bytes received when successful; -1 otherwise.

CXIPListener Class Reference

Interface `cxiplsnr.h`

Superclasses Not applicable.

Subclasses None.

Syntax `class CXIPListener { ... };`

Constructor and Destructor

CXIPListener( )

Creates a CXIPListener object.

Syntax `CXIPListener::CXIPListener();`

Parameters None.

~CXIPListener( )

Destroys a CXIPListener object.

Syntax `virtual ~CXIPListener();`

Methods

This section lists the methods of the CXIPListener class.

Init( )

Initializes the listener.

Syntax
```
int CXIPListener::Init(const char *conf, const char *sec,
const char *sys = "system");
```

Parameters The `Init()` method has the following parameters:

`conf`	the (ini-formatted) configuration file name
`sec`	the section name in the configuration file
`sys`	the system section name in the configuration file - "system" is the default

Returns 0 when successful; -1 otherwise.

Run( )

Starts up (runs) the listener.

Syntax `virtual int CXIPListener::Run(int blocked = 0);`

Parameters The `Run()` method has the following parameter:

`blocked`	the flag indicating whether to run the listener in blocking mode or not - 0 is non-blocking, any other value is blocking

Returns 0 when successful; -1 otherwise.

ProcessMessage( )

Processes a message received from a given conection.

Syntax `virtual int CXIPListener::ProcessMessage(CXIPConnection *conn, const char *m);`

Parameters The `ProcessMessage()` method has the following parameters:

`conn`	the connection from which the message is received
`m`	the null-terminated message string

Returns 0 when successful; -1 otherwise.

Syntax `virtual int CXIPListener::ProcessMessage(CXIPConnection *conn, CXIP_MSG *m);`

Parameters The `ProcessMessage()` method has the following parameters:

`conn`	the connection from which the message is received
`m`	the parsed XML message in CXIP_MSG format

Returns 0 when successful; -1 otherwise.

CXSubmit Class Reference

Interface `cxsbmt.h`

Superclasses Not applicable.

Subclasses None.

Syntax `class CXSubmit { ... };`

Constructor and Destructor

CXSubmit( )

Creates a CXSubmit object.

Syntax `CXSubmit::CXSubmit();`

Parameters None.

~CXSubmit( )

Destroys a CXSubmit object.

Syntax `virtual ~CXSubmit();`

Methods

This section lists the methods of the `CXSubmit` class.

Submit( )

Submits the document using related parameters specifed inside this object.

Syntax `int CXSubmit::Submit();`

Parameters None. Parameters are specified inside this object.

Returns 0 when successful; -1 otherwise.

SetHost( )

Sets the host name or IP address to submit to.

Syntax `int CXSubmit::SetHost(const char *host);`

Parameters The `CXSubmit()` method has the following parameter:

`host`	the host name or IP address to submit to

Returns 0 when successful; -1 otherwise.

SetPort( )

Sets the port number to submit to.

Syntax `int CXSubmit::SetPort(const char *host);`

Parameters The `SetPort()` method has the following parameter:

`port`	the port number to submit to

Returns 0 when successful; -1 otherwise.

SetSender( )

Sets the sender name.

Syntax `int CXSubmit::SetSender(const char *sender);`

Parameters The `SetSender()` method has the following parameter:

`sender`	the sender of the submission

Returns 0 when successful; -1 otherwise.

SetReceiver( )

Sets the receiver name.

Syntax `int CXSubmit::SetReceiver(const char *receiver);`

Parameters The `SetReceiver()` method has the following parameter:

`receiver`	the receiver of the submission

Returns 0 when successful; -1 otherwise.

SetDocType( )

Sets the document type.

Syntax `int CXSubmit::SetDocType(const char *doctype);`

Parameters The `SetDocType()` method has the following parameter:

`doctype`	the document type of the submission

Returns 0 when successful; -1 otherwise.

SetDocPath( )

Sets the document path.

Syntax `int CXSubmit::SetDocPath(const char *docpath);`

Parameters The `SetDocPath()` method has the following parameter:

`docpath`	the document path of the submission

Returns 0 when successful; -1 otherwise.

SetDocTransport( )

Sets the document transport method.

Syntax `int CXSubmit::SetDocTransport(const char *doctrans);`

Parameters The `SetDocTransport()` method has the following parameter:

`doctrans`	the transport method of the submission

Returns 0 when successful; -1 otherwise.

SetIDs( )

Sets the sender and receiver IDs/names for CXIP message.

Syntax `int CXSubmit::SetIDs(const char *s, const char *r);`

Parameters The `SetIDs()` method has the following parameters:

`s`	the sender id
`r`	the receiver id

NOTE These are *not* the same Sender/Receiver as in the partnership.

Examples

Makefile The `Makefile.{solaris|hpux}` under the example directory needs only minimal modifications to build the sample programs. The two steps are:

1. Change the `ECXpert = ${ECXPERT-INSTALLATION-LOCATION}` to the path of the installation. For example:

   `/user/apps/ECX/NS-apps/ECXpert`

2. Change the `CC = ${YOUR_CPP_COMPILER}` to the path of the C++ compiler.

Source Code See the source files under the `xmlsdk/example` directory.

Configuration File See `xmlsdk/config/xmlserver.ini` for a configuration example.

Chapter 6

The EcxBase Class

This chapter describes the `EcxBase` class, which is the base class for all APIs in ECXpert. This chapter contains the following sections:

About the EcxBase Class

The `EcxBase` class defines the class from which all ECXpert API classes are derived. For example, ECXpert's `EcxSubmit` class is derived from the `EcxBase` class. You may define a subclass derived from the `EcxBase` class. The `EcxBase` class is intended to be used as an abstract class. You should never need to create `EcxBase` objects.

The `EcxBase` class defines methods that are common to the ECXpert API classes you use to interact programmatically with the ECXpert System. The class provides methods that allow you to get, set, and clear the error number corresponding to the last error reported by ECXpert.

Methods Summary list:

Constructor and destructor	
`EcxBase()`	Creates an `EcxBase` object.
`~EcxSBase()`	Destroys an `EcxBase` object.
Error handling	
`Errnum()`	Retrieves or sets the last error.

`ClearErr()`	Clears the last error that occurred.
`ErrMsg`	Returns error message string.

EcxBase Class Reference

Interface `ecxbase.h`

Superclasses None

Subclasses `EcxAddresses`, `EcxDocument`, `EcxFTPClient`, `EcxInit`, `EcxLog`, `EcxLogin`, `EcxMember`, `EcxPartnership`, `ECXService`, `ECXServiceList`, `EcxSubmit`, `EcxTracking`

Friend Classes None

Syntax
```
class EcxBase { ... };
```

Constants and Data Types

The following definitions, which are defined at file scope, allow you to specify boolean values as integers:

Syntax
```
#define TRUE 1
#define FALSE        0
```

`TRUE`	A true value, which is represented as 1.
`FALSE`	A false value, which is represented as 0.

Constructor and Destructor

EcxBase( )

Creates an `EcxBase` object.

Syntax EcxBase(void);

~EcxBase( )

Destroys an `EcxBase` object.

Syntax `virtual ~EcxBase();`

Methods

This section lists the methods of the `EcxBase` class.

ClearErr( )

Clears the last error that occurred.

Syntax `virtual void ClearErr(void);`

Discussion The last error that occurred as a result of calling a method in the ECXpert API is available until it is explicitly cleared by calling this method or until it has been reset by calling the `Errnum()` method. The `ClearErr()` method sets the error number to 0.

Example `pSubmitObj->ClearErr();`

See also The `Errnum()` method on page 107.

Errnum( )

Retrieves or sets the last error.

Syntax `virtual long Errnum(void);`

`virtual void Errnum(long ErrNum);`

Parameters The `Errnum()` method has the following parameters:

`ErrNum`	A long integer that specifies the error number.

Returns A long integer that contains the last error that occurred.

Discussion The first form of the `Errnum( )` method returns the last error that occurred. The second form sets the value of the error number. The second form is protected.

NOTE When you use the API, ECXpert sets the error number.

Example
```
if ( pSubmitObj->Errnum( ) )
   printf("Error: %ld occurred\n", pSubmitObj->Errnum( );
```

See also Call the `ClearErr( )` method on page 107 to reset the error number to 0.

Errmsg( )

Returns error message string.

Syntax `virtual const char * Errmsg(void);`

Returns Pointer to a character string containing the last error message that occurred.

Discussion This value could be null, because not every object gets the error message. Refer to the code examples for each class in this book to determine whether it will return an error message. For example, the `ECXLogin` class will return an error message if it fails.

Example

```
if((pLogin = new EcxLogin())->Errnum()) {
    cout << "EcxLogin Object Error:" << endl;
    cout << "\tErrnum: " << pLogin->Errnum() << endl;
    cout << "\tErrmsg: " << pLogin->Errmsg() << endl;
    cout << endl;
    return(NULL);}
```

See Also The `EcxLogin( )` class on page 127.

Chapter 7

The EcxInit Class

This chapter describes the `EcxInit` class, whose objects initialize your application to for ECXpert database access. This chapter contains the following sections:

About the EcxInit Class

You must create an `EcxInit` object before using any other class in the SDK.

Methods Summary list:

Constructor and destructor	
`EcxInit()`	Creates an `EcxInit` object.
`~EcxInit()`	Destroys an `EcxInit` object.

Using the EcxInit Class

You must create a single `EcxInit` object within your application. You can call the class's `Errnum()` method to determine whether initialization succeeded.

```
int main(int argc, char * argv[])
{
  ...
  EcxInit EcxInitObject;
  ...
  do // main processing loop
  {
    if ( EcxInitObject.Errnum() != 0 )
    {
       printf("Failed to initialize EcxInit object.\n");
      break;
    }
  ...
  }
  ...
}
```

EcxInit Class Reference

Interface `ecxinit.h`

Superclasses `EcxBase`

Subclasses None

Friend Classes None

Syntax `class EcxInit : public EcxBase { ... };`

Constructor and Destructor

EcxInit( )

Creates an `EcxInit` object.

Syntax `EcxInit(void);`

Example See "Using the EcxInit Class" on page 110.

~EcxInit( )

Destroys an `EcxInit` object.

Syntax `virtual ~EcxInit();`

~EcxInit( )

Chapter 8

The EcxSubmit Class

This chapter describes the `EcxSubmit` class, which defines methods that you use to submit files to ECXpert. This chapter contains the following sections:

About the EcxSubmit Class

The `EcxSubmit` class defines methods that you use to submit a file to ECXpert. You can use these methods to provide a file submission capability within your application instead of requiring the user to execute a command or use ECXpert's HTML interface to submit an object.

You may create objects from the `EcxSubmit` class and use them directly or you may define a subclass of the `EcxSubmit` class and create objects from the derived class. For example, you might define a subclass that handles much of the application logic associated with files to be submitted to ECXpert. Objects derived from your subclass would inherit the ability to submit files to ECXpert.

Before you create an `EcxSubmit` object, you must first create an `EcxInit` object. You then can create an `EcxSubmit` object and specify the following information:

- Member ID of the sender
- Member ID of the recipient
- Sender's password, which is optional for trusted members
- Full path of ECXpert's configuration file
- Map name (optional)

- Delivery method (optional)
- File name
- File type

You call methods to specify this information. For example, you call the object's `SetSender` method (page 125) to specify the sender's member ID.

You must specify the files that you wish to submit to ECXpert. You build a submission list by calling the object's `AddFile()` method (page 118) to add a file to the list. You specify the following information when you add a file:

- Document name
- Document type, such as EDIFACT or EDIX12, or a non-EDI type

You can add as many files as you want. If you add more than one file, the files become part of a single multi-part file. When you finish adding the files to the submission list, you can call the object's `Submit()` method (page 125) to submit the files.

If the file being submitted is in the local file system, ECXpert moves the file being submitted to the directory specified by the `repository` entry in the configuration file's `tcpip-connector` section.

You can also submit files to ECXpert using a TCP/IP connection. You specify whether or not to use a TCP/IP connection when you call the object's `Submit()` method. Using a TCP/IP connection causes ECXpert to stream the contents of the files through a socket to the server. This is a useful technique if your application runs on a remote computer and the files being submitted are relatively small. If you want to submit large files from a remote computer, you should consider using a protocol such as FTP to copy the files to the server and then submit them from the server.

NOTE If you stream data through a TCP/IP connection, the source file is not deleted after the data has been streamed to the server.

After you submit a file, you should check for errors. If no error occurred, you can call the object's `GetFirstTrackingID()` method (page 119) to determine the tracking ID of the first file submitted and the object's `GetNextTrackingID()` method (page 121) to determine the tracking ID for each additional file in the list.

When you no longer need references to these files, you can call the object's `ClearFileList()` method (page 119) to remove the files from the list. You could then add new file(s) by calling the `AddFile()` method and then submit the new file by calling the `Submit()` method.

Methods Summary list:

Constructor and destructor	
`EcxSubmit()`	Creates a submission object.
`~EcxSubmit()`	Destroys a submission object.
Retrieving submission information	
`GetDeliveryMethod`	Gets the delivery method.
`GetEcxIniFileName`	Gets the full pathname of ECXpert's configuration file.
`GetMapName`	Gets the map name.
`GetPassword`	Gets the sender's password
`GetSender`	Gets the sender's member ID.
Setting submission information	
`SetSender()`	Sets the sender's member ID.
`SetRecipient()`	Sets the recipient's member ID.
`SetPassword()`	Sets the sender's password.
`SetEcxIniFileName()`	Sets the full pathname of ECXpert's configuration file.
`SetMapName()`	Sets the map name.
`SetDeliveryMethod()`	Sets the delivery method.
Manipulating the submission list	
`AddFile()`	Adds a file to the submission list.
`ClearFileList()`	Clears the submission list.
`GetFirstTrackingID()`	Retrieves the tracking ID for the first file in the object's submission list.
`GetNextTrackingID()`	Retrieves the tracking ID for the next file in the object's submission list.
Submitting files	
`Submit()`	Submits objects to ECXpert for processing.

Using the EcxSubmit Class

The following program shows how to use the `EcxSubmit` class. The program creates an `EcxSubmit` object and sets the sender, receiver, password, map name, and initialization file. It then adds three files to the submission list and submits them to ECXpert for processing. After submitting the files, the program retrieves the tracking IDs of these files.

```
#include <stdio.h>
#include "ecxsubmit.h"
int main(int argc, char * argv[])
{
   int  retval = -1;
EcxInit     EcxInitObject;    // must instantiate this
                              // before calling sdk
EcxSubmit * pSubmitObj = 0;
do
   {
       if ( EcxInitObject.Errnum() != 0 )
       {
           printf("Failed to initialize EcxInit object.\n");
           break;
       }
if ( (pSubmitObj = new EcxSubmit) == 0 )
       {
           printf("No memory to create Ecxpert submission object.\n");
           break;
       }
if ( pSubmitObj->SetSender("jim1").Errnum()            ||
            pSubmitObj->SetRecipient("smani1").Errnum()       ||
            pSubmitObj->SetPassword("jim1").Errnum()          ||
            pSubmitObj->SetMapName("mymap").Errnum()          ||
            pSubmitObj->SetEcxIniFileName("ecx.ini").Errnum() ||
            pSubmitObj->SetDeliveryMethod("via-my-app").Errnum() )
       {
           printf("Failed to set submission parameters.\n");
           break;
       }
if ( pSubmitObj->AddFile("input1.dat", "edi850").Errnum() ||
            pSubmitObj->AddFile("input2.dat", "edi850").Errnum() ||
            pSubmitObj->AddFile("input3.dat", "edi850").Errnum() )
       {
           printf("Failed to add files to the submission object.\n");
           break;
       }
printf("Submission parameters are as follows:\n"
             "ECXpert configuration file = %s\n"
             "Sender name = %s\n"
             "Recipient name = %s\n"
             "Password = %s\n"
             "Delivery method = %s\n"
             "Map name = %s\n",
             pSubmitObj->GetEcxIniFileName(),pSubmitObj->GetSender(),
```

```
            pSubmitObj->GetRecipient(), pSubmitObj->GetPassword(),
            pSubmitObj->GetDeliveryMethod(),pSubmitObj->GetMapName());
printf("Submitting files now......\n");
if ( pSubmitObj->Submit().Errnum() )
       {
           printf("Submission failed.\n");
           break;
       }
long TrackingID = pSubmitObj->GetFirstTrackingID();
for ( int LoopCount = 1; TrackingID != 0; ++LoopCount )
       {
           printf("Registered file input%d with Tracking ID %ld\n",
                                              LoopCount, TrackingID);
           TrackingID = pSubmitObj->GetNextTrackingID();
       }
retval = 0; // set return code to success
   }
   while( 0 );
if ( pSubmitObj )
   {
       if ( pSubmitObj->Errnum() )
       {
           printf("Error: %ld\n", pSubmitObj->Errnum());
       }
delete pSubmitObj;
   }
return(retval);
```

EcxSubmit Class Reference

Interface `ecxsubmit.h`

Superclasses `EcxBase`

Subclasses None

Friend Classes None

Syntax `class EcxSubmit : public EcxBase { ... };`

Constructor and Destructor

EcxSubmit( )

Creates a submission object.

Syntax `EcxSubmit(void);`

Discussion The constructor creates a submission object.

Example See "Using the EcxSubmit Class" on page 116.

~EcxSubmit( )

Destroys a submission object.

Syntax `~EcxSubmit(void);`

Discussion The destructor destroys a submission object.

Example See "Using the EcxSubmit Class" on page 116.

See Also The `Submit()` method on page 125.

Methods

This section lists the methods of the `EcxSubmit` class.

AddFile( )

Adds a file to the submission list.

Syntax
```
EcxSubmit& AddFile(const char * pFileName,
                   const char * pFileType);
```

Parameters The `AddFile()` method has the following parameters:

`pFileName`	A pointer to the path and file name of the file you want to include with this submission.
`pFileType`	A pointer to the data type of the file you want to include with this submission.

Returns A reference to this submission object.

Discussion The `AddFile()` method adds the specified file to the submission list. You can add as many files to the submission list as you wish. If you add more than one file, the files become part of a single multi-part file.

If you do not specify the path name, ECXpert looks for the file in the directory where the `tcpip-connector` server is executing. You can avoid errors locating the file by specifying the full path name as part of the file name.

After you add the files and specify the other information associated with the submission object, you can call the object's `Submit()` method to submit the files to ECXpert for processing. You should immediately check for errors after calling the `Submit()` method. If an error occurs, none of the files are submitted. They are either all submitted successfully or none of them are submitted.

Example See "Using the EcxSubmit Class" on page 116.

See Also The `Submit()` method on page 125.

ClearFileList( )

Clears the file list.

Syntax `void ClearFileList(void);`

Discussion All files associated with this submission instance can no longer be referenced.

See Also The `AddFile()` method on page 118.

GetDeliveryMethod( )

Retrieves the delivery method set by the `SetDeliveryMethod()` method.

Syntax `virtual const char* GetDeliveryMethod(void) const;`

Returns A pointer to a character string that contains the delivery method set by the `SetDeliveryMethod()` method.

Discussion The `GetDeliveryMethod()` method will return a NULL (zero) value if the delivery method has not already been set by the `SetDeliveryMethod()` method.

Example See "Using the EcxSubmit Class" on page 116.

See Also The `SetDeliveryMethod()` method on page 122.

GetEcxIniFileName( )

Retrieves the full pathname of ECXpert's configuration file set by the `SetEcxIniFileName()` method.

Syntax `virtual const char* GetEcxIniFileName(void) const;`

Returns A pointer to a character string that contains the full pathname of ECXpert's configuration file set by the `SetEcxIniFileName()` method.

Discussion The `GetEcxIniFileName()` method will return a NULL (zero) value if the file name has not already been set by the `SetEcxIniFileName()` method.

Example See "Using the EcxSubmit Class" on page 116.

See Also The `SetEcxIniFileName()` method on page 123.

GetFirstTrackingID( )

Retrieves the tracking ID for the first file in the object's submission list.

Syntax `long GetFirstTrackingID(void);`

Returns A long integer that contains the tracking ID of the first file in the submission list or returns 0 if there are no files in the list.

Discussion The submission list contains references to all the files since you created the object or since the last time you called the object's `ClearFileList()` method. You should only call the `GetFirstTrackingID()` method after you call the `Submit()` method. If you do not first call the `Submit()` method or if it fails, the value returned by calling the `GetFirstTrackingID()` method is undefined.

After you call the object's `GetFirstTrackingID()` method, the tracking ID for the second file in the list will be the next ID to be returned, if the file exists.

Example See "Using the EcxSubmit Class" on page 116.

See Also The `GetNextTrackingID()` method on page 121. The `Submit()` method on page 125.

GetMapName ( )

Retrieves the map name set by the `SetMapName()` method.

Syntax `virtual const char* GetMapName(void) const;`

Returns A pointer to a character string that contains the map name set by the `SetMapName()` method.

Discussion The `GetMapName()` method will return a NULL (zero) value if the map name has not already been set by the `SetMapName()` method.

Example See "Using the EcxSubmit Class" on page 116.

See Also The `SetMapName()` method on page 123.

GetNextTrackingID( )

Retrieves the tracking ID for the next file in the object's submission list.

Syntax `long GetNextTrackingID(void);`

Returns A long integer that contains the tracking ID of the next file in the submission list or returns 0 if there are no more files in the list.

Discussion The submission list contains references to all the files since you created the object or since the last time you called the object's `ClearFileList()` method. You can call the `GetNextTrackingID()` method repeatedly to retrieve the tracking IDs of each file in the list, in the order that you added them.

You should only call the `GetNextTrackingID()` method after you call the `Submit()` method. If you do not first call the `Submit()` method or if it fails, the value returned by calling the `GetNextTrackingID()` method is undefined.

After you call the `GetFirstTrackingID()` method, the `GetNextTrackingID()` method returns the tracking ID for the second file in the list, if it exists. If you call the `GetNextTrackingID()` method after creating the object or after clearing the file list without first calling the object's `GetFirstTrackingID()` method, the `GetNextTrackingID()` method returns the tracking ID of the first file in the list or returns 0 if the list is empty.

Example See "Using the EcxSubmit Class" on page 116.

See Also The `GetFirstTrackingID()` method on page 121. The `Submit()` method on page 125.

GetPassword( )

Retrives the sender's password set by the `SetPassword()` method.

Syntax `virtual const char* GetPassword(void) const;`

Returns A pointer to a character string that contains the sender's password set by the `SetPassword()` method.

Discussion The `GetPassword()` method will return a NULL (zero) value if the sender's password has not already been set by the `SetPassword()` method.

Example See "Using the EcxSubmit Class" on page 116.

See Also The `SetPassword()` method on page 124.

GetRecipient( )

Retrieves the recipient's member ID set by the `SetRecipient()` method.

Syntax `virtual const char* GetRecipient(void) const;`

Returns A pointer to a character string that contains the recipient's member ID set by the `SetRecipient()` method.

Discussion The `GetRecipient()` method will return a NULL (zero) value if the recipient's password has not already been set by the `SetRecipient()` method.

See Also The `SetRecipient()` method on page 124.

GetSender( )

Retrieves the sender's member ID set by the `SetSender()` method.

Syntax `virtual const char* GetSender(void) const;`

Returns A pointer to a character string that contains the sender's member ID set by the `SetSender()` method.

Discussion The `GetSender()` method will return a NULL (zero) value if the sender's password has not already been set by the `SetSender()` method.

Example See "Using the EcxSubmit Class" on page 116.

See Also The `SetSender()` method on page 125.

SetDeliveryMethod( )

Sets the delivery method.

Syntax
```
virtual EcxSubmit& SetDeliveryMethod(const char *
pDeliveryMethod);
```

Parameters The `SetDeliveryMethod()` method has the following parameters:

`pDeliveryMethod`	A pointer to a character string that specifies the delivery method.

Returns A reference to this submission object.

Discussion Call this method if you want to specify the way in which the file was submitted to ECXpert. If you do not call this method, the transport type for this submission is NULL in the database.

Example See "Using the EcxSubmit Class" on page 116.

See Also "Tracking-related Tables" on page 406.

SetEcxIniFileName( )

Sets the full pathname of ECXpert's configuration file.

Syntax `EcxSubmit& SetEcxIniFileName(const char * pIniFileName);`

Parameters The `SetEcxIniFileName()` method has the following parameters:

`pIniFileName`	A pointer to a character string that specifies the configuration file.

Returns A reference to this submission object.

Discussion The configuration file is typically found in the `config` subdirectory from the directory where ECXpert was installed. You must call the `SetEcxIniFileName()` method before you call the `Submit()` method.

Example See "Using the EcxSubmit Class" on page 116.

See Also The `Submit()` method on page 125.

SetMapName( )

Sets the map name.

Syntax `EcxSubmit& SetMapName(const char * pMapName);`

Parameters The `SetMapName()` method has the following parameters:

`pMapName`	A pointer to a character string that contains the map name.

Returns A reference to this submission object.

Discussion Call this method if you want to override the partnership document map name for this submission with the specified map name.

Example See "Using the EcxSubmit Class" on page 116.

SetPassword( )

Sets the sender's password.

Syntax `EcxSubmit& SetPassword(const char * pPassword);`

Parameters The `SetPassword()` method has the following parameters:

`pPassword`	A pointer to a character string that contains the password.

Returns A reference to this submission object.

Discussion A password can contain as many as 60 characters. It can contain letters, numbers, and special characters, and is case sensitive. You must call the `SetPassword()` method before you call the `Submit()` method, unless the sender is trusted member.

Example See "Using the EcxSubmit Class" on page 116.

See Also The `Submit()` method on page 125.

SetRecipient( )

Sets the recipient's member ID.

Syntax `EcxSubmit& SetRecipient(const char * pRecipient);`

Parameters The `SetRecipient()` method has the following parameters:

`pRecipient`	A pointer to a character string that contains the member ID.

Returns A reference to this submission object.

Discussion A member ID can contain as many as 60 characters. It can contain letters, numbers, and special characters, and is case sensitive. You must call the `SetRecipient()` method before you call the `Submit()` method.

Example See "Using the EcxSubmit Class" on page 116.

See Also The `Submit()` method on page 125.

SetSender( )

Sets the sender's member ID.

Syntax `EcxSubmit& SetSender(const char * pSender);`

Parameters The `SetSender()` method has the following parameters:

`pSender`	A pointer to a character string that contains the member ID.

Returns A reference to this submission object.

Discussion A member ID can contain as many as 60 characters. It can contain letters, numbers, and special characters, and is case sensitive. You must call the `SetSender()` method before you call the `Submit()` method.

Example See "Using the EcxSubmit Class" on page 116.

See Also The `Submit()` method on page 125.

Submit( )

Submits objects to ECXpert for processing.

Syntax `EcxSubmit& Submit(int bDataStreaming = FALSE);`

Parameters The `Submit()` method has the following parameters:

`bDataStreaming`	Specify `TRUE` if you want to stream data through a TCP/IP connection; the default is `FALSE`.

Returns A reference to this submission object.

Discussion This method submits one or more files to ECXpert. Before you can submit a file, you must specify the sender and recipient, the sender's password if the sender is not a trusted member, and the ECXpert configuration file.

You must call the methods described on page 115 to set the submission information for the `EcxSubmit` object.

The `bDataStreaming` parameter specifies whether to use a TCP/IP connection to submit the files; set it to `TRUE` to use this kind of connection. The default is `FALSE`, which specifies moving the files after they are on the server. See "About the EcxSubmit Class" on page 113 for more information about streaming versus moving files.

NOTE If you stream data through a TCP/IP connection, the source file is not deleted after the data has been streamed to the server.

If you call the `Submit()` method again, you only need to specify the values that have changed. For example, to submit additional files without changing the sender and receiver, you only need to call the `ClearFileList()` method to remove the current files from the list, call the `AddFile()` method for each file you want to add, and then call the `Submit()` method again to submit the new files.

After you call the object's `Submit()` method, you should immediately check for errors. If an error occurred, none of the files were submitted. The files in the submission list are either all submitted successfully or none of them are submitted.

Example See "Using the EcxSubmit Class" on page 116.

See Also To specify the sender, call the `SetSender()` method on page 125. To specify the recipient, call the `SetRecipient()` method on page 124. To specify the sender's password, call the `SetPassword()` method on page 124. To specify the map file, call the `SetMapName()` method on page 123. To specify the configuration file, call the `SetEcxIniFileName()` method on page 123. To add files, call the `AddFile()` method on page 118. To remove files from the list, call the `ClearFileList()` method on page 119.

Chapter 9

The EcxLogin Class

This chapter describes the `EcxLogin` class, which allows a user to access the database. This chapter contains the following sections:

About the EcxLogin Class

Objects of the `EcxLogin` class represent connections to the database. To log into the database, you can create an `EcxLogin` object and call the object's `Login` method. When you no longer need the connection to the database, you can call the object's `Logout` method.

Methods Summary list:

Constructor and destructor	
`EcxLogin()`	Creates an `EcxLogin` object.
`~EcxLogin()`	Destroys an `EcxLogin` object.
Logging in and out	
`Login()`	Logs into the database.
`Logout()`	Logs out of the database.
Determining the type of member	
`MemberType()`	Determines the type of member currently logged in.

Using the EcxLogin Class

The following example shows how to create an `EcxLogin` object and call the object's `Login` method to create a connection to the database.

```
EcxLogin * login(const char *name, const char *password) {

  EcxLogin *pLogin = NULL;

  if((pLogin = new EcxLogin())->Errnum()) {
    cout << "EcxLogin Object Error:" << endl;
    cout << "\tErrnum: " << pLogin->Errnum() << endl;
    cout << "\tErrmsg: " << pLogin->Errmsg() << endl;
    cout << endl;
    return(NULL);
  }

  if((pLogin->Login(name, password)).Errnum()) {
    cout << "EcxLogin.Login() Failed for user: " << name << endl;
    cout << "\tErrnum: " << pLogin->Errnum() << endl;
    cout << "\tErrmsg: " << pLogin->Errmsg() << endl;
    cout << endl;
    delete pLogin;
    return(NULL);
  }
  return(pLogin);
}
```

EcxLogin Class Reference

Interface `ecxlogin.h`

Superclasses None.

Subclasses `EcxBase`

Friend Classes None.

Syntax `class EcxLogin : public EcxBase { ... };`

Constructor and Destructor

EcxLogin( )

Creates an `EcxLogin` object.

Syntax `EcxLogin(void);`

Example See "Using the EcxLogin Class" on page 128.

~EcxLogin( )

Destroys an `EcxLogin` object.

Syntax `virtual ~EcxLogin();`

Methods

This section describes the methods of the `EcxLogin` class.

Login()

Logs into the database.

Syntax
```
virtual EcxLogin& Login(const char *username, const char
*password);
```

Parameters The `Login()` method has the following parameters:

`username`	A pointer to a character string that represents the user name.
`password`	A pointer to a character string that represents the password.

Returns A pointer to this `EcxLogin` object.

Discussion The user name must match that of a member in the database. If the member is a trusted member, the password in not checked.

Example See "Using the EcxLogin Class" on page 128.

Logout( )

Logs out of the database.

Syntax `virtual EcxLogin& Logout(void);`

Returns A pointer to this `EcxLogin` object.

MemberType( )

Determines the type of member currently logged in.

Syntax `unsigned int MemberType();`

Parameters The `MemberType()` method has the following parameters:

`type`	An unsigned integer that specifies whether the member is an administrator.

Returns An unsigned integer that contains the type of member.

Discussion A type of `ADMINISTRATOR` indicates that the member is an administrator. A type of `MEMBER` indicates that the member is not an administrator. If no member is currently logged in, the `MemberType()` method returns a type of `MEMBER`. The `MemberType()` method does not modify the database.

See also "Class Variables" on page 193.

Chapter 10

The EcxAddresses Class

This chapter describes the `EcxAddresses` class, which defines objects that represent trading addresses. This chapter contains the following sections:

About the EcxAddresses Class

The `EcxAddresses` class represents trading address records in an iPlanet ECXpert database. Administrators can manipulate any address record; non-administrators can only add and delete their own address records. A user must be logged in to the database before accessing a record.

Methods Summary list:

Constructor and destructor	
`EcxAddresses()`	Creates an `EcxAddresses` object.
`~EcxAddresses()`	Destroys an `EcxAddresses` object.
Allowing database access	
`SetLogin()`	Allows the object to access the database.
Adding and deleting address records	
`Add()`	Adds an address record to the database.
`Delete()`	Deletes an address record from the database.

Listing address records	
`List()`	Retrieves a list of address records from the database.
`More()`	Determines whether more records are left in the list.
`Next()`	Associates the object with the next record in the list.
Resetting an object's state	
`Clear()`	Clears the state associated with an object, including its list.
Accessing key fields	
`Member()`	Determines or specifies a member.
`Qual()`	Determines or specifies a member's trading address qualifier.
`QualId()`	Determines or specifies a member's trading address.

Using the EcxAddresses Class

The following example shows how to create an `EcxAddresses` object and set the login to provide database access for the object.

```
BOOL ImportMad::MakeAddressObj()
{
  m_pLogin = new EcxLogin();
  if (m_pLogin == NULL)
  {
    SetGeneralError(INSUFFICIENT_MEMORY, m_fdiscard);
    return FALSE;
  }
  else if (m_pLogin->Errnum())
  {
    PrintEcxMessage("EcxLogin()", m_pLogin, 0, 0);
    m_pLogin = NULL;
    return FALSE;
  }
  if ((m_pLogin->Login(GetUserName(), GetPassword())).Errnum())
  {
    PrintEcxMessage("EcxLogin()", m_pLogin, 0, 0);
    return FALSE;
  }

  m_pAddress = new EcxAddresses();
if (m_pAddress == NULL)
  {
    SetGeneralError(INSUFFICIENT_MEMORY, m_fdiscard);
    return FALSE;
  }
  else if (m_pAddress ->Errnum())
```

```
  {
    PrintEcxMessage("EcxAddresses()", m_pAddress, 0, 0);
    m_pAddress = NULL;
    return FALSE;
  }

  if ((m_pAddress ->SetLogin(*m_pLogin)).Errnum())
  {
    PrintEcxMessage("EcxAddresses()", m_pAddress, 0, 0);
    return FALSE;
  }
  return TRUE;
}
```

EcxAddresses Class Reference

Interface `ecxaddresses.h`

Superclasses `EcxBase`

Subclasses None

Friend Classes None

Syntax `class EcxAddresses : public EcxBase { ... };`

Constructor and Destructor

EcxAddresses( )

Creates an `EcxAddresses` object.

Syntax `EcxAddresses(void);`

`EcxAddresses(EcxLogin& login);`

Discussion The first form of the constructor allows you to create a stack-based object. The second form of the constructor requires that you create an `EcxLogin` object before you create this object.

Example See "Using the EcxAddresses Class" on page 132.

See also The `SetLogin()` method on page 137. The `EcxLogin` class on page 127.

~EcxAddresses( )

Destroys an `EcxAddresses` object.

Syntax `~EcxAddresses(void);`

Discussion The destructor is called when you delete the object. You can reuse an object instead of deleting it by calling the object's `Clear()` method. The destructor does not destroy the associated `EcxLogin` object.

See also The `Clear()` method on page 134.

Methods

This section describes the methods of the `EcxAddresses` class.

Add( )

Adds an address record to the database.

Syntax `EcxAddresses& Add(void);`

Returns A reference to this member object.

Discussion Non-administrators can only add addresses for themselves. Administrators can add addresses for any member. You must specify the member's name in the object, by calling the `Member()` method, before calling the `Add()` method. The combination of qualifier and qualifier ID must be unique for the member.

The parent name and the group-modified-by fields are set to the parent name of the logged-in user; by default, this is 'rootgroup'. The user-modified-by field is set to the name of the logged-in user. Any other fields not specified in the object will become 0 or NULL in the database.

See also The `Member()` method on page 135.

Clear( )

Clears the state associated with an object, including its list.

Syntax `void Clear(void);`

Delete( )

Deletes an address from the database.

Syntax `EcxAddresses& Delete(void);`

Returns A reference to this member object.

Discussion You must be an administrator and be logged in before calling this method.

CAUTION All records whose qualifiers and qualifier IDs match the fields of this object are deleted; the member name is not used.

List( )

Retrieves a list of address records from the database.

Syntax `EcxAddresses& List(void);`

Returns A reference to this member object.

Discussion After calling the `List()` method, the address object contains fields from the first record from the list.

Member( )

Determines or specifies the name of the member.

Syntax `const char* Member() const;`

`void Member(const char* name);`

Parameters The `Member()` method has the following parameters:

`name`	A pointer to a character string that contains the member's name.

Returns The first form of the method returns a pointer to a character string that contains the name.

Discussion Use the first form of the method to determine the member's name. Use the second form to specify the name. The `Member()` method does not modify the database.

More( )

Determines whether more records are left in the list.

Syntax `long More(void);`

Returns A long integer that contains the number of records not yet accessed from the list.

Discussion After calling the `List()` method and before calling the `Next()` method, the `More()` method returns the total number of records in the list. All records have been accessed when the `More()` method returns 0.

Next( )

Associates the object with the next record in the list.

Syntax `EcxAddresses& Next(void);`

Returns A reference to this member object.

Discussion The `Next()` method sets the fields in the object to match those in the next record in the list. The `Next()` method decrements the number of records not yet accessed, which is returned by the `More()` method.

CAUTION Do not call the `Next()` method if the `More()` method returns a value less than 1; the results are unpredictable.

See also The `More()` method on page 135.

Qual( )

Determines or specifies a member's trading address qualifier.

Syntax

```
const char* Qual() const;
void Qual(const char* qualifier);
```

Parameters The `Qual()` method has the following parameters:

`qualifier`	A pointer to the character string that contains the qualifier.

Returns The first form of the method returns a pointer to a character string that contains the qualifier.

Discussion Use the first form of the method to determine the qualifier. Use the second form to specify the qualifier. The `Qual()` method does not modify the database.

QualId( )

Determines or specifies a member's trading address.

Syntax `const char* QualId() const;`

`void QualId(const char* id);`

Parameters The `QualId()` method has the following parameters:

`id`	A pointer to the character string that contains the trading address.

Returns The first form of the method returns a pointer to a character string that contains the trading address.

Discussion Use the first form of the method to determine the trading address. Use the second form to specify the trading address. The `QualId()` method does not modify the database.

SetLogin( )

Allows the object to access the database.

Syntax `EcxAddresses& SetLogin(EcxLogin& login);`

Parameters The `SetLogin()` method has the following parameters:

`login`	A reference to a valid `EcxLogin` object

Returns A reference to this member object.

Discussion If you do not use the form of the constructor that accepts a login object, you must call the `SetLogin()` method before accessing this object.

See also The `EcxAddresses` constructor on page 133. The `EcxLogin` class on page 127.

~EcxAddresses( )

Chapter 11

Partnership-Related Classes

This chapter describes the `EcxPartnership` class, which represents a view of partnership records and related standards information, group, and document information records in an iPlanet ECXpert database. This chapter also describes the `EcxPartnerId` class, which represents key values for `EcxPartnership` objects. This chapter contains the following sections:

About the EcxPartnership Class

The `EcxPartnership` class represents a view on the following kinds of records in an iPlanet ECXpert database:

- partnerships
- EDI standards information
- partnership groups
- document types

A record in the view represents a partnership record whose ID matches a standards information ID, a group ID and a document type ID and whose group type matches the document type.

Only administrators can add, change, or delete records using this view. An administrator can retrieve any record from the view; a non-administrator can only retrieve records from the view that includes the user as either a sender or receiver. A user must be logged in to the database before accessing a record through the view.

Methods Summary list:

Constructor and destructor	
`EcxPartnership()`	Creates an `EcxPartnership` object.
`~EcxPartnership()`	Destroys an `EcxPartnership` object.
Allowing database access	
`SetLogin()`	Allows the object to access the database.
Adding, retrieving, changing and deleting partnership view-related records	
`Add()`	Adds partnership view-related records to the database.
`Get()`	Retrieves partnership view-related records from the database.
`Change()`	Changes partnership view-related records in the database.
`Delete()`	Deletes partnership view-related records from the database.
Listing partnership records	
`List()`	Retrieves a list of partnership view-related records from the database.
`More()`	Determines whether more records are left in the list.
`Next()`	Associates the object with the next record in the list.
Resetting an object's state	
`Clear()`	Clears the state associated with an object, including its list.
Accessing key fields	
`PartnerId()`	Determines or specifies the partnership ID.
`DocType()`	Determines or specifies the kind of EDI document.

`GroupType()`	Determines or specifies the kind of EDI documents in the group.
Accessing partnership information	
`SenderName()`	Determines or specifies the sender's member name.
`SenderQual()`	Determines or specifies the sender's trading address qualifier.
`SenderQualId()`	Determines or specifies the sender's trading address.
`SenderCertificateType()`	Determines or specifies the sender's certificate type.
`ReceiverName()`	Determines or specifies the receiver's member name.
`ReceiverQual()`	Determines or specifies the receiver's trading address qualifier.
`ReceiverQualId()`	Determines or specifies the receiver's trading address
`ReceiverCertificateType()`	Determines or specifies the receiver's certificate type.
`Active()`	Determines or specifies whether the partnership is active.
`Security()`	Determines or specifies the kind of security.
`Description()`	Determines or specifies the partnership's description.
Accessing standards information	
`StandardName()`	Determines or specifies the name of the EDI standard.
`StandardVersion()`	Determines or specifies the standard's version number.
`StandardRelease()`	Determines or specifies the standard's release number.
`IntchngLastControlNumber()`	Determines or specifies the last interchange control number generated.
`IntchngLock()`	Determines or specifies whether the document has been read at the interchange level.

`IntchngGenerateAck()`	Determines or specifies whether to generate interchange acknowledgments flags.
`IntchngAckWaitPeriod()`	Determines or specifies the number of minutes to wait before the acknowledgment becomes overdue.
`TestProductionFlag()`	Determines or specifies whether the partnership is used for testing or production.
`SegmentTerminator()`	Determines or specifies the segment terminator character.
`ElementSeparator()`	Determines or specifies the data element terminator character.
`SubElementSeparator()`	Determines or specifies the data subelement terminator character.
`DecimalPointCharacter()`	Determines or specifies the decimal point character.
`ReleaseCharacter()`	Determines or specifies the release character.
`OutStandard()`	Determines or specifies the interchange standard user wishes to appear in bundled EDI documents.
`OutVersion()`	Determines or specifies the interchange version user wishes to appear in bundled EDI documents
`OutRelease()`	Determines or specifies the interchange release user wishes to appear in bundled EDI documents.
`GenOptEnv()`	Determines or specifies the enveloping options.
Accessing group information	
`GroupLastControlNumber()`	Determines or specifies the last group control number generated.
`GroupLock()`	Determines or specifies whether the document has been read at the group level.
`GroupGenerateDocAck()`	Determines or specifies the to generate group acknowledgments flags
`SndrAppQual()`	Determines or specifies the sending member main trading address.
`SndrAppCode()`	Determines or specifies the application sender code.
`RcvrAppQual()`	Determines or specifies the receiving member main trading address.

`RcvrAppCode()`	Determines or specifies the application receiver code.
Accessing document type specific information	
`DocPriority()`	Determines or specifies the document processing priority.
`MapName()`	Determines or specifies the map file name.
`MapDirection()`	Determines or specifies the document translation type.
`AckExpected()`	Determines or specifies the number of minutes to wait before an acknowledgment becomes overdue.
`DocLastControlNumber()`	Determines or specifies the last document control number generated.
`DocLock()`	Determines or specifies whether the document has been read.
`PrimaryXportType()`	Determines or specifies the primary transport protocol.
`PrimaryXportParam()`	Determines or specifies the primary transport protocol parameter.
`SecondaryXportType()`	Determines or specifies the secondary transport protocol.
`SecondaryXportParam()`	Determines or specifies the secondary transport protocol parameter.
`SendType()`	Determines or specifies when the document is to be sent.
`DeleteWaitPeriod()`	Determines or specifies the number of days to retain documents before deleting them.
`ArchiveWaitPeriod()`	Determines or specifies the number of days to retain documents before archiving them.
`PreEnveloped()`	Determines or specifies whether documents are preenveloped.

Using the EcxPartnership Class

The following sections show how to

- create partnership objects
- add partnerships to the database
- list partnerships in the database
- delete partnerships from the database

Creating Partnership Objects

The following example shows how to create an `EcxPartnership` object and how to allow access to the database by calling the object's `SetLogin()` method:

```
EcxPartnership * make_partnershipobj(EcxLogin * pLogin) {

  EcxPartnership * pPartnership = NULL;

  if((pPartnership = new EcxPartnership())->Errnum()) {
    cout << "EcxPartnership Object Error:" << endl;
    cout << "\tErrnum: " << pPartnership->Errnum() << endl;
    cout << "\tErrmsg: " << pPartnership->Errmsg() << endl;
    cout << endl;
    return(NULL);
  }

  if((pPartnership->SetLogin(*pLogin)).Errnum()) {
    cout << "EcxPartnership.SetLogin() Failed:" << endl;
    cout << "\tErrnum: " << pPartnership->Errnum() << endl;
    cout << "\tErrmsg: " << pPartnership->Errmsg() << endl;
    cout << endl;
    delete pPartnership;
    return(NULL);
  }

  return(pPartnership);
}
```

Alternatively, you can pass the login object to the `EcxPartnership` constructor without having to call `SetLogin()`.

Adding Partnerships

The following example shows how to add records associated with a partnership view to the database. An administrator's login must be associated with the object you want to add.

```
int add_partnership(EcxPartnership *pPartnership,
           const char *name1,
           const char *name2,
           const char *doctype) {

  pPartnership->Clear();
  pPartnership->SenderName(name1);
  pPartnership->SenderQual("NONE");
  pPartnership->SenderQualId(name1);
  pPartnership->ReceiverName(name2);
  pPartnership->ReceiverQual("NONE");
  pPartnership->ReceiverQualId(name2);
  pPartnership->StandardName("X");
  pPartnership->StandardVersion("3");
  pPartnership->StandardRelease("0");
  pPartnership->GroupType("FF");
  pPartnership->DocType(doctype);
  pPartnership->Active(TRUE);

  if((pPartnership->Add()).Errnum()) {
    cout << "EcxPartnership.add() Failed for :";
    cout << name1 << ":" << name2 << ":" << doctype << ":" <<
endl;
    cout << "\tErrnum: " << pPartnership->Errnum() << endl;
    cout << "\tErrmsg: " << pPartnership->Errmsg() << endl;
    return(pPartnership->Errnum());
  }

  cout << "*** Added partnership :";
  cout << name1 << ":" << name2 << ":" << doctype << ":" << endl;

  return(0);
}
```

Listing Partnerships

The following example shows how to retrieve records for a list of views. In this example, all view-related records are retrieved for administrators. For non-administrators, this example retrieves all view-related records for views in which the user is either the sender or receiver. The following rules apply to the `List()` method, as well:

- If neither the sender or receiver is specified, the `List()` method retrieves all view-related records for views in which the user is either the sender or receiver.
- If only the sender is specified, the `List()` method retrieves all view-related records for views in which the user is the sender.
- If only the receiver is specified, the `List()` method retrieves all view-related records for views in which the user is the receiver.
- If both the sender and receiver are specified, the `List()` method retrieves all view-related records for views that match both the sender and receiver; in which case, the user must be either the receiver or sender.

```
int list(EcxPartnership *pPartnership) {

  pPartnership->Clear();

  if((pPartnership->List()).Errnum()) {
    cout << "EcxPartnership.List() Failed:" << endl;
    cout << "\tErrnum: " << pPartnership->Errnum() << endl;
    cout << "\tErrmsg: " << pPartnership->Errmsg() << endl;
    return(pPartnership->Errnum());
  }

  cout << "*** Listing partnerships" << pPartnership->More();
  cout << " records found. ***" << endl;

  while(pPartnership->More()) {
    cout << pPartnership->SenderName()      << ":";
    cout << pPartnership->ReceiverName()    << ":";
    cout << pPartnership->DocType()         << ":";
    cout << pPartnership->StandardName()    << ":";
    cout << pPartnership->StandardVersion() << ":";
    cout << pPartnership->GroupType()       << endl;

    pPartnership->Next();
  }

  return(0);
}
```

The following example shows how to retrieve records for two lists of views. The sender is used to filter the first list. The receiver is used to filter the second list. For administrators, the example shows how to retrieve all view-related records that match the respective sender and receiver. For non-administrators, the example shows how to retrieve these records as long as the user is the sender in the first list and the receiver in the second list.

CAUTION For non-administrators, calling the `List()` method in this example mutates the sender or receiver name to match the user name if the names do not already match.

```
int list_member(EcxPartnership *pPartnership, const char *uname) {

  pPartnership->Clear();

  pPartnership->SenderName(uname);

  if((pPartnership->List()).Errnum()) {
    cout << "EcxPartnership.List(" << uname << ",NULL) Failed:" << endl;
    cout << "\tErrnum: " << pPartnership->Errnum() << endl;
    cout << "\tErrmsg: " << pPartnership->Errmsg() << endl;
    return(pPartnership->Errnum());
  }

  cout << "*** Listing partnerships where sender is " << uname;
  cout << ". " << pPartnership->More() << " records found. ***" << endl;

  while(pPartnership->More()) {
    cout << pPartnership->SenderName()      << ":";
    cout << pPartnership->ReceiverName()    << ":";
    cout << pPartnership->DocType()         << ":";
    cout << pPartnership->StandardName()    << ":";
    cout << pPartnership->StandardVersion() << ":";
    cout << pPartnership->GroupType()       << endl;

    pPartnership->Next();
  }

  pPartnership->Clear();

  pPartnership->ReceiverName(uname);

  if((pPartnership->List()).Errnum()) {
    cout << "EcxPartnership.List(NULL," << uname << ") Failed:" << endl;
    cout << "\tErrnum: " << pPartnership->Errnum() << endl;
    cout << "\tErrmsg: " << pPartnership->Errmsg() << endl;
    return(pPartnership->Errnum());
  }
```

```
  cout << "*** Listing partnerships where receiver is " << uname;
  cout << ". " << pPartnership->More() << " records found. ***" << endl;

  while(pPartnership->More()) {
    cout << pPartnership->SenderName()      << ":";
    cout << pPartnership->ReceiverName()    << ":";
    cout << pPartnership->DocType()         << ":";
    cout << pPartnership->StandardName()    << ":";
    cout << pPartnership->StandardVersion() << ":";
    cout << pPartnership->GroupType()       << endl;

    pPartnership->Next();
  }

  cout << endl;

  return(0);
}
```

Deleting Partnerships

The following example shows how to delete the records associated with a partnership view from the database. All records matching the specified sender name, receiver name, and document type are deleted. An administrator's login must be associated with the object you want to delete.

```
int del_partnership(EcxPartnership *pPartnership,
          const char *name1,
          const char *name2,
          const char *doctype) {

  pPartnership->Clear();
  pPartnership->SenderName(name1);
  pPartnership->ReceiverName(name2);
  pPartnership->DocType(doctype);

  if((pPartnership->Delete()).Errnum()) {
    cout << "EcxPartnership.Delete() Failed for : ";
    cout << name1 << ":" << name2 << ":" << doctype << ":" <<
endl;
    cout << "\tErrnum: " << pPartnership->Errnum() << endl;
    cout << "\tErrmsg: " << pPartnership->Errmsg() << endl;
    return(pPartnership->Errnum());
  }

  cout << "*** Deleted partnership :";
  cout << name1 << ":" << name2 << ":" << doctype << ":" << endl;
```

```
  return(0);
}
```

EcxPartnership Class Reference

Interface `ecxpartnership.h`

Superclasses `EcxBase`

Subclasses None

Friend Classes None

Syntax `class EcxPartnership : public EcxBase { ... };`

Class Variables

The following class variables allow you to identify the member as either an administrator or an ordinary member:

Syntax `static int SENDTYPE_UNKNOWN;`

```
static int SENDTYPE_IMMEDIATE;
static int SENDTYPE_ONETIME;
static int SENDTYPE_PERIODIC;
static int SECURITY_PLAIN;
static int SECURITY_ENCRYPTED;
static int SECURITY_SIGNED;
static int SECURITY_SIGNEDANDENCRYPTED;
static int PRIORITY_UNKNOWN;
static int PRIORITY_HIGH;
static int PRIORITY_MEDIUM;
static int PRIORITY_LOW;
static int CERTTYPE_UNKNOWN;
static int CERTTYPE_SELF;
```

```
static int CERTTYPE_VERISIGN1;
static int CERTTYPE_VERISIGN2;
static int CERTTYPE_VERISIGN3;
static int ENVELOPE_UNKNOWN;
static int ENVELOPE_NONE;
static int ENVELOPE_REGULAR;
static int ENVELOPE_EDI;
static int XLATTYPE_UNKNOWN;
static int XLATTYPE_INBOUND;
static int XLATTYPE_OUTBOUND;
static int XLATTYPE_EDI2EDI;
static int XLATTYPE_APP2APP;
static int XLATTYPE_NONE;
```

SENDTYPE_UNKNOWN	Unknown send type.
SENDTYPE_IMMEDIATE	Send immediately.
SENDTYPE_ONETIME	Send once.
SENDTYPE_PERIODIC	Send periodically.
SECURITY_PLAIN	No security; base-64 encoding only.
SECURITY_ENCRYPTED	Encrypted with receiver's public key.
SECURITY_SIGNED	Signed with sender's private key.
SECURITY_SIGNEDANDENCRYPTED	Signed with sender's private key, then encrypted with receiver's public key.
PRIORITY_UNKNOWN	Unknown priority.
PRIORITY_HIGH	High priority.
PRIORITY_MEDIUM	Medium priority.
PRIORITY_LOW	Low priority.
CERTTYPE_UNKNOWN	Unknown certificate type.
CERTTYPE_SELF	Self-signed certificate type.
CERTTYPE_VERISIGN1	VeriSign class-1 certificate type.
CERTTYPE_VERISIGN2	VeriSign class-2 certificate type.

CERTTYPE_VERISIGN3	VeriSign class-3 certificate type.
ENVELOPE_UNKNOWN	Unknown envelope status for document.
ENVELOPE_NONE	No envelope for document.
ENVELOPE_REGULAR	Enveloped document.
ENVELOPE_EDI	Preenveloped EDI document.
XLATTYPE_UNKNOWN	Unknown translation.
XLATTYPE_INBOUND	EDI-to-application translation.
XLATTYPE_OUTBOUND	Application-to-EDI translation
XLATTYPE_EDI2EDI	EDI-to-EDI translation.
XLATTYPE_APP2APP	Application-to-application translation.
XLATTYPE_NONE	No translation; passthrough mode.

Constructor and Destructor

EcxPartnership( )

Creates an `EcxPartnership` object.

Syntax `EcxPartnership(void);`

`EcxPartnership(EcxLogin& login);`

Parameters The constructor has the following parameters:

`login`	The login object to associate with this partnership object.

Discussion The first form of the constructor allows you to create a stack-based object. The second form of the constructor requires that you create an `EcxLogin` object before you create this object.

Example See "Creating Partnership Objects" on page 144.

See also The `SetLogin()` method on page 179. The `EcxLogin` class on page 127.

~EcxPartnership( )

Destroys an `EcxPartnership` object.

Syntax `virtual ~EcxPartnership(void);`

Discussion The destructor is called when you delete the object. You can reuse an object instead of deleting it by calling the object's `Clear()` method. The destructor does not destroy the associated `EcxLogin` object.

See also The `Clear()` method on page 155.

Methods

This section describes the methods of the `EcxPartnership` class.

AckExpected( )

Determines or specifies the number of minutes to wait before an acknowledgment becomes overdue.

Syntax `unsigned int AckExpected() const;`

```
void AckExpected ( const unsigned int& minutes );
```

Parameters The `AckExpected()` method has the following parameters:

`minutes`	An unsigned integer that specifies the number of minutes.

Returns The first form of the method returns an unsigned integer that contains the number of minutes to wait before an acknowledgment becomes overdue.

Discussion Use the first form of the method to determine the number of minutes to wait before an acknowledgment becomes overdue. Use the second form to specify the number of minutes. The `AckExpected()` method does not modify the database.

Active( )

Determines or specifies whether the partnership is active.

Syntax `unsigned int Active() const;`

```
void Active(const unsigned int status);
```

Parameters The `Active()` method has the following parameters:

`status`	An unsigned integer that specifies whether the partnership is active.

Returns The first form of the method returns an unsigned integer that contains the status.

Discussion Use the first form of the method to determine whether the partnership is active. Use the second form to specify whether the partnership is active. A status of TRUE (1) indicates that the partnership is active. A status of FALSE (0) indicates that the partnership is inactive. The `Active()` method does not modify the database.

Example "Adding Partnerships" on page 145.

`Add()`

Adds partnership view-related records to the database.

Syntax `EcxPartnership& Add(void);`

Returns A reference to this partnership object.

Discussion The `Add()` method adds a partnership record and its related standards information, group, and document information records to the database. The `Add()` method sets the partnership ID in the database and the partnership object.

You must be an administrator and be logged in before calling this method. You must specify the sender name, receiver name, qualifier, qualifier ID, group type, document type, EDI standard, and the standard's release and version numbers in the object, before calling the `Add()` method.

The group-modified-by and user-modified-by fields are set to the group and name of the logged-in user, respectively. Acknowledgment wait periods are set to MINUTES_IN_10_YEARS. Any other fields not specified in the object will become 0 or NULL in the database.

Example See "Adding Partnerships" on page 145.

See also The `SenderName()` method on page 176. The `SenderQual()` method on page 177. The `SenderQualID()` method on page 177. The `ReceiverName()` method on page 171. The `ReceiverQual()` method on page 172. The `ReceiverQualID()` method on page 172. The `GroupType()` method on page 161. The `DocType()` method on page 159.

ArchiveWaitPeriod( )

Determines or specifies the number of days to retain documents before archiving them.

Syntax `unsigned int ArchiveWaitPeriod() const;`

`void ArchiveWaitPeriod ( const unsigned int& days );`

Parameters The `ArchiveWaitPeriod()` method has the following parameters:

`days`	An unsigned integer that specifies the number of days.

Returns The first form of the method returns an unsigned integer that contains the number of days to retain documents before archiving them.

Discussion Use the first form of the method to determine the number of days to retain documents before archiving them. Use the second form to specify the number of days. The `ArchiveWaitPeriod()` method does not modify the database.

Change( )

Changes partnership view-related records in the database.

Syntax `EcxPartnership& Change(void);`

Returns A reference to this partnership object.

Discussion You must be an administrator and be logged in before calling this method. This method updates the last record retrieved by calling the object's `Get()`, `List()`, or `Next()` method. Only administrators may call the `Change()` method. The group-modified-by and user-modified-by fields are set to the group and name of the logged-in user, respectively. Acknowledgment wait periods are set to MINUTES_IN_10_YEARS. Any other fields not specified in the object will become 0 or NULL in the database.

CAUTION If you do not call the object's `Get()`, `List()`, or `Next()` method first, the object's Partnership ID field, which is set by calling the `PartnerID()` method, specifies the records to change. In this case, the records are completely overwritten using the object's fields. Any fields not set in the object will be replaced by 0 or NULL in the database.

See also The `Get()` method on page 160. The `List()` method on page 164. The `Next()` method on page 166. The `PartnerID()` method on page 167.

Clear( )

Clears the state associated with an object, including its list.

Syntax `void Clear(void);`

Discussion All fields in the object are reset to 0 or NULL. A list contains no records.

Example See "Listing Partnerships" on page 146.

DecimalPointCharacter( )

Determines or specifies the decimal point character.

Syntax

```
const char* DecimalPointCharacter() const;
void DecimalPointCharacter ( const char* decPt );
```

Parameters The `DecimalPointCharacter()` method has the following parameters:

`decPt`	A pointer to a character string that contains the decimal point character.

Returns The first form of the method returns a pointer to a character string that contains the decimal point character.

Discussion Use the first form of the method to determine the decimal point character. Use the second form to specify the decimal point character. The `DecimalPointCharacter()` method does not modify the database.

Delete( )

Deletes partnership view-related records from the database.

Syntax `EcxPartnership& Delete(void);`

Returns A reference to this partnership object.

Discussion You must be an administrator and be logged in before calling this method. After this method executes, the object is reset; fields of the object are reset to 0 or NULL. A list contains no records. The partnership record is deleted from the database. Dangling standards information, group, and document information records, which are those records that no longer reference other records in the database, are also deleted.

CAUTION You should call the object's `Get()`, `List()`, or `Next()` method before calling the `Delete()` method to ensure that the intended records are deleted.

Example See "Deleting Partnerships" on page 148.

See also The `Get()` method on page 160. The `List()` method on page 164. The `Next()` method on page 166.

DeleteWaitPeriod( )

Determines or specifies the number of days to retain documents before deleting them.

Syntax

```
unsigned int DeleteWaitPeriod() const;
void DeleteWaitPeriod ( const unsigned int& flag );
```

Parameters The `DeleteWaitPeriod()` method has the following parameters:

`days`	An unsigned integer that specifies the number of days.

Returns The first form of the method returns an unsigned integer that contains the number of days to retain documents before deleting them.

Discussion Use the first form of the method to determine the number of days to retain documents before deleting them. Use the second form to specify the number of days. The `DeleteWaitPeriod()` method does not modify the database.

Description( )

Determines or specifies the partnership's description.

Syntax

```
const char* Description() const;
void Description ( const char* description );
```

Parameters The `Description()` method has the following parameters:

`desc`	A pointer to a character string that contains the description.

Returns The first form of the method returns a pointer to a character string that contains the description.

Discussion Use the first form of the method to determine the description. Use the second form to specify the description. The `Description()` method does not modify the database.

DocLastControlNumber( )

Determines or specifies the last document control number generated.

Syntax

```
const char* DocLastControlNumber() const;
void DocLastControlNumber ( const char* controlNumber );
```

Parameters The `DocLastControlNumber()` method has the following parameters:

`controlNumber`	A pointer to a character string that contains the control number.

Returns The first form of the method returns a pointer to a character string that contains the control number.

Discussion Use the first form of the method to determine the control number. Use the second form to specify the control number. The `DocLastControlNumber()` method does not modify the database.

DocLock( )

Determines or specifies whether or not the document has been read at the document level.

Syntax
```
unsigned int DocLock() const;
void DocLock( const unsigned int& );
```

Returns The first form of the method returns an unsigned integer that specifies whether or not the submission has been read at the document level.

Example See "Using the EcxPartnership Class" on page 144.

DocPriority( )

Determines or specifies the document processing priority.

Syntax
```
unsigned int  DocPriority() const;
void DocPriority ( const unsigned int& priority );
```

Parameters The `DocPriority()` method has the following parameters:

`priority`	An unsigned integer that specifies the priority.

Returns The first form of the method returns an unsigned integer that contains the priority.

Discussion Use the first form of the method to determine the priority. Use the second form to specify the priority. The `DocPriority()` method does not modify the database.

You can use any of the following values:

Constant	Value
`PRIORITY_UNKNOWN`	0
`PRIORITY_HIGH`	1
`PRIORITY_MEDIUM`	2
`PRIORITY_LOW`	3

See also "Class Variables" on page 149.

DocType( )

Determines or specifies the kind of EDI document.

Syntax `const char* DocType() const;`

`void DocType ( const char* type );`

Parameters The `DocType()` method has the following parameters:

`type`	A pointer to a character string that contains the document type.

Returns The first form of the method returns a pointer to a character string that contains the document type.

Discussion Use the first form of the method to determine the type. Use the second form to specify the type. The `DocType()` method does not modify the database.

Example See "Adding Partnerships" on page 145. See "Listing Partnerships" on page 146.

ElementSeparator( )

Determines or specifies the data element terminator character.

Syntax `const char* ElementSeparator() const;`

`void ElementSeparator ( const char* separator );`

Parameters The `ElementSeparator()` method has the following parameters:

`separator`	A pointer to a character string that contains the terminator character.

Returns The first form of the method returns a pointer to a character string that contains the terminator character.

Discussion Use the first form of the method to determine the terminator character. Use the second form to specify the terminator character. The `ElementSeparator()` method does not modify the database.

GenOptEnv ( )

Determines or specifies the enveloping options.

Syntax `unsigned int GenOptEnv() const;`

`void GenOptEnv(const unsigned int& );`

Returns The first form of the method returns an unsigned integer that specifies the enveloping options.

Discussion You can use any of the following values:

Constant	Value
`No UNA, No UNG`	0
`UNA only`	1
`UNG only`	2
`UNA and UNG`	3

Example See "Using the EcxPartnership Class" on page 144.

Get( )

Retrieves partnership view-related records from the database.

Syntax `EcxPartnership& Get(EcxPartnerId& prntnrid);`

Parameters The `Get()` method has the following parameters:

`prntnrid`	A reference to an `EcxPartnerId` that specifies the partnership.

Returns A reference to this partnership object.

Discussion Administrators may retrieve records for any view. Non-administrators can only retrieve records for views in which either the sender or receiver member name matches the user's login name. You call the partnership ID object's `SetValues()` method to specify the view whose records you wish to retrieve.

If you wish use the `Get()` method to retrieve a specific partnership, you must first construct an instance of `EcxPartnerId()` with the proper keys, such as partner ID, standard ID, etc. An easier way to retrieve a partnership would be to use the `List()` method. You may use the `List()` method to list the partnership by sender name and receiver name. If the user is logged in as an administrator, the user can list any partnership by setting the sender name and receiver name. If the user is not logged in as an administrator, the user can only list the partnership that the user belongs to, meaning the partnership with the logged in user either as the sender or receiver.

See also The `EcxPartnerId::SetValues()` method on page 185. The `List()` method on page page 164.

GroupGenerateDocAck( )

Specifies whether to generate an acknowledgement for the submission at the group level.

Syntax
```
unsigned int GroupGenerateDocAck() const;
void GroupGenerateDocAck( const unsigned int& );
```

Returns The first form of the method returns an unsigned integer that indicates whether or not to generate an acknowledgement for the submission at the group level.

Example See "Using the EcxPartnership Class" on page 144.

GroupLastControlNumber( )

Determines or specifies the last group control number generated.

Syntax
```
const char* GroupLastControlNumber() const;
void GroupLastControlNumber ( const char* controlNumber );
```

Parameters The `GroupLastControlNumber()` method has the following parameters:

`controlNumber`	A pointer to a character string that contains the control number.

Returns The first form of the method returns a pointer to a character string that contains the control number.

Discussion Use the first form of the method to determine the control number. Use the second form to specify the control number. The `GroupLastControlNumber()` method does not modify the database.

GroupLock( )

Determines or specifies whether the document has been read at the group level.

Syntax
```
unsigned int GroupLock() const;
void GroupLock (const unsigned int& )
```

Returns The first form of the method returns an unsigned integer that indicates whether or not the document has been read at the group level.

Example See "Using the EcxPartnership Class" on page 144.

GroupType( )

Determines or specifies the kind of EDI documents in the group.

Syntax
```
const char* GroupType() const;
void GroupType ( const char* type );
```

Parameters The `GroupType()` method has the following parameters:

`type`	A pointer to a character string that contains the group type.

Returns The first form of the method returns a pointer to a character string that contains the group type.

Discussion Use the first form of the method to determine the type. Use the second form to specify the type. The `GroupType()` method does not modify the database.

Example See "Adding Partnerships" on page 145. See "Listing Partnerships" on page 146.

IntchngAckWaitPeriod( )

Determines or specifies the number of minutes to wait before the acknowledgment becomes overdue.

Syntax
```
unsigned int IntchngAckWaitPeriod() const;
void IntchngAckWaitPeriod ( const unsigned int& period );
```

Parameters The `IntchngAckWaitPeriod()` method has the following parameters:

`period`	An unsigned integer that specifies the number of minutes to wait.

Returns The first form of the method returns an unsigned integer that contains the number of minutes to wait before an acknowledgment becomes overdue.

Discussion Use the first form of the method to determine the number of minutes to wait before an acknowledgment becomes overdue. Use the second form to specify the number of minutes. The `IntchngAckWaitPeriod()` method does not modify the database.

IntchngLastControlNumber( )

Determines or specifies the last interchange control number generated.

Syntax
```
const char* IntchngLastControlNumber() const;
void IntchngLastControlNumber ( const char* controlNumber );
```

Parameters The `IntchngLastControlNumber()` method has the following parameters:

`controlNumber`	A pointer to a character string that contains the control number.

Returns The first form of the method returns a pointer to a character string that contains the control number.

Discussion Use the first form of the method to determine the control number. Use the second form to specify the control number. The `IntchngLastControlNumber()` method does not modify the database.

IntchngGenerateAck( )

Specifies whether to generate an acknowledgement at the interchange level.

Syntax
```
unsigned int IntchngGenerateAck() const;
void IntchngGenerateAck (const unsigned int& )
```

Returns An unsigned integer that specifies whether to generate an acknowledgement at the interchange level.

Example See "Using the EcxPartnership Class" on page 144.

IntchngLock( )

Determines or specifies whether the document has ben read at the interchange level.

Syntax
```
unsigned int IntchngLock() const;
void IntchngLock (const unsigned int& )
```

Returns An unsigned integer that specifies whether the document has been read at the interchange level.

Example See "Using the EcxPartnership Class" on page 144.

List( )

Retrieves a list of partnership view-related records from the database.

Syntax
```
EcxPartnership& List(const char* partner = NULL);
```

Parameters The `List()` method has the following parameters:

`partner`	A pointer to a character string that contains the name of the receiving member or NULL if not specified.

Returns A reference to this partnership object.

Discussion Administrators may retrieve records for any view. Non-administrators can only retrieve records for views in which either the sender or receiver member name matches the user's login name. The views retrieved for non-administrators depend on whether the sender or receiver member names are specified in the partnership object:

- If neither the sender or receiver is specified, the List() method retrieves all view-related records for views in which the user is either the sender or receiver.
- If only the sender is specified, the `List()` method retrieves all view-related records for views in which the user is the sender.
- If only the receiver is specified, the `List()` method retrieves all view-related records for views in which the user is the receiver.

- If both the sender and receiver are specified, the `List()` method retrieves all view-related records for views that match both the sender and receiver; in which case, the user must be either the receiver or sender.

You can restrict the views, and thus the records that are retrieved, by specifying a partnership in the `partner` parameter. In this case, the `List()` method uses only views that match both the specified partner and user as either the sender or receiver.

If you wish use the `Get()` method to retrieve a specific partnership, you must first construct an instance of `EcxPartnerId()` with the proper keys, such as partner ID, standard ID, etc. An easier way to retrieve a partnership would be to use the `List()` method. You may use the `List()` method to list the partnership by sender name and receiver name. If the user is logged in as an administrator, the user can list any partnership by setting the sender name and receiver name. If the user is not logged in as an administrator, the user can only list the partnership that the user belongs to, meaning the partnership with the logged in user either as the sender or receiver.

CAUTION If only the sender or receiver is specified for a non-administrator, the `List()` method mutates the sender or receiver name to match the user name if the respective name (sender or receiver) does not match the user name.

After calling the `List()` method, the partnership object's fields contain values from the records related to the first partnership view in the list.

Example See "Listing Partnerships" on page 146.

See Also The `Get()` method on page 160.

MapName( )

Determines or specifies the map file name.

Syntax

```
const char* MapName() const;
void MapName ( const char* map );
```

Parameters The `MapName()` method has the following parameters:

`map`	A pointer to a character string that contains the map name.

Returns The first form of the method returns a pointer to a character string that contains the map name.

Discussion Use the first form of the method to determine the map name. Use the second form to specify the map name. The `MapName()` method does not modify the database.

More( )

Determines whether more records are left in the list.

Syntax `long More(void);`

Returns A long integer that contains the number of records not yet accessed from the list.

Discussion After calling the `List()` method and before calling the `Next()` method, the `More()` method returns the total number of records in the list. All records have been accessed when the `More()` method returns 0.

Example See "Listing Partnerships" on page 146.

See also The `List()` method on page 164. The `Next()` method on page 166.

Next( )

Associates the object with the next record in the list.

Syntax `EcxPartnership& Next(void);`

Returns A reference to this partnership object.

Discussion The `Next()` method sets the fields in the object to match those in the next record in the list. The `Next()` method decrements the number of records not yet accessed, which is returned by the `More()` method.

CAUTION Do not call the `Next()` method if the `More()` method returns a value less than 1; the results are unpredictable.

Example See "Listing Partnerships" on page 146.

See also The `More()` method on page 165.

OutRelease( )

Determines or specifies the interchange release the user wishes to appear in bundled EDI documents.

Syntax `const char* OutVersion() const;`

`void OutVersion (const char* );`

Returns The first form of the method returns a pointer to a character string that contains the interchange release the user wishes to appear in bundled EDI documents.

Discussion Use the first form of the method to determine interchange release the user wishes to appear in bundled EDI documents. Use the second form to specify the interchange release the user wishes to appear in bundled EDI documents.

OutStandard( )

Determines or specifies the interchange standard the user wishes to appear in bundled EDI documents.

Syntax `const char* OutVersion() const;`

`void OutVersion (const char* );`

Returns The first form of the method returns a pointer to a character string that contains the interchange standard the user wishes to appear in bundled EDI documents.

Discussion Use the first form of the method to determine interchange standard the user wishes to appear in bundled EDI documents. Use the second form to specify the interchange standard the user wishes to appear in bundled EDI documents.

OutVersion( )

Determines or specifies the interchange version the user wishes to appear in bundled EDI documents.

Syntax `const char* OutVersion() const;`

`void OutVersion (const char* );`

Returns The first form of the method returns a pointer to a character string that contains the interchange version the user wishes to appear in bundled EDI documents.

Discussion Use the first form of the method to determine interchange version the user wishes to appear in bundled EDI documents. Use the second form to specify the interchange version the user wishes to appear in bundled EDI documents.

PartnerId( )

Determines or specifies the partnership ID.

Syntax `EcxPartnerId& PartnerId();`

```
void PartnerId ( const EcxPartnerId& id );
```

Parameters The `PartnerID()` method has the following parameters:

`id`	A reference to an `EcxPartnerId` that specifies the partnership.

Returns The first form of the method returns a reference to an `EcxPartnerId` object that contains the ID.

Discussion Use the first form of the method to determine the partnership ID. Use the second form to specify the partnership ID. The `PartnerID()` method does not modify the database.

See also The `EcxPartnerId` class on page 183.

PreEnveloped( )

Determines or specifies whether documents are preenveloped.

Syntax `unsigned int PreEnveloped() const;`

```
void PreEnveloped ( const unsigned int& type );
```

Parameters The `PreEnveloped()` method has the following parameters:

`type`	An unsigned integer that specifies the envelope type.

Returns The first form of the method returns an unsigned integer that contains the envelope type.

Discussion Use the first form of the method to determine the envelope type. Use the second form to specify the envelope type. The `PreEnveloped()` method does not modify the database.

You can use any of the following values:

Constant	Value
ENVELOPE_UNKNOWN	0
ENVELOPE_REGULAR	1
ENVELOPE_NONE	2
ENVELOPE_EDI	3

See also "Class Variables" on page 149.

PrimaryXportParam( )

Determines or specifies the primary transport protocol parameter.

Syntax

```
const char* PrimaryXportParam() const;
void PrimaryXportParam ( const char* param );
```

Parameters The `PrimaryXportParam()` method has the following parameters:

`param`	A pointer to a character string that contains the protocol parameter.

Returns The first form of the method returns a pointer to a character string that contains the protocol parameter.

Discussion Use the first form of the method to determine the protocol parameter. Use the second form to specify the protocol parameter. The `PrimaryXportParam()` method does not modify the database.

PrimaryXportType( )

Determines or specifies the primary transport protocol.

Syntax

```
const char* PrimaryXportType() const;
void PrimaryXportType ( const char* protocol );
```

Parameters The `PrimaryXportType()` method has the following parameters:

`protocol`	A pointer to a character string that contains the protocol.

Returns The first form of the method returns a pointer to a character string that contains the protocol.

Discussion Use the first form of the method to determine the protocol. Use the second form to specify the protocol. The `PrimaryXportType()` method does not modify the database.

RcvrAppCode( )

Determines or specifies the application receiver code.

Syntax
```
const char* RcvrAppCode() const;
void RcvrAppCode ( const char* );
```

Returns The first form of the method returns a pointer to a character string that contains the application receiver code.

Discussion Use the first form of the method to determine the application receiver code. Use the second form to specify the application receiver code.

RcvrAppQual( )

Determines or specifies the receiving member main trading address.

Syntax
```
const char* RcvrAppQual() const;
void RcvrAppQual ( const char* );
```

Returns The first form of the method returns a pointer to a character string that contains the receiving member main trading address.

Discussion Use the first form of the method to determine the receiving member main trading address. Use the second form to specify the receiving member main trading address.

ReceiverCertificateType( )

Determines or specifies the receiver's certificate type

Syntax
```
unsigned int ReceiverCertificateType() const;
void ReceiverCertificateType ( const unsigned int& type );
```

Parameters The `ReceiverCertificateType()` method has the following parameters:

`type`	An unsigned integer that specifies the certificate type.

Returns The first form of the method returns an unsigned integer that contains the certificate type.

Discussion Use the first form of the method to determine the certificate type. Use the second form to specify the certificate type. The `ReceiverCertificateType()` method does not modify the database.

You can use any of the following values:

Constant	Value
`CERTTYPE_UNKNOWN`	0
`CERTTYPE_SELF`	1
`CERTTYPE_VERISIGN1`	2
`CERTTYPE_VERISIGN2`	3
`CERTTYPE_VERISIGN3`	4

See also "Class Variables" on page 149.

ReceiverName( )

Determines or specifies the receiver's member name.

Syntax

```
const char* ReceiverName() const;
void ReceiverName ( const char* name );
```

Parameters The `ReceiverName()` method has the following parameters:

`name`	A pointer to a character string that contains the member name.

Returns The first form of the method returns a pointer to a character string that contains the member name.

Discussion Use the first form of the method to determine the member name. Use the second form to specify the member name. The `ReceiverName()` method does not modify the database.

Example See "Adding Partnerships" on page 145. See "Listing Partnerships" on page 146.

ReceiverQual( )

Determines or specifies the receiver's trading address qualifier.

Syntax `const char* ReceiverQual() const;`

`void ReceiverQual ( const char* qualifier );`

Parameters The `ReceiverQual()` method has the following parameters:

`qualifier`	A pointer to a character string that contains the qualifier.

Returns The first form of the method returns a pointer to a character string that contains the qualifier.

Discussion Use the first form of the method to determine the qualifier. Use the second form to specify the qualifier. The `ReceiverQual()` method does not modify the database.

Example See "Adding Partnerships" on page 145.

ReceiverQualId( )

Determines or specifies the receiver's trading address

Syntax `const char* ReceiverQualId() const;`

`void ReceiverQualId ( const char* id );`

Parameters The `ReceiverQualId()` method has the following parameters:

`id`	A pointer to a character string that contains the trading address.

Returns The first form of the method returns a pointer to a character string that contains the trading address.

Discussion Use the first form of the method to determine the trading address. Use the second form to specify the trading address. The `ReceiverQualId()` method does not modify the database.

Example See "Adding Partnerships" on page 145.

ReleaseCharacter( )

Determines or specifies the release character.

Syntax `const char* ReleaseCharacter() const;`

`void ReleaseCharacter ( const char* relChar );`

Parameters The `ReleaseCharacter()` method has the following parameters:

`relChar`	A pointer to a character string that contains the release character.

Returns The first form of the method returns a pointer to a character string that contains the release character.

Discussion Use the first form of the method to determine the release character. Use the second form to specify the release character. The `ReleaseCharacter()` method does not modify the database.

SecondaryXportParam( )

Determines or specifies the secondary transport protocol parameter.

Syntax `const char* SecondaryXportParam() const;`

`void SecondaryXportParam ( const char* param );`

Parameters The `SecondaryXportParam()` method has the following parameters:

`param`	A pointer to a character string that contains the protocol parameter.

Returns The first form of the method returns a pointer to a character string that contains the protocol parameter.

Discussion Use the first form of the method to determine the protocol parameter. Use the second form to specify the protocol parameter. The `SecondaryXportParam()` method does not modify the database.

SecondaryXportType( )

Determines or specifies the secondary transport protocol.

Syntax `const char* SecondaryXportParam() const;`

```
void SecondaryXportType ( const char* protocol );
```

Parameters The `SecondaryXportType()` method has the following parameters:

`protocol`	A pointer to a character string that contains the protocol.

Returns The first form of the method returns a pointer to a character string that contains the protocol.

Discussion Use the first form of the method to determine the protocol. Use the second form to specify the protocol. The `SecondaryXportType()` method does not modify the database.

Security( )

Determines or specifies the kind of security.

Syntax `unsigned int Security() const;`

```
void Security ( const unsigned int& security );
```

Parameters The `Security()` method has the following parameters:

`security`	An unsigned integer that specifies the security.

Returns The first form of the method returns an unsigned integer that contains the certificate type.

Discussion Use the first form of the method to determine the security. Use the second form to specify the security. The `Security()` method does not modify the database.

You can use any of the following values:

Constant	Value
`SECURITY_PLAIN`	0
`CERTTYPE_SELF`	1
`SECURITY_ENCRYPTED`	2
`SECURITY_SIGNEDANDENCRYPTED`	3

See also "Class Variables" on page 149.

SegmentTerminator( )

Determines or specifies the segment terminator character.

Syntax `const char* SegmentTerminator() const;`

```
void SegmentTerminator ( const char* terminator );
```

Parameters The `SegmentTerminator()` method has the following parameters:

`terminator`	A pointer to a character string that contains the terminator character.

Returns The first form of the method returns a pointer to a character string that contains the terminator character.

Discussion Use the first form of the method to determine the terminator character. Use the second form to specify the terminator character. The `SegmentTerminator()` method does not modify the database.

SndrAppCode( )

Determines or specifies the application sender code.

Syntax `const char* SndrAppCode() const;`

```
void SndrAppCode ( const char* );
```

Returns The first form of the method returns a pointer to a character string that contains the application sender code.

Discussion Use the first form of the method to determine the application sender code. Use the second form to specify the application sender code.

SndrAppQual( )

Determines or specifies the sending member main trading address.

Syntax `const char* SndrAppQual() const;`

```
void SndrAppQual ( const char* );
```

Returns The first form of the method returns a pointer to a character string that contains the sending member main trading address.

Discussion Use the first form of the method to determine the sending member main trading address. Use the second form to specify the sending member main trading address.

SenderCertificateType( )

Determines or specifies the sender's certificate type.

Syntax `unsigned int SenderCertificateType() const;`

```
void SenderCertificateType ( const unsigned int& type );
```

Parameters The `SenderCertificateType()` method has the following parameters:

`type`	An unsigned integer that specifies the certificate type.

Returns The first form of the method returns an unsigned integer that contains the certificate type.

Discussion Use the first form of the method to determine the certificate type. Use the second form to specify the certificate type. The `SenderCertificateType()` method does not modify the database.

You can use any of the following values:

Constant	Value
`CERTTYPE_UNKNOWN`	0
`CERTTYPE_SELF`	1
`CERTTYPE_VERISIGN1`	2
`CERTTYPE_VERISIGN2`	3
`CERTTYPE_VERISIGN3`	4

See also "Class Variables" on page 149.

SenderName( )

Determines or specifies the sender's member name.

Syntax `const char* SenderName() const;`

```
void SenderName ( const char* name );
```

Parameters The `SenderName()` method has the following parameters:

`name`	A pointer to a character string that contains the member name.

Returns The first form of the method returns a pointer to a character string that contains the member name.

Discussion Use the first form of the method to determine the member name. Use the second form to specify the member name. The `SenderName()` method does not modify the database.

Example See "Adding Partnerships" on page 145. See "Listing Partnerships" on page 146.

SenderQual( )

Determines or specifies the sender's trading address qualifier.

Syntax

```
const char* SenderQual() const;
void SenderQual ( const char* qualifier );
```

Parameters The `SenderQual()` method has the following parameters:

`qualifier`	A pointer to a character string that contains the qualifier.

Returns The first form of the method returns a pointer to a character string that contains the qualifier.

Discussion Use the first form of the method to determine the qualifier. Use the second form to specify the qualifier. The `SenderQual()` method does not modify the database.

Example See "Adding Partnerships" on page 145.

SenderQualId( )

Determines or specifies the sender's trading address.

Syntax `const char* SenderQualId() const;`

`void SenderQualId ( const char* id );`

Parameters The `SenderQualId()` method has the following parameters:

`id`	A pointer to a character string that contains the trading address.

Returns The first form of the method returns a pointer to a character string that contains the trading address.

Discussion Use the first form of the method to determine the trading address. Use the second form to specify the trading address. The `SenderQualId()` method does not modify the database.

Example See "Adding Partnerships" on page 145.

SendType( )

Determines or specifies when the document is to be sent.

Syntax `unsigned int SendType() const;`

`void SendType ( const unsigned int& type );`

Parameters The `SendType()` method has the following parameters:

`type`	An unsigned integer that specifies the send type.

Returns The first form of the method returns an unsigned integer that contains the send type.

Discussion Use the first form of the method to determine the send type. Use the second form to specify the send type. The `SendType()` method does not modify the database.

You can use any of the following values:

Constant	Value
`SENDTYPE_UNKNOWN`	0

Constant	Value
SENDTYPE_IMMEDIATE	1
SENDTYPE_ONETIME	2
SENDTYPE_PERIODIC	3

See also "Class Variables" on page 149.

SetLogin( )

Allows the object to access the database.

Syntax `EcxPartnership& SetLogin(EcxLogin& login);`

Parameters The `SetLogin()` method has the following parameters:

`login`	A reference to a valid `EcxLogin` object

Returns A reference to this partnership object.

Discussion If you do not use the form of the constructor that accepts a login object, you must call the `SetLogin()` method before accessing this object.

Example See "Creating Partnership Objects" on page 144.

See also The `EcxPartnership` constructor on page 151. The `EcxLogin` class on page 127.

StandardName( )

Determines or specifies the name of the EDI standard.

Syntax

```
const char* StandardName() const;
void StandardName ( const char* name );
```

Parameters The `StandardName()` method has the following parameters:

`name`	A pointer to a character string that contains the standard name.

Returns The first form of the method returns a pointer to a character string that contains the standard name.

Discussion Use the first form of the method to determine the standard name. Use the second form to specify the standard name. The `StandardName()` method does not modify the database.

Example See "Adding Partnerships" on page 145. See "Listing Partnerships" on page 146.

StandardRelease( )

Determines or specifies the standard's release number.

Syntax

```
const char* StandardRelease() const;
void StandardRelease ( const char* release );
```

Parameters The `StandardRelease()` method has the following parameters:

`release`	A pointer to a character string that contains the release number.

Returns The first form of the method returns a pointer to a character string that contains the release number.

Discussion Use the first form of the method to determine the release number. Use the second form to specify the release number. The `StandardRelease()` method does not modify the database.

Example See "Adding Partnerships" on page 145.

StandardVersion( )

Determines or specifies the standard's version number.

Syntax

```
const char* StandardVersion() const;
void StandardVersion ( const char* version );
```

Parameters The `StandardVersion()` method has the following parameters:

`version`	A pointer to a character string that contains the version number.

Returns The first form of the method returns a pointer to a character string that contains the version number.

Discussion Use the first form of the method to determine the version number. Use the second form to specify the version number. The `StandardRelease()` method does not modify the database.

Example See "Adding Partnerships" on page 145. See "Listing Partnerships" on page 146.

SubElementSeparator( )

Determines or specifies the data subelement terminator character.

Syntax
```
const char* SubElementSeparator() const;
void SubElementSeparator ( const char* separator );
```

Parameters The `SubElementSeparator()` method has the following parameters:

`separator`	A pointer to a character string that contains the terminator character.

Returns The first form of the method returns a pointer to a character string that contains the terminator character.

Discussion Use the first form of the method to determine the terminator character. Use the second form to specify the terminator character. The `SubElementSeparator()` method does not modify the database.

TestProductionFlag( )

Determines or specifies whether the partnership is used for testing or production.

Syntax
```
unsigned int TestProductionFlag() const;
void TestProductionFlag ( const unsigned int& flag );
```

Parameters The `TestProductionFlag()` method has the following parameters:

`flag`	An unsigned integer that specifies the flag value.

Returns The first form of the method returns an unsigned integer that contains the flag value.

Discussion Use the first form of the method to determine the flag value. Use the second form to specify the flag value. The `TestProductionFlag()` method does not modify the database.

You can set or receive any of the following values:

Description	Value
`Unknown`	0
`Production`	1
`Test`	2

See also "Class Variables" on page 149.

MapDirection( )

Determines or specifies the document translation type.

Syntax `unsigned int MapDirection() const;`

```
void MapDirection ( const unsigned int& type );
```

Parameters The `MapDirection()` method has the following parameters:

`type`	An unsigned integer that specifies the translation type.

Returns The first form of the method returns an unsigned integer that contains the translation type.

Discussion Use the first form of the method to determine the translation type. Use the second form to specify the translation type. The `MapDirection()` method does not modify the database.

You can use any of the following values:

Constant	Value
XLATTYPE_UNKNOWN	0
XLATTYPE_INBOUND	1
XLATTYPE_OUTBOUND	2
XLATTYPE_EDI2EDI	3
XLATTYPE_APP2APP	4
XLATTYPE_NONE	5

See also "Class Variables" on page 149.

About the EcxPartnerID Class

The `EcxPartnerID` class represents a key from which partnership views can be retrieved from the database. You must create an `EcxPartnerID` object before you can call the partnership's `Get()` and PartnerID() methods. A partner ID key consists of the following values:

- partnership ID
- standard ID
- document type

In general, values for a partnership ID and a standard ID are the same for each record in the view.

Methods Summary list:

Constructor and destructor	
`EcxPartnerID()`	Creates an `EcxPartnerID` object.
`~EcxPartnerID()`	Destroys an `EcxPartnerID` object.
Setting key values	
`SetValues()`	Sets the values associated with a partnership view key.

Determining key values	
`DocType()`	Determines the document type in the key.
`PartnershipID()`	Determines the partnership ID in the key.
`StandardID()`	Determines the standard ID in the key.

EcxPartnerID Class Reference

Interface `ecxpartnership.h`

Superclasses None

Subclasses None

Friend Classes None

Syntax `class DLL_ecxsdk EcxPartnerId { ... };`

Constructor and Destructor

EcxPartnerId( )

Creates an `EcxPartnerId` object.

Syntax `EcxPartnerId(void);`

~EcxPartnerId( )

Destroys an `EcxPartnerId` object.

Syntax `virtual ~EcxPartnerId(void);`

Methods

This section describes the methods of the `EcxPartnerId` class.

DocType( )

Determines the document type in the key.

Syntax `const char* DocType(void) const;`

Returns A pointer to a character string that contains the document type.

PartnershipId( )

Determines the partnership ID in the key.

Syntax `long PartnershipId(void) const;`

Returns A long integer that contains the partnership ID.

SetValues( )

Sets the values associated with a partnership view key.

Syntax
```
void SetValues( long partnership_id,
                long standard_id,
                const char* doctype);
```

Parameters The `SetValues()` method has the following parameters:

`partnership_id`	A long integer that specifies the partnership ID.
`standard_id`	A long integer that specifies the standard ID.
`doctype`	A pointer to a character string that specifies the document type.

Example
```
EcxPartnerId ecxpartner;
...
ecxpartner.SetValues(m_partnership_id, m_partnership_id, m_doctype);
m_pPartnership->PartnerId(ecxpartner);
```

StandardId( )

Determines the standard ID in the key.

Syntax `long StandardId(void) const;`

Returns A long integer that contains the standard ID.

Chapter 12

The EcxMember Class

This chapter describes the `EcxMember` class, which represents member records in an iPlanet ECXpert database. This chapter contains the following sections:

About the EcxMember Class

The `EcxMember` class represents member records in an iPlanet ECXpert database. Administrators can manipulate any member record for their trading partnerships; non-administrators can only change contact information in their own record. A user must be logged in to the database before accessing a record.

Methods Summary list:

Constructor and destructor	
`EcxMember()`	Creates an `EcxMember` object.
`~EcxMember()`	Destroys an `EcxMember` object.
Allowing database access	
`SetLogin()`	Allows the object to access the database.

Adding, retrieving, changing and deleting member records	
`Add()`	Adds a member record to the database.
`Get()`	Retrieves a member record from the database.
`Change()`	Changes a member record in the database.
`Delete()`	Deletes a member from the database.
Listing member records	
`List()`	Retrieves a list of member records from the database.
`More()`	Determines whether more records are left in the list.
`Next()`	Associates the object with the next record in the list.
Resetting an object's state	
`Clear()`	Clears the state associated with an object, including its list.
Accessing key fields	
`Name()`	Determines or specifies the name of the member.
Accessing contact information	
`ContactName()`	Determines or specifies the name of the contact person for this member.
`ContactCompany()`	Determines or specifies the contact's company.
`ContactAddress1()`	Determines or specifies the first line of the contact's address.
`ContactAddress2()`	Determines or specifies the second line of the contact's address.
`ContactCity()`	Determines or specifies the contact's city.
`ContactState()`	Determines or specifies the contact's state.
`ContactZip()`	Determines or specifies the contact's zip or postal code.
`ContactCountry()`	Determines or specifies the contact's country.
`ContactPhone()`	Determines or specifies the contact's phone number.
`ContactFax()`	Determines or specifies the contact's fax number.
`ContactEmailId()`	Determines or specifies the contact's e-mail address.
Accessing other fields	
`Description()`	Determines or specifies the member's description.
`Type()`	Determines or specifies the type of member.
`ParentName()`	Determines the name of the parent member.

`IsGroup()`	Determines or specifies whether the member is a group or individual.
`Active()`	Determines or specifies whether the member is active.
`Password()`	Determines or specifies the member's password.
`Trusted()`	Determines or specifies whether the member is trusted.
`ObjPerm()`	Determines or specifies the record's access permissions.
`ModByGroup()`	Determines the group that last modified the record.
`ModByUser()`	Determines the user that last modified the record.
`ModDt()`	Determines the date the record was last modified.

Using the EcxMember Class

The following sections show how to

- create member objects
- add members to the database
- change members' records in the database
- list members in the database
- delete members from the database

Creating Member Objects

The following example shows how to create an `EcxMember` object and how to allow access to the database by calling the object's `SetLogin()` method:

```
EcxMember * make_memberobj(EcxLogin * pLogin) {

  EcxMember * pMember = NULL;

  if((pMember = new EcxMember())->Errnum()) {
    cout << "EcxMember Object Error:" << endl;
    cout << "\tErrnum: " << pMember->Errnum() << endl;
    cout << "\tErrmsg: " << pMember->Errmsg() << endl;
    cout << endl;
    return(NULL);
  }
```

```
  if((pMember->SetLogin(*pLogin)).Errnum()) {
    cout << "EcxMember.SetLogin() Failed:" << endl;
    cout << "\tErrnum: " << pMember->Errnum() << endl;
    cout << "\tErrmsg: " << pMember->Errmsg() << endl;
    cout << endl;
    delete pMember;
    return(NULL);
  }

  return(pMember);
}
```

Alternatively, you can pass the login object to the `EcxMember` constructor without having to call `SetLogin()`.

Adding Members

The following example shows how to add a member record to the database. An administrator's login must be associated with the object you want to add.

```
int add_member(EcxMember *pMember, const char *name) {

  pMember->Clear();

  pMember->Name(name);
  pMember->Description("This is the description");
  pMember->Type(pMember->MEMBER);
  pMember->IsGroup(FALSE);
  pMember->Active(TRUE);
  pMember->Password(name);
  pMember->Trusted(FALSE);
  pMember->ContactName("Jack Flack");
  pMember->ContactCompany("Company AAA");
  pMember->ContactAddress1("109 Short Stack St.");
  pMember->ContactAddress2("Apt. #12");
  pMember->ContactCity("Big City");
  pMember->ContactState("New California");
  pMember->ContactZip("12666");
  pMember->ContactCountry("AUFD");
  pMember->ContactPhone("123 456-7890");
  pMember->ContactFax("123 456-7899");
  pMember->ContactEmailId("crank@flipant.org");
  pMember->ObjPerm(755);

  if((pMember->Add()).Errnum()) {
    cout << "EcxMember.add() Failed for user: " << name << endl;
```

```
    cout << "\tErrnum: " << pMember->Errnum() << endl;
    cout << "\tErrmsg: " << pMember->Errmsg() << endl;
    return(pMember->Errnum());
  }

  cout << "*** Added member: " << name << endl;

  return(0);
}
```

Changing Members' Fields

The following example shows how to change the contact's e-mail address. The `Get()` method retrieves the record to modify using the key field, which is specified by calling the object's `Name()` method.

NOTE Non-administrators can only retrieve their own record and, thus, change only their own record.

```
int change_email(EcxMember * pMember, const char * name) {

  char email[1024];

  pMember->Clear();
  pMember->Name(name);

  if((pMember->Get()).Errnum()) {
    cout << "EcxMember.Get() Failed for user: " << name << endl;
    cout << "\tErrnum: " << pMember->Errnum() << endl;
    cout << "\tErrmsg: " << pMember->Errmsg() << endl;
    return(pMember->Errnum());
  }

  strcpy(email, name);
  strcat(email, "@heaven.org");

  pMember->ContactEmailId(email);

  if((pMember->Change()).Errnum()) {
    cout << "EcxMember.Change() Failed for user: " << name <<
endl;
    cout << "\tErrnum: " << pMember->Errnum() << endl;
    cout << "\tErrmsg: " << pMember->Errmsg() << endl;
    return(pMember->Errnum());
  }
```

```
  return(0);
}
```

Listing Members

The following example shows how to create a list of all members.

NOTE If the login object specifies a non-administrator, this example returns only that member's record.

```
int list(EcxMember *pMember) {

  pMember->Clear();

  if((pMember->List()).Errnum()) {
    cout << "EcxMember.List() Failed:" << endl;
    cout << "\tErrnum: " << pMember->Errnum() << endl;
    cout << "\tErrmsg: " << pMember->Errmsg() << endl;
    return(pMember->Errnum());
  }

  cout << "*** Listing members" << pMember->More();
  cout << " records found. ***" << endl;

  while(pMember->More()) {
    cout << pMember->Name()            << ":";
    cout << pMember->Type()            << ":";
    cout << pMember->ContactName()     << ":";
    cout << pMember->ContactAddress1() << ":";
    cout << pMember->ContactAddress2() << ":";
    cout << pMember->ContactEmailId()  << endl;
    pMember->Next();
  }

  return(0);
}
```

Deleting Members

The following example shows how to delete a member record from the database. An administrator's login must be associated with the object you want to delete.

```
int delete_member(EcxMember *pMember, const char * name) {

  pMember->Clear();
  pMember->Name(name);

  if((pMember->Delete()).Errnum()) {
    cout << "EcxMember.Delete() Failed for user: " << name <<
endl;
    cout << "\tErrnum: " << pMember->Errnum() << endl;
    cout << "\tErrmsg: " << pMember->Errmsg() << endl;
    return(pMember->Errnum());
  }

  cout << "*** Deleted member: " << name << endl;

  return(0);
}
```

EcxMember Class Reference

Interface `ecxmember.h`

Superclasses `EcxBase`

Subclasses None

Friend Classes None

Syntax `class EcxMember : public EcxBase { ... };`

Class Variables

The following class variables allow you to identify the member as either an administrator or an ordinary member:

Syntax `static int ADMINISTRATOR;`

```
static int MEMBER;
```

ADMINISTRATOR	Administrator
MEMBER	Member (non-administrator)

Constructor and Destructor

EcxMember( )

Creates an EcxMember object.

Syntax `EcxMember(void);`

`EcxMember(EcxLogin& login);`

Parameters The constructor has the following parameters:

login	The login object to associate with this member object.

Discussion The first form of the constructor allows you to create a stack-based object. The second form of the constructor requires that you create an EcxLogin object before you create this object.

Example See "Creating Member Objects" on page 189.

See also The SetLogin() method on page 207. The EcxLogin class on page 127.

~EcxMember( )

Destroys an EcxMember object.

Syntax `virtual ~EcxMember(void);`

Discussion The destructor is called when you delete the object. You can reuse an object instead of deleting it by calling the object's Clear() method. The destructor does not destroy the associated EcxLogin object.

See also The `Clear()` method on page 196.

Methods

This section describes the methods of the `EcxMember` class.

Active( )

Determines or specifies whether the member is active.

Syntax `unsigned int Active() const;`

void Active(const unsigned int status);

Parameters The `Active()` method has the following parameters:

`status`	An unsigned integer that specifies whether the member is active.

Returns The first form of the method returns an unsigned integer that contains the status.

Discussion Use the first form of the method to determine whether the member is active. Use the second form to specify whether the member is active. A status of TRUE (1) indicates that the member is active. A status of FALSE (0) indicates that the member is inactive. The `Active()` method does not modify the database.

Example See "Adding Members" on page 190.

Add( )

Adds a member record to the database.

Syntax `EcxMember& Add(void);`

Returns A reference to this member object.

Discussion You must be an administrator and be logged in before calling this method. You must specify the member's name in the object, by calling the `Name()` method, before calling the `Add()` method.

The parent name and the group-modified-by fields are set to the parent name of the logged-in user; by default, this is 'rootgroup'. The user-modified-by field is set to the name of the logged-in user. Any other fields not specified in the object will become 0 or NULL in the database.

Example See "Adding Members" on page 190.

See also The `Name()` method on page 205.

Change( )

Changes a member record in the database.

Syntax `EcxMember& Change(void);`

Returns A reference to this member object.

Discussion This method updates the last record retrieved by calling the object's `Get()`, `List()`, or `Next()` method. Administrators may change any field for which a mutator method is provided. Non-administrators can only change the contact information in their own record. Specifically, a non-administrator cannot change the contents of the trusted, active, parent name, or isGroup fields.

CAUTION If you do not call the object's `Get()`, `List()`, or `Next()` method first, the object's name field, which is set by calling the `Name()` method, specifies the record that is changed. In this case, the record is completely overwritten using the object's fields. Any fields not set in the object will be replaced by 0 or NULL in the database.

Example See "Changing Members' Fields" on page 191.

See also The `Get()` method on page 203. The `List()` method on page 204. The `Next()` method on page 205. The `Name()` method on page 205.

Clear( )

Clears the state associated with an object, including its list.

Syntax `void Clear(void);`

Discussion The parent name is set to 'rootgroup'. Other fields of the object are reset to 0 or NULL. A list contains no records.

Example See "Listing Members" on page 192.

ContactAddress1( )

Determines or specifies the first line of the contact's address.

Syntax
```
const char* ContactAddress1() const;
void ContactAddress1(const char* addr1);
```

Parameters The `ContactAddress1()` method has the following parameters:

`addr1`	A pointer to a character string that contains the address line.

Returns The first form of the method returns a pointer to a character string that contains the address line.

Discussion Use the first form of the method to determine the first line of the address. Use the second form to specify the address line. The `ContactAddress1()` method does not modify the database.

Example See "Adding Members" on page 190.

ContactAddress2( )

Determines or specifies the second line of the contact's address.

Syntax

```
const char* ContactAddress2() const;
void ContactAddress2(const char* addr2);
```

Parameters The `ContactAddress2()` method has the following parameters:

`addr2`	A pointer to a character string that contains the address line.

Returns The first form of the method returns a pointer to a character string that contains the address line.

Discussion Use the first form of the method to determine the second line of the address. Use the second form to specify the address line. The `ContactAddress2()` method does not modify the database.

Example See "Adding Members" on page 190.

ContactCity( )

Determines or specifies the second line of the contact's city.

Syntax

```
const char* ContactCity() const;
void ContactCity(const char* city);
```

Parameters The `ContactCity()` method has the following parameters:

`city`	A pointer to a character string that contains the city.

Returns The first form of the method returns a pointer to a character string that contains the city.

Discussion Use the first form of the method to determine the city. Use the second form to specify the city. The `ContactCity()` method does not modify the database.

Example See "Adding Members" on page 190.

ContactCompany( )

Determines or specifies the contact's company.

Syntax

```
const char* ContactCompany() const;
void ContactCompany(const char * company);
```

Parameters The `ContactCompany()` method has the following parameters:

`company`	A pointer to a character string that contains the company name.

Returns The first form of the method returns a pointer to a character string that contains the company name.

Discussion Use the first form of the method to determine the company name. Use the second form to specify the company name. The `ContactCompany()` method does not modify the database.

Example See "Adding Members" on page 190.

ContactCountry( )

Determines or specifies the contact's country.

Syntax

```
const char* ContactCountry() const;
void ContactCountry(const char* country);
```

Parameters The `ContactCountry()` method has the following parameters:

`country`	A pointer to a character string that contains the country name.

Returns The first form of the method returns a pointer to a character string that contains the country name.

Discussion Use the first form of the method to determine the country. Use the second form to specify the country. The `ContactCountry()` method does not modify the database.

Example See "Adding Members" on page 190.

ContactEmailId( )

Determines or specifies the contact's e-mail address.

Syntax

```
const char* ContactEmailId() const;
void ContactEmailId(const char* emailID);
```

Parameters The `ContactEmailId()` method has the following parameters:

`emailID`	A pointer to a character string that contains the e-mail address.

Returns The first form of the method returns a pointer to a character string that contains the e-mail address.

Discussion Use the first form of the method to determine the e-mail address. Use the second form to specify the e-mail address. The `ContactEmailId()` method does not modify the database.

Example See "Adding Members" on page 190.

ContactFax( )

Determines or specifies the contact's fax number.

Syntax

```
const char* ContactFax() const;
void ContactFax(const char* fax);
```

Parameters The `ContactFax()` method has the following parameters:

`fax`	A pointer to a character string that contains the fax number.

Returns The first form of the method returns a pointer to a character string that contains the fax number.

Discussion Use the first form of the method to determine the fax number. Use the second form to specify the fax number. The `ContactFax()` method does not modify the database.

Example See "Adding Members" on page 190.

ContactName( )

Determines or specifies the name of the contact person for this member.

Syntax

```
const char* ContactName() const;
void ContactName(const char* name);
```

Parameters The `ContactName()` method has the following parameters:

`name`	A pointer to a character string that contains the contact's name.

Returns The first form of the method returns a pointer to a character string that contains the name.

Discussion Use the first form of the method to determine the contact's name. Use the second form to specify the name. The `ContactName()` method does not modify the database.

Example See "Adding Members" on page 190.

ContactPhone( )

Determines or specifies the contact's phone number.

Syntax

```
const char* ContactPhone() const;
void ContactPhone(const char* phone);
```

Parameters The `ContactPhone()` method has the following parameters:

`phone`	A pointer to a character string that contains the phone number.

Returns The first form of the method returns a pointer to a character string that contains the phone number.

Discussion Use the first form of the method to determine the phone number. Use the second form to specify the phone number. The `ContactPhone()` method does not modify the database.

Example See "Adding Members" on page 190.

ContactState( )

Determines or specifies the contact's state.

Syntax
```
const char* ContactState() const;
void ContactState(const char* state);
```

Parameters The `ContactState()` method has the following parameters:

`state`	A pointer to a character string that contains the state.

Returns The first form of the method returns a pointer to a character string that contains the state.

Discussion Use the first form of the method to determine the state. Use the second form to specify the state. The `ContactState()` method does not modify the database.

Example See "Adding Members" on page 190.

ContactZip( )

Determines or specifies the contact's zip or postal code.

Syntax
```
const char* ContactZip() const;
void ContactZip(const char* zip);
```

Parameters The `ContactZip()` method has the following parameters:

`zip`	A pointer to a character string that contains the zip or postal code.

Returns The first form of the method returns a pointer to a character string that contains the zip or postal code.

Discussion Use the first form of the method to determine the zip or postal code. Use the second form to specify the zip or postal code. The `ContactZip()` method does not modify the database.

Example See "Adding Members" on page 190.

Delete( )

Deletes a member from the database.

Syntax `EcxMember& Delete(void);`

Returns A reference to this member object.

Discussion You must be an administrator and be logged in before calling this method. You must specify the member's name in the object by calling the `Name()` method before you call the `Delete()` method. After this method executes, the object is reset; the parent name is set to 'rootgroup' and other fields of the object are reset to 0 or NULL. A list contains no records.

CAUTION In addition to deleting the membership record, the `Delete()` method also deletes the partnerships and services associated with the member.

Example See "Deleting Members" on page 193.

See also The `Name()` method on page 205.

Description( )

Determines or specifies the member's description.

Syntax

```
const char* Description() const;
void Description(const char* desc);
```

Parameters The `Description()` method has the following parameters:

`desc`	A pointer to a character string that contains the description.

Returns The first form of the method returns a pointer to a character string that contains the description.

Discussion Use the first form of the method to determine the description. Use the second form to specify the description. The `Description()` method does not modify the database.

Example See "Adding Members" on page 190.

Get( )

Retrieves a member record from the database.

Syntax `EcxMember& Get(void);`

Returns A reference to this member object.

Discussion Administrators may retrieve any membership record. Non-administrators can only retrieve their own record. You must specify the member's name in the object by calling the `Name()` method before you call the `Get()` method.

Example See "Changing Members' Fields" on page 191.

See also The `Name()` method on page 205.

IsGroup( )

Determines or specifies whether the member is a group or individual.

Syntax

```
unsigned int IsGroup() const;
void IsGroup(const unsigned int status);
```

Parameters The `IsGroup()` method has the following parameters:

`status`	An unsigned integer that specifies whether the member is a group.

Returns The first form of the method returns an unsigned integer that contains the status.

Discussion Use the first form of the method to determine whether the member is a group. Use the second form to specify whether the member is a group. A status of TRUE (1) indicates that the member is a group. A status of FALSE (0) indicates that the member is an individual. The `IsGroup()` method does not modify the database.

Example See "Adding Members" on page 190.

List( )

Retrieves a list of member records from the database.

Syntax `EcxMember& List(void);`

Returns A reference to this member object.

Discussion If you specify the member's name in the object by calling the `Name()` method first, only the record matching with the specified name will be retrieved. After calling the `List()` method, the member object contains fields from the first record from the list.

Example See "Listing Members" on page 192.

See also The `Name()` method on page 205.

ModByGroup( )

Determines the group that last modified the record.

Syntax `const char* ModByGroup() const;`

Returns A pointer to a character string that contains the group.

ModByUser( )

Determines the user that last modified the record.

Syntax `const char* ModByUser() const;`

Returns A pointer to a character string that contains the user name.

ModDt( )

Determines the date the record was last modified.

Syntax `const char* ModDt() const;`

Returns A pointer to a character string that contains the date.

More( )

Determines whether more records are left in the list.

Syntax `long More(void);`

Returns A long integer that contains the number of records not yet accessed from the list.

Discussion After calling the `List()` method and before calling the `Next()` method, the `More()` method returns the total number of records in the list. All records have been accessed when the `More()` method returns 0.

Example See "Listing Members" on page 192.

See also The `List()` method on page 204. The `Next()` method on page 205.

Name( )

Determines or specifies the name of the member.

Syntax

```
const char* Name() const;
void Name(const char* name);
```

Parameters The `Name()` method has the following parameters:

`name`	A pointer to a character string that contains the member's name.

Returns The first form of the method returns a pointer to a character string that contains the name.

Discussion Use the first form of the method to determine the member's name. Use the second form to specify the name. The `Name()` method does not modify the database.

Example See "Adding Members" on page 190.

Next( )

Associates the object with the next record in the list.

Syntax `EcxMember& Next(void);`

Returns A reference to this member object.

Discussion The `Next()` method sets the fields in the object to match those in the next record in the list. The `Next()` method decrements the number of records not yet accessed, which is returned by the `More()` method.

CAUTION	Do not call the `Next()` method if the `More()` method returns a value less than 1; the results are unpredictable.

Example See "Listing Members" on page 192.

See also The `More()` method on page 205.

ObjPerm( )

Determines or specifies the record's access permissions.

Syntax

```
unsigned int ObjPerm() const;
void ObjPerm(const unsigned int permissions);
```

Parameters The `ObjPerm()` method has the following parameters:

`permissions`	An unsigned integer that specifies the access permissions.

Returns The first form of the method returns an unsigned integer that contains the permissions.

Discussion Use the first form of the method to determine the record's access permissions. Use the second form to specify the permissions. The `ObjPerm()` method does not modify the database.

Example See "Adding Members" on page 190.

ParentName( )

Determines the name of the parent member.

Syntax

```
const char* ParentName() const;
```

Returns A pointer to a character string that contains the name.

Password( )

Determines or specifies the member's password.

Syntax `const char* Password() const;`

`void Password(const char* passwd);`

Parameters The `Password()` method has the following parameters:

`passwd`	A pointer to a character string that contains the password.

Returns The first form of the method returns a pointer to a character string that contains the password.

Discussion Use the first form of the method to determine the member's password. Use the second form to specify the password. The `Password()` method does not modify the database.

Example See "Adding Members" on page 190.

SetLogin( )

Allows the object to access the database.

Syntax `EcxMember& SetLogin(EcxLogin& login);`

Parameters The `SetLogin()` method has the following parameters:

`login`	A reference to a valid `EcxLogin` object

Returns A reference to this member object.

Discussion If you do not use the form of the constructor that accepts a login object, you must call the `SetLogin()` method before using this object.

Example See "Creating Member Objects" on page 189.

See also The `EcxMember` constructor on page 194. The `EcxLogin` class on page 127.

Trusted( )

Determines or specifies whether the member is trusted.

Syntax `unsigned int Trusted() const;`

```
void Trusted(const unsigned int status);
```

Parameters The `Trusted()` method has the following parameters:

`status`	An unsigned integer that specifies whether the member is a trusted member.

Returns The first form of the method returns an unsigned integer that contains the status.

Discussion Use the first form of the method to determine whether the member is a trusted member. Use the second form to specify whether the member is a trusted member. A status of TRUE (1) indicates that the member is a trusted member. A status of FALSE (0) indicates that the member is not a trusted member. The `Trusted()` method does not modify the database.

Example See "Adding Members" on page 190.

Type( )

Determines or specifies the type of member.

Syntax

```
unsigned int Type() const;
void Type(const unsigned int type);
```

Parameters The `Type()` method has the following parameters:

`type`	An unsigned integer that specifies whether the member is an administrator.

Returns The first form of the method returns an unsigned integer that contains the type.

Discussion Use the first form of the method to determine whether the member is an administrator. Use the second form to specify whether the member is an administrator. A type of `ADMINISTRATOR` indicates that the member is an administrator. A type of `MEMBER` indicates that the member is not an administrator. The `Type()` method does not modify the database.

Example See "Adding Members" on page 190.

See also "Class Variables" on page 193.

Chapter 13

Document-Related Classes

This chapter describes the `EcxDocument` class, which represents documents sent to the logged-in user via iPlanet ECXpert. This chapter also describes the `EcxDocId` class, which represents key values for `EcxDocument` objects. This chapter contains the following sections:

About the EcxDocument Class

The `EcxDocument` class represents documents sent to the logged-in user via iPlanet ECXpert. You can retrieve these document records and access information that identifies them, such as the filename that contains the document's content.

Methods Summary list:

Constructor and destructor	
`EcxDocument()`	Creates an `EcxDocument` object.
`~EcxDocument()`	Destroys an `EcxDocument` object.
Allowing database access	
`SetLogin()`	Allows the object to access the database.

Retrieving and listing document records	
`Get()`	Retrieves a document record from the database.
`List()`	Retrieves a list of document records from the database.
`More()`	Determines whether more records are left in the list.
`Next()`	Associates the object with the next record in the list.
`Delete`	Deletes document records from the database.
Resetting an object's state	
`Clear()`	Clears the state associated with an object, including its list.
Accessing key fields	
`DocId()`	Determines the document ID.
Accessing document information	
`FileName()`	Determines the name of the file associated with the document.
`SecondaryTitle()`	Determines the secondary title.
`SecondaryValue()`	Determines the secondary value.
`SenderName()`	Determines or specifies the sender's member name.
`State()`	Determines the document's state.
`Title()`	Determines the document's title.
`Value()`	Determines the document's value.
`XportParam()`	Determines the transport parameter.
`XportType()`	Determines the transport protocol.
`Filename()`	Determines the name of the file associated with the document.
`CreationDate()`	Determines the date the document was created.
`ModifyDate()`	Determines the most recent document modification date.
`DocType()`	Determines the document type.
`Standard()`	Determines the document's EDI standard.
`Version()`	Determines the document's EDI version.
`Release()`	Determines the document's EDI standard release number.
`CardCount()`	Determines the number of cards associated with the document.
`DataState()`	Determines the state the document data is in.

`Read()`	Determines whether the document has been read.
Accessing card-level information	
`CardIOType()`	Determines the card input/output type.
`CardFlags()`	Accesses information about what card flags have been set.
`TrackState()`	Determines the document's tracking state.
`TranslatedFileName()`	Accesses the name of the translated file.
`SetReadyForPurge()`	Sets the document to "ready to be purged" state.

Using the EcxDocument Class

The following example shows how to create an `EcxDocument` object and use it to list the tracking records for incoming documents in the database. Records received by the "ECXSDK" transport type are listed first, followed by those from the specified sender by the "ECXSDK" transport type.

```
#include <stdio.h>
#include <fstream.h>

#include "ecxsdk.h"

int main(int argc, char * argv[]) {
  int  retval = -1;

  EcxInit ecxinit;
  EcxLogin * pLogin;
  EcxDocument * pDocument;
  EcxDocId id;

  if(argc != 3) {
    usage(argv);
    return(retval);
  }

  if((pLogin = new EcxLogin())->Errnum()) {
    cout << "EcxLogin Object Error:" << endl;
    cout << "\tErrnum: " << pLogin->Errnum() << endl;
    cout << "\tErrmsg: " << pLogin->Errmsg() << endl;
    cout << endl;
    return(pLogin->Errnum());
  }

  if((pLogin->Login(argv[1], argv[2])).Errnum()) {
    cout << "EcxLogin.Login() Failed:" << endl;
    cout << "\tErrnum: " << pLogin->Errnum() << endl;
```

```
  cout << "\tErrmsg: " << pLogin->Errmsg() << endl;
  cout << endl;
  return(pLogin->Errnum());
}

cout << "Successful login for user: " << argv[1] << endl;

if((pDocument = new EcxDocument())->Errnum()) {
  cout << "EcxDocument Object Error:" << endl;
  cout << "\tErrnum: " << pDocument->Errnum() << endl;
  cout << "\tErrmsg: " << pDocument->Errmsg() << endl;
  cout << endl;
  return(pDocument->Errnum());
}

if((pDocument->SetLogin(*pLogin)).Errnum()) {
  cout << "EcxDocument.SetLogin() Failed:" << endl;
  cout << "\tErrnum: " << pDocument->Errnum() << endl;
  cout << "\tErrmsg: " << pDocument->Errmsg() << endl;
  cout << endl;
  return(pDocument->Errnum());
}

cout << "Created EcxDocument object!" << endl;

pDocument->XportType("ECXSDK");

if((pDocument->List()).Errnum()) {
  cout << "EcxDocument.List() Failed:" << endl;
  cout << "\tErrnum: " << pDocument->Errnum() << endl;
  cout << "\tErrmsg: " << pDocument->Errmsg() << endl;
  return(pDocument->Errnum());
}

cout << "*** " << pDocument->More() << " records found. ***" << endl;

while(pDocument->More()) {
  cout << "---" << endl;
  cout << "SenderName:     " << pDocument->SenderName() << endl;
  cout << "State:          " << pDocument->State() << endl;
  cout << "Title:          " << pDocument->Title() << endl;
  cout << "Value:          " << pDocument->Value() << endl;
  cout << "SecondaryTitle: " << pDocument->SecondaryTitle() << endl;
  cout << "SecondaryValue: " << pDocument->SecondaryValue() << endl;
  cout << "FileName(1):    " << pDocument->FileName(1) << endl;
  cout << "FileName(2):    " << pDocument->FileName(2) << endl;
  cout << "FileName(3):    " << pDocument->FileName(3) << endl;
  cout << endl;
  id = pDocument->DocId();
  pDocument->Next();
}

pDocument->Clear();
pDocument->SenderName(argv[1]);
pDocument->XportType("ECXSDK");
```

```
  if((pDocument->List()).Errnum()) {
    cout << "EcxDocument.List(" << argv[1] << ") Failed:" << endl;
    cout << "\tErrnum: " << pDocument->Errnum() << endl;
    cout << "\tErrmsg: " << pDocument->Errmsg() << endl;
    return(pDocument->Errnum());
  }

  cout << "*** " << pDocument->More() << " records found. ***" << endl;

  while(pDocument->More()) {
    cout << "---" << endl;
    cout << "SenderName:     " << pDocument->SenderName() << endl;
    cout << "State:          " << pDocument->State() << endl;
    cout << "Title:          " << pDocument->Title() << endl;
    cout << "Value:          " << pDocument->Value() << endl;
    cout << "SecondaryTitle: " << pDocument->SecondaryTitle() << endl;
    cout << "SecondaryValue: " << pDocument->SecondaryValue() << endl;
    cout << "FileName(1):    " << pDocument->FileName(1) << endl;
    cout << "FileName(2):    " << pDocument->FileName(2) << endl;
    cout << "FileName(3):    " << pDocument->FileName(3) << endl;
    cout << endl;
    id = pDocument->DocId();
    pDocument->Next();
  }

  cout << "*** EcxDocument test complete ***" << endl;

  retval = 0;
  return(retval);
}
```

EcxDocument Class Reference

Interface `ecxdocument.h`

Superclasses `EcxBase`

Subclasses None

Friend Classes None

Syntax `class EcxDocument : public EcxBase { ... };`

Constants and Data Types

The following definitions, which are defined at file scope, allow you to specify the kind of list you want to create:

Syntax `#define ECXDOCUMENT_GET_READ 1`
`#define ECXDOCUMENT_GET_UNREAD 2`

`ECXDOCUMENT_GET_READ`	Include documents that have been accessed.
`ECXDOCUMENT_GET_UNREAD`	Include documents that have not been accessed.

Constructor and Destructor

EcxDocument( )

Creates an `EcxDocument` object.

Syntax `EcxDocument(void);`

`EcxDocument(EcxLogin& login);`

Parameters The constructor has the following parameters:

`login`	The login object to associate with this member object.

Discussion The first form of the constructor allows you to create a stack-based object. The second form of the constructor requires that you create an `EcxLogin` object before you create this object.

Example See "Using the EcxDocument Class" on page 211.

See also The `SetLogin()` method on page 221. The `EcxLogin` class on page 127.

~EcxDocument( )

Destroys an `EcxDocument` object.

Syntax `virtual ~EcxDocument(void);`

Discussion The destructor is called when you delete the object. You can reuse an object instead of deleting it by calling the object's `Clear()` method. The destructor does not destroy the associated `EcxLogin` object.

See also The `Clear()` method on page 215.

Methods

This section describes the methods of the `EcxDocument` class.

CardCount( )

Determines the number of cards associated with the document.

Syntax `const int* CardCount(void) const;`

Returns An integer that contains the number of cards associated with the document.

Example See "Using the EcxDocument Class" on page 211.

CardFlags( )

Accesses information about what card flags have been set.

Syntax `short CardFlags(int cardnum)`

Parameters The `CardFlags()` method has the following parameters:

`cardnum`	An integer that contains the card number.

Returns A short integer that indicates what card flags have been set.

Example See "Using the EcxDocument Class" on page 211.

CardIOType( )

Determines the card input/output type.

Syntax `short CardIOType(int cardnum)`

Parameters The `CardIOType()` method has the following parameters:

`cardnum`	An integer that contains the card number.

Returns A short integer that indicates the card input/output type.

Example See "Using the EcxDocument Class" on page 211.

Clear( )

Clears the state associated with an object, including its list.

Syntax `void Clear(void);`

Discussion All fields in the object are reset to 0 or NULL. A list contains no records.

Example See "Using the EcxDocument Class" on page 211.

CreationDate( )

Determines the date the document was created.

Syntax `const long CreationDate(void) const;`

Returns A long integer that indicates the date the document was created.

Example See "Using the EcxDocument Class" on page 211.

DataState( )

Determines the state the document data is in.

Syntax `short DataState(void) const;`

Returns A short integer that indicates what state the document data is in.

Discussion You can receive any of the following values:

Description	Value
`DSunknown`	0
`DSready for purge`	1
`DSpurged`	2
`DSready for archive`	3
`DSarchived`	4
`DSready for restore`	5
`DSrestored`	6

Example See "Using the EcxDocument Class" on page 211.

Delete( )

Deletes document records from the database.

Syntax `EcxDocument& Delete(void);`

Returns A reference to this document object.

Discussion You must be an administrator and be logged in before calling this method.

DocId( )

Determines the document ID.

Syntax `EcxDocId& DocId(void);`

Returns A reference to an `EcxDocId` object that contains the document ID.

Example See "Using the EcxDocument Class" on page 211.

See also The `EcxDocId` class on page 225.

DocType( )

Determines the document type.

Syntax `const char* DocType(void) const;`

Returns A pointer to a character string that contains the document type.

Example See "Using the EcxDocument Class" on page 211.

FileName( )

Determines the name of the file associated with the document.

Syntax `const char* FileName(int cardnum = 1);`

Parameters The `FileName()` method has the following parameters:

`cardnum`	An integer that contains the card number.

Returns A pointer to a character string that contains the full path name of the file.

Example See "Using the EcxDocument Class" on page 211.

Discussion If you do not specify a card number, the file name associated with the first card is returned. Typically, the first card is an input card.

Get( )

Retrieves a document record from the database.

Syntax `EcxDocument& Get(EcxDocId& docid, const int mark_read = TRUE);`

Parameters The `Get()` method has the following parameters:

`docid`	A reference to the `EcxDocId` object that specifies the retrieval criteria.
`mark_read`	An integer value that specifies whether to mark the document as having been read.

Returns A reference to this document object.

Discussion You call the document ID object's `SetValues()` method to specify the document you wish to retrieve. The logged-in user's name must match the receiver's name for the retrieved document and the values for the document ID, tracking ID, group ID, and interchange ID in the `docid` parameter must match the corresponding values for the retrieved document. You can specify that the document has not been read by setting the `mark_read` parameter to FALSE before calling the `Get()` method; otherwise, the `Get()` method marks the document as having been read.

See also The `EcxDocId::SetValues()` method on page 227.

List( )

Retrieves a list of document records from the database.

Syntax `EcxDocument& List(const int flags = ECXDOCUMENT_GET_UNREAD);`

Parameters The `List()` method has the following parameters:

`flags`	An integer that specifies which documents to retrieve.

Returns A reference to this document object.

Discussion The logged-in user's name must match the receiver's name for the retrieved document. By default, only unread documents are retrieved. You can further control the records that are retrieved by specifying a value in the `flags` parameter or by calling the `SenderName()` method:

- Set the `flags` parameter to `ECXDOCUMENT_GET_READ` to specify retrieval of only documents that have been read.
- Set the flags parameter to both ECXDOCUMENT_GET_READ and ECXDOCUMENT_GET_UNREAD (ECXDOCUMENT_GET_READ | ECXDOCUMENT_GET_UNREAD) to specify retrieval of all document records.
- Call the `SenderName()` method first to restrict the list to include only documents with the specified sender.

After calling the `List()` method, the document object's fields contain values from the record related to the first document in the list.

If you are trying to use this method to retrieve tracking IDs to enable you to retrieve a particular value, but no value is being returned, simply skip to the next tracking ID. For example, if you are using this method to get tracking IDs to retrieve the translated file name from the database, not every tracking ID has a corresponding translated file, because some tracking IDs are generated by bundle.

Example See "Using the EcxDocument Class" on page 211.

See also The `SenderName()` method on page 221. The `XportType()` method on page 225. "Constants and Data Types" on page 213.

ModifyDate( )

Determines the most recent document modification date.

Syntax
```
const long ModifyDate(void) const;
```

Returns A long integer that indicates the most recent document modification date.

Example See "Using the EcxDocument Class" on page 211.

More( )

Determines whether more records are left in the list.

Syntax
```
long More(void);
```

Returns A long integer that contains the number of records not yet accessed from the list.

Discussion After calling the `List()` method and before calling the `Next()` method, the `More()` method returns the total number of records in the list. All records have been accessed when the `More()` method returns 0.

Example See "Using the EcxDocument Class" on page 211.

See also The `List()` method on page 218. The `Next()` method on page 220.

Next( )

Associates the object with the next record in the list.

Syntax `EcxDocument& Next(void);`

Returns A reference to this document object.

Discussion The `Next()` method sets the fields in the object to match those in the next record in the list. The `Next()` method decrements the number of records not yet accessed, which is returned by the `More()` method.

CAUTION Do not call the `Next()` method if the `More()` method returns a value less than 1; the results are unpredictable.

Example See "Using the EcxDocument Class" on page 211.

See also The `More()` method on page 219.

Read( )

Determines whether the document has been read.

Syntax `short Lock(void)const;`

Returns A short integer that indicates whether the document has been read.

Discussion This method will set a document's state to read (pass in 1 or true) or unread (pass in 0 or false).

Example See "Using the EcxDocument Class" on page 211.

Release( )

Determines the document's EDI standard release number.

Syntax `const char* Release(void) const;`

Returns A pointer to a character string that contains the document's EDI standard release number.

Example See "Using the EcxDocument Class" on page 211.

SecondaryTitle( )

Determines the secondary title.

Syntax `const char* SecondaryTitle(void) const;`

Returns A pointer to a character string that contains the title.

Example See "Using the EcxDocument Class" on page 211.

SecondaryValue( )

Determines the secondary value.

Syntax `const char*  SecondaryValue(void) const;`

Returns A pointer to a character string that contains the value.

Example See "Using the EcxDocument Class" on page 211.

SenderName( )

Determines or specifies the sender's member name.

Syntax `const char* SenderName(void) const;`

`void SenderName(const char * name);`

Parameters The `SenderName()` method has the following parameters:

`name`	A pointer to a character string that specifies the name.

Returns The first form of the method returns a pointer to a character string that contains the name.

Discussion Use the first form of the method to determine the sender's member name. Use the second form to specify the name before calling the `List()` method. The `SenderName()` method does not modify the database.

Example See "Using the EcxDocument Class" on page 211.

See also The `List()` method on page 218.

SetLogin( )

Allows the object to access the database.

Syntax `EcxDocument& SetLogin(EcxLogin& login);`

Parameters The `SetLogin()` method has the following parameters:

`login`	A reference to a valid `EcxLogin` object

Returns A reference to this document object.

Discussion If you do not use the form of the constructor that accepts a login object, you must call the `SetLogin()` method before accessing this object.

Example See "Using the EcxDocument Class" on page 211.

See also The `EcxDocument` constructor on page 214. The `EcxLogin` class on page 127.

SetReadyForPurge( )

Sets document to "ready to be purged" state.

Syntax `EcxDocument& SetReadyForPurge(EcxDocId& docid)`

Parameters The `SetReadyForPurge()` method has the following parameters:

`docid`	A reference to the document's ID number

Returns A reference to this document object.

Example See "Using the EcxDocument Class" on page 211.

Standard( )

Determines the document's EDI standard.

Syntax `const char* Standard(void) const;`

Returns A pointer to a character string that contains the document's EDI standard.

Example See "Using the EcxDocument Class" on page 211.

State( )

Determines the document's state.

Syntax `short State(void) const;`

Returns A short integer that specifies the document's state.

Discussion You can receive any of the following values:

Description	Value
`Unknown`	0
`Ready`	1
`In progress`	2
`Done okay`	3
`Done bad`	4
`All done okay`	5
`Bundled`	6

Example See "Using the EcxDocument Class" on page 211.

Title( )

Determines the document's title.

Syntax `const char* Title(void) const;`

Returns A pointer to a character string that contains the title.

TrackState( )

Determines the document's tracking state.

Syntax `void TrackState(const short state);`

Parameters The `TrackState()` method has the following parameters:

`state`	A bitmap that specifies the document's state

Returns A reference to this document object.

Discussion To retrieve a list of docs with a specific state, specify the state before you call the `List()` method. You can receive any of the following states:

Description	Value
`Unknown`	0
`Complete`	1
`In progress`	2
`Warning`	4
`Failed`	8

TranslatedFileName( )

Accesses the name of the translated file.

Syntax `const char* TranslatedFileName(void);`

Returns A pointer to a character string that contains the name of the translated file.

Discussion If you are trying to use this method to retrieve tracking IDs to enable you to retrieve a particular value, but no value is being returned, simply skip to the next tracking ID. For example, if you are using this method to get tracking IDs to retrieve the translated file name from the database, not every tracking ID has a corresponding translated file, because some tracking IDs are generated by bundle.

Value( )

Determines the document's value.

Syntax `const char* Value(void) const;`

Returns A pointer to a character string that contains the value.

Version( )

Determines the document's EDI standard version number.

Syntax `const char* version(void) const;`

Returns A pointer to a character string that contains the document's EDI standard version number.

Example See "Using the EcxDocument Class" on page 211.

XportParam( )

Determines the transport parameter.

Syntax `const char* XportParam(void) const;`

Returns A pointer to a character string that contains the parameter.

XportType( )

Determines the transport protocol.

Syntax `const char* XportType(void) const;`

`void XportType(const char * protocol);`

Parameters The `XportType()` method has the following parameters:

`protocol`	A pointer to a character string that specifies the protocol.

Returns The first form of the method returns a pointer to a character string that contains the protocol.

Discussion Use the first form of the method to determine the protocol. Use the second form to specify the protocol before calling the `List()` method. The `XportType()` method does not modify the database.

Example See "Using the EcxDocument Class" on page 211.

See also The `List()` method on page 218.

About the EcxDocID Class

The `EcxDocID` class represents a key from which documents can be retrieved from the database. You must create an `EcxDocID` object before you can call the partnership's `Get()` and `DocID()` methods. A document ID key consists of the following values:

- tracking ID
- interchange ID
- group ID
- document ID

Methods Summary list:

Constructor and destructor	
`EcxDocID()`	Creates an `EcxDocID` object.
`~EcxDocID()`	Destroys an `EcxDocID` object.
Setting key values	
`SetValues()`	Sets the values associated with a document's key.
Determining key values	
`DocumentID()`	Determines the document ID in the key.
`TrackingID()`	Determines the tracking ID in the key.
`InterchangeID()`	Determines the interchange ID in the key.
`GroupID()`	Determines the group ID in the key.

EcxDocID Class Reference

Interface `ecxdocument.h`

Superclasses None

Subclasses None

Friend Classes None

Syntax `class EcxDocId { ... };`

Constructor and Destructor

EcxDocID( )

Creates an `EcxDocID` object.

Syntax `EcxDocID(void);`

~EcxDocID( )

Destroys an `EcxDocID` object.

Syntax `virtual ~EcxDocID(void);`

Methods

This section describes the methods of the `EcxDocID` class.

DocumentId( )

Determines the document ID in the key.

Syntax `long DocumentId(void);`

Returns A long integer that contains the document ID.

GroupId( )

Determines the group ID in the key.

Syntax `long GroupId(void);`

Returns A long integer that contains the group ID.

InterchangeId( )

Determines the interchange ID in the key.

Syntax `long InterchangeId(void);`

Returns A long integer that contains the interchange ID.

SetValues( )

Sets the values associated with a document's key.

Syntax
```
void SetValues( long trackID,
            long interchangeID,
            long groupID,
            long documentID);
```

Parameters The `SetValues()` method has the following parameters:

`trackID`	A long integer that specifies the tracking ID.
`interchangeID`	A long integer that specifies the interchange ID.

`groupID`	A long integer that specifies the group ID.
`documentID`	A long integer that specifies the document ID.

TrackingId( )

Determines the tracking ID in the key.

Syntax `long TrackingId(void);`

Returns A long integer that contains the tracking ID.

Chapter 14

The EcxTracking Class

This chapter describes the `EcxTracking` class, which represents documents sent from the logged-in user via iPlanet ECXpert. This chapter contains the following sections:

About the EcxTracking Class

The `EcxTracking` class represents documents sent from the logged-in user via iPlanet ECXpert. You can retrieve the tracking status of a document using an `EcxTracking` object.

Methods Summary list:

Constructor and destructor	
`EcxTracking()`	Creates an `EcxTracking` object.
`~EcxTracking()`	Destroys an `EcxTracking` object.
Allowing database access	
`SetLogin()`	Allows the object to access the database.

Listing document records	
`List()`	Retrieves a list of document records from the database.
`More()`	Determines whether more records are left in the list.
`Next()`	Associates the object with the next record in the list.
`Get()`	Retrieves document ID records from the database.
`Delete()`	Deletes a document record.
Resetting an object's state	
`Clear()`	Clears the state associated with an object, including its list.
Accessing document information	
`SecondaryTitle()`	Determines the secondary title.
`SecondaryValue()`	Determines the secondary value.
`ReceiverName()`	Determines receiver's member name.
`State()`	Determines the document's state.
`Title()`	Determines the document's title.
`Value()`	Determines the document's value.
`Progress()`	Determines the document's progress.
`FileName()`	Accesses the file name of the document.
`Standard()`	Determines the document's EDI standard.
`Version()`	Determines the document's EDI standard version number.
`Release()`	Determines the document's EDI standard release number.
`TranslatedFileName()`	Accesses the name of the translated file.
`CreationDate()`	Accesses the date the document was created.
`ModifyDate()`	Accesses the date the document was last modified.
`DocType()`	Determines the document type.
`DataState()`	Determines what state the data is in.
`SetReadyForPurge()`	Specifies whether the document is ready to be purged.

Using the EcxTracking Class

The following example shows how to create an `EcxTracking` object and use it to list the tracking-related records in the database:

```
#include <stdio.h>
#include <fstream.h>

#include "ecxsdk.h"
int main(int argc, char * argv[]) {
  int  retval = -1;

  EcxInit ecxinit;
  EcxLogin * pLogin;
  EcxTracking * pTracking;

  if((pLogin = new EcxLogin())->Errnum()) {
    cout << "EcxLogin Object Error:" << endl;
    cout << "\tErrnum: " << pLogin->Errnum() << endl;
    cout << "\tErrmsg: " << pLogin->Errmsg() << endl;
    cout << endl;
    return(pLogin->Errnum());
  }

  if((pLogin->Login(argv[1], argv[2])).Errnum()) {
    cout << "EcxLogin.Login() Failed:" << endl;
    cout << "\tErrnum: " << pLogin->Errnum() << endl;
    cout << "\tErrmsg: " << pLogin->Errmsg() << endl;
    cout << endl;
    return(pLogin->Errnum());
  }

  cout << "Successful login for user: " << argv[1] << endl;

  if((pTracking = new EcxTracking())->Errnum()) {
    cout << "EcxTracking Object Error:" << endl;
    cout << "\tErrnum: " << pTracking->Errnum() << endl;
    cout << "\tErrmsg: " << pTracking->Errmsg() << endl;
    cout << endl;
    return(pTracking->Errnum());
  }

  if((pTracking->SetLogin(*pLogin)).Errnum()) {
    cout << "EcxTracking.SetLogin() Failed:" << endl;
    cout << "\tErrnum: " << pTracking->Errnum() << endl;
    cout << "\tErrmsg: " << pTracking->Errmsg() << endl;
    cout << endl;
    return(pTracking->Errnum());
  }

  cout << "Created EcxTracking object!" << endl;

  if((pTracking->List()).Errnum()) {
    cout << "EcxTracking.List() Failed:" << endl;
```

```
    cout << "\tErrnum: " << pTracking->Errnum() << endl;
    cout << "\tErrmsg: " << pTracking->Errmsg() << endl;
    return(pTracking->Errnum());
  }

  cout << "*** " << pTracking->More() << " records found. ***" << endl;

  while(pTracking->More()) {
    cout << "---" << endl;
    cout << "ReceiverName:   " << pTracking->ReceiverName() << endl;
    cout << "State:          " << pTracking->State() << endl;
    cout << "Title:          " << pTracking->Title() << endl;
    cout << "Value:          " << pTracking->Value() << endl;
    cout << "SecondaryTitle: " << pTracking->SecondaryTitle() << endl;
    cout << "SecondaryValue: " << pTracking->SecondaryValue() << endl;
    cout << endl;
    pTracking->Next();
  }

  cout << "*** EcxTracking test complete ***" << endl;

  retval = 0;
  return(retval);
}
```

EcxTracking Class Reference

Interface `ecxtracking.h`

Superclasses `EcxBase`

Subclasses None

Friend Classes None

Syntax `class EcxTracking : public EcxBase { ... };`

Class Variables

The following class variables allow you to identify the state of the documents you want to list:

Syntax `static int COMPLETE;`
`static int INPROGRESS;`
`static int WARNING;`
`static int FAILED`
`static int UNKNOWN;`

`COMPLETE`	Document processing is complete.
`INPROGRESS`	Document is being processed.
`WARNING`	Document was processed with a warning.
`FAILED`	Document could not be processed due to errors.
`UNKNOWN`	Document is unknown.

Constructor and Destructor

EcxTracking( )

Creates an `EcxTracking` object.

Syntax `EcxTracking(void);`

EcxTracking(EcxLogin& login);

Parameters The constructor has the following parameters:

`login`	The login object to associate with this tracking object.

Discussion The first form of the constructor allows you to create a stack-based object. The second form of the constructor requires that you create an `EcxLogin` object before you create this object.

Example See "Using the EcxTracking Class" on page 231.

See also The `SetLogin()` method on page 239. The `EcxLogin` class on page 127.

~EcxTracking( )

Destroys an `EcxTracking` object.

Syntax `virtual ~EcxTracking(void);`

Discussion The destructor is called when you delete the object. You can reuse an object instead of deleting it by calling the object's `Clear()` method. The destructor does not destroy the associated `EcxLogin` object.

See also The `Clear()` method on page 234.

Methods

This section describes the methods of the `EcxTracking` class.

Clear( )

Clears the state associated with an object, including its list.

Syntax `void Clear(void);`

Discussion All fields in the object are reset to 0 or NULL. A list contains no records.

CreationDate( )

Accesses the date the document was created.

Syntax `const long CreationDate(void)`

Returns A long integer that contains the date the document record was created.

Example See "Using the EcxTracking Class" on page 231.

Delete( )

Deletes a document record.

Syntax `EcxTracking& Delete`

Returns A reference to this tracking object.

Example See "Using the EcxTracking Class" on page 231.

DataState( )

Determines what state the document data is in.

Syntax `short DataState(void) const;`

Returns A short integer that indicates what state the record data is in.

Discussion You can receive any of the following values:

Description	Value
`DSunknown`	0
`DSready for purge`	1
`DSpurged`	2
`DSready for archive`	3
`DSarchived`	4
`DSready for restore`	5
`DSrestored`	6

Example See "Using the EcxTracking Class" on page 231.

DocType( )

Determines the document type.

Syntax `const char* DocType(void) const;`

Returns A pointer to a character string that indicates the document type.

Example See "Using the EcxTracking Class" on page 231.

FileName( )

Accesses the file name of the document.

Syntax `const char* FileName(void) const;`

Returns A pointer to a character string that contains the document's file name.

Example See "Using the EcxTracking Class" on page 231.

Get( )

Retrieves document ID records from the database.

Syntax `EcxTracking& Get(EcxDocId& docid, const inst mark_read = TRUE);`

Parameters The `Get()` method has the following parameters:

`docid`	A reference to an `EcxDocId` that specifies the document.

Returns A reference to this tracking object.

List( )

Retrieves a list of document records from the database.

Syntax
```
EcxTracking& List( CStr receiver = NULL,
                   const struct tm* fromdate = NULL,
                   const struct tm* todate = NULL,
                   const int state_flag = 0,
                   CStr sender = NULL);
```

Parameters The `List()` method has the following parameters:

`receiver`	A `CStr` structure that specifies the receiver's member name.
`fromdate`	A pointer to a `tm` structure that specifies the starting date.
`todate`	A pointer to a `tm` structure that specifies the ending date.
`state_flag`	An integer that contains the state flags.
`sender`	A `CStr` structure that specifies the sender's member name.

Returns A reference to this tracking object.

Discussion An administrator can specify any sender's member name in the `sender` parameter. A non-administrator can specify only his or her user login name as the sender's member name. If an administrator specifies NULL for the `sender` parameter, which is the default, the sender's member name is not used to select records; all records matching the other criteria are retrieved. If a non-administrator specifies NULL for the `sender` parameter, only document records whose sender's member name match the user's login name and match the other criteria are retrieved.

Values for the remaining criteria, if specified, are ANDed together:

- Specify a value for the `receiver` parameter to restrict retrieval to records for a specific recipient. If you do not specify a value for the `receiver` parameter, all recipients will be considered for retrieval.
- Specify a value for the `fromdate` parameter to restrict retrieval from the specified starting date, inclusive. If you do not specify a value for the `fromdate` parameter, all records will be considered.
- Specify a value for the `todate` parameter to restrict retrieval to the specified ending date, inclusive. If you do not specify a value for the `todate` parameter, all records will be considered.
- Specify one or more flags for the `state_flag` parameter to restrict retrieval to document records that match the specified state. If you do not specify a value for the `state_flag` parameter, all records will be considered. Valid flags are `COMPLETE`, `INPROGRESS`, `WARNING`, and `FAILED`. The flags are ORed together before being ANDed with the other criteria.

After calling the `List()` method, the document object's fields contain values from the record related to the first document that matches the criteria.

If you are trying to use this method to retrieve tracking IDs to enable you to retrieve a particular value, but no value is being returned, simply skip to the next tracking ID. For example, if you are using this method to get tracking IDs to retrieve the translated file name from the database, not every tracking ID has a corresponding translated file, because some tracking IDs are generated by bundle.

Example See "Using the EcxTracking Class" on page 231.

See also "Class Variables" on page 233.

ModifyDate( )

Accesses the date the document was last modified.

Syntax `const long ModifyDate(void) const;`

Returns A long integer that indicates the date the document was last modified

Example See "Using the EcxTracking Class" on page 231.

More( )

Determines whether more records are left in the list.

Syntax `long More(void);`

Returns A long integer that contains the number of records not yet accessed from the list.

Discussion After calling the `List()` method and before calling the `Next()` method, the `More()` method returns the total number of records in the list. All records have been accessed when the `More()` method returns 0.

Example See "Using the EcxTracking Class" on page 231.

See also The `List()` method on page 236. The `Next()` method on page 238.

Next( )

Associates the object with the next record in the list.

Syntax `EcxTracking& Next(void);`

Returns A reference to this tracking object.

Discussion The `Next()` method sets the fields in the object to match those in the next record in the list. The `Next()` method decrements the number of records not yet accessed, which is returned by the `More()` method.

CAUTION Do not call the `Next()` method if the `More()` method returns a value less than 1; the results are unpredictable.

Example See "Using the EcxTracking Class" on page 231.

See also The `More()` method on page 237.

Progress( )

Determines the document's progress.

Syntax `const int Progress(void) const;`

Returns An integer that indicates the document's progress.

Example See "Using the EcxTracking Class" on page 231.

ReceiverName( )

Determines the receiver's member name.

Syntax `const char* ReceiverName(void) const;`

Returns A pointer to a character string that contains the name.

Example See "Using the EcxTracking Class" on page 231.

Release( )

Determines the document's EDI standard release number.

Syntax `const char* Release(void) const;`

Returns A pointer to a character string that indicates the document's EDI standard release number.

Example See "Using the EcxTracking Class" on page 231.

SecondaryTitle( )

Determines the secondary title.

Syntax `const char* SecondaryTitle(void) const;`

Returns A pointer to a character string that contains the title.

Example See "Using the EcxTracking Class" on page 231.

SecondaryValue( )

Determines the secondary value.

Syntax `const char*  SecondaryValue(void) const;`

Returns A pointer to a character string that contains the value.

Example See "Using the EcxTracking Class" on page 231.

SetLogin( )

Allows the object to access the database.

Syntax `EcxTracking& SetLogin(EcxLogin& login);`

Parameters The `SetLogin()` method has the following parameters:

`login`	A reference to a valid `EcxLogin` object

Returns A reference to this tracking object.

Discussion If you do not use the form of the constructor that accepts a login object, you must call the `SetLogin()` method before accessing this object.

Example See "Using the EcxTracking Class" on page 231.

See also The `EcxTracking` constructor on page 233. The `EcxLogin` class on page 127.

SetReadyForPurge( )

Specifies whether the document is ready to be purged.

Syntax `EcxTracking& SetReadyForPurge(EcxDocId& docid)`

Parameters The `SetReadyForPurge()` method has the following parameters:

`docid`	A reference to the document's ID number

Returns A reference to this tracking object.

Example See "Using the EcxTracking Class" on page 231.

Standard( )

Determines the document's EDI standard.

Syntax `const char* Standard(void) const;`

Returns A pointer to a character string that contains the document's EDI standard.

Example See "Using the EcxTracking Class" on page 231.

State( )

Determines the document's state.

Syntax `short State(void) const;`

Returns A short integer that specifies the document's state.

Discussion You can receive any of the following values:

Description	Value
`TSunknown - indicates NULL value`	0
`TSready - indicates service has yet to be invoked`	1
`TSinProgress - indicates service has been invoked`	2
`STSdoneOK - indicates service is done with no errors`	3

Description	Value
TSdoneBad - indicates service is done with errors	4
TSalldoneOK - indicates last service on service list is done and TRKState is TSdoneOK	5
TSbundled - identifies bundle generated trackings	6

Example See "Using the EcxTracking Class" on page 231.

Title( )

Determines the document's title.

Syntax `const char* Title(void) const;`

Returns A pointer to a character string that contains the title.

Example See "Using the EcxTracking Class" on page 231.

TranslatedFileName( )

Accesses the name of the translated file.

Syntax `const char* TranslatedFileName(void);`

Returns A pointer to a character string that contains the name of the translated file.

Discussion If you are trying to use this method to retrieve tracking IDs to enable you to retrieve a particular value, but no value is being returned, simply skip to the next tracking ID. For example, if you are using this method to get tracking IDs to retrieve the translated file name from the database, not every tracking ID has a corresponding translated file, because some tracking IDs are generated by bundle.

Value( )

Determines the document's value.

Syntax `const char* Value(void) const;`

Returns A pointer to a character string that contains the value.

Example See "Using the EcxTracking Class" on page 231.

Version( )

Determines the document's EDI standard version number.

Syntax `const char* version(void) const;`

Returns A pointer to a character string that contains the document's EDI strandard version number.

Example See "Using the EcxTracking Class" on page 231.

Chapter 15

The EcxLog Class

This chapter describes the `EcxLog` class, which represents entries in the iPlanet ECXpert log. This chapter contains the following sections:

About the EcxLog Class

The `EcxLog` class represents entries in the iPlanet ECXpert log. You can use an `EcxLog` object to add an entry to the log.

Methods Summary list:

Constructor and destructor	
`EcxLog()`	Creates an `EcxLog` object.
`~EcxLog()`	Destroys an `EcxLog` object.
Allowing database access	
`SetLogin()`	Allows the object to access the database.
Logging an event	
`LogEvent()`	Adds an entry to the log.

Resetting an object's state	
`Clear()`	Clears the state associated with an object.
Accessing log information	
`Next`	Associates the object with the next record in the list.
`More`	Determines whether more records are left in the list.
`RetrieveLog`	Retrieves log information.
`ELId`	Determines the ID number of the event in the event log.
`ELEventId`	Determines the ID number of the event in the event log.
`ELCategory`	Determines the category of the event in the event log.
`ELSeverity`	Determines the severity of the event in the event log.
`ELEventShortMsg`	Determines the short message associated with the event in the event log.
`ElTrkId`	Determines the tracking ID of the event in the event log.
`ElIntgId`	Determines the interchange identifier.
`ELGrpId`	Determines the group ID of the event in the event log.
`ElDocId`	Determines the ID number of the document in the event log.
`ElTDId`	Determines the document-level internal tracking ID associated with the event.

Using the EcxLog Class

The following example shows how to write infromational messages, warning messages, and fatal error messages to the iPlanet ECXpert log:

```
int main(int argc, char * argv[]) {
...
if((pLog = new EcxLog())->Errnum()) {
    cout << "EcxLog Object Error:" << endl;
    cout << "\tErrnum: " << pLog->Errnum() << endl;
    cout << "\tErrmsg: " << pLog->Errmsg() << endl;
    cout << endl;
    return(pLog->Errnum());
  }

  if((pLog->SetLogin(*pLogin)).Errnum()) {
    cout << "EcxLog.SetLogin() Failed:" << endl;
    cout << "\tErrnum: " << pLog->Errnum() << endl;
    cout << "\tErrmsg: " << pLog->Errmsg() << endl;
```

```
    cout << endl;
    return(pLog->Errnum());
  }

  cout << "Created EcxLog object!" << endl;

  if((pLog->LogEvent(99,
           pLog->informational,
           "This is a informational TEST message")).Errnum()) {
    cout << "EcxLog.LogEvent() Failed:" << endl;
    cout << "\tErrnum: " << pLog->Errnum() << endl;
    cout << "\tErrnum: " << pLog->Errnum() << endl;
    cout << "\tErrmsg: " << pLog->Errmsg() << endl;
    return(pLog->Errnum());
  }

  cout << "WROTE: This is a informational TEST message" << endl;

  if((pLog->LogEvent(99,
           pLog->warning,
           "This is a warning TEST message")).Errnum()) {
    cout << "EcxLog.LogEvent() Failed:" << endl;
    cout << "\tErrnum: " << pLog->Errnum() << endl;
    cout << "\tErrmsg: " << pLog->Errmsg() << endl;
    return(pLog->Errnum());
  }

  cout << "WROTE: This is a warning TEST message" << endl;

  if((pLog->LogEvent(99,
           pLog->error,
           "This is a error TEST message")).Errnum()) {
    cout << "EcxLog.LogEvent() Failed:" << endl;
    cout << "\tErrnum: " << pLog->Errnum() << endl;
    cout << "\tErrmsg: " << pLog->Errmsg() << endl;
    return(pLog->Errnum());
  }

  cout << "WROTE: This is an error TEST message" << endl;

  cout << "*** EcxLog test complete ***" << endl;

  retval = 0;
  return(retval);
}
```

EcxLog Class Reference

Interface `ecxlog.h`

Superclasses `EcxBase`

Subclasses None

Friend Classes None

Syntax `class EcxLog : public EcxBase { ... };`

Class Variables

The following class variables allow you to identify the kind of message being written to the database:

Syntax
```
const int informational;
const int warning;
const int error;
```

`informational`	Informational message.
`warning`	Warning message.
`error`	Fatal error message.

Constructor and Destructor

EcxLog( )

Creates an `EcxLog` object.

Syntax
```
EcxLog(void);
EcxLog(EcxLogin& login);
```

Parameters The constructor has the following parameters:

`login`	The login object to associate with this tracking object.

Discussion The first form of the constructor allows you to create a stack-based object. The second form of the constructor requires that you create an `EcxLogin` object before you create this object.

Example See "Using the EcxLog Class" on page 244.

See also The `SetLogin()` method on page 251. The `EcxLogin` class on page 127.

~EcxLog( )

Destroys an `EcxLog` object.

Syntax `virtual ~EcxLog(void);`

Discussion The destructor is called when you delete the object. The destructor does not destroy the associated `EcxLogin` object.

Methods

This section describes the methods of the `EcxLog` class.

Clear( )

Clears the state associated with an object, including its list.

Syntax `void Clear()`

Discussion All fields in the object are reset to 0 or NULL. A list contains no records.

ELCategory( )

Determines the functional area the event took place in.

Syntax `const char* ELCategory() const;`

Returns A pointer to a character string that contains the functional area the event took place in (e.g. bundle, dispatcher, parse, etc.).

Example See "Using the EcxLog Class" on page 244.

ELDocId( )

Determines the ID number of the document in event log.

Syntax `const char* ELDocId`

Returns A pointer to a character string that contains the document ID number.

Example See "Using the EcxLog Class" on page 244.

ELEventId( )

Determines ID number associated with event in event log.

Syntax `unsigned ELEventId() const;`

Returns An unsigned integer that contains the ID number associated with event in event log.

Example See "Using the EcxLog Class" on page 244.

ELEventShortMsg( )

Determines the short message associated with the event in the event log.

Syntax `const char* ELEventShortMsg() const;`

Returns A pointer to a character string that contains the short message associated with event in event log.

Example See "Using the EcxLog Class" on page 244.

ELGrpId( )

Determines the group ID of the event in the event log.

Syntax `unsigned ELGrpId() const;`

Returns Unsigned integer that contains the group ID of event in event log.

Example See "Using the EcxLog Class" on page 244.

ELId( )

Determines the ID number of the event in the event log.

Syntax `unsigned ELId () const;`

Returns Unsigned integer that contains the ID number of event in event log.

Example See "Using the EcxLog Class" on page 244.

ELIntgId( )

Determines the interchange identifier.

Syntax `unsigned ELIntgId () const;`

Returns Unsigned integer that contains the interchange identifier.

Example See "Using the EcxLog Class" on page 244.

ELSeverity( )

Severity associated with the event in the event log.

Syntax `unsigned ELSeverity () const;`

Returns Unsigned integer that contains the severity associated with event in event log.

Discussion The level of severity can be informational, warning, or error.

Example See "Using the EcxLog Class" on page 244.

ELTDId( )

Determines the document-level internal tracking ID associated with the event.

Syntax `const char* ELTDId() const;`

Returns A pointer to a character string that contains the document-level internal tracking ID associated with an event.

Example See "Using the EcxLog Class" on page 244

ELTrkId( )

Track ID of the event in the event log.

Syntax `unsigned ELTrkId`

Returns Unsigned integer that contains the tracking ID of event in event log.

Example See "Using the EcxLog Class" on page 244.

LogEvent( )

Adds an entry to the event log.

Syntax
```
EcxLog& LogEvent(long errnum, int severity, const char *
message);
```

Parameters The `LogEvent()` method has the following parameters:

`errnum`	A long integer that specifies the error number you want to associate with the entry.
`severity`	An integer that specifies the kind of entry.
`message`	A pointer to a character string that specifies the messge to write to the log.

Returns A reference to this log object.

Discussion You can specify one of the following constant for the kind of entry: `informational`, `warning`, or `fatal`. The user name of the logged-in user is also written to the log.

NOTE The tracking ID written to the log is always 0.

Example See "Using the EcxLog Class" on page 244.

See also "Class Variables" on page 246.

More( )

Determines whether more records are left in the list.

Syntax `long More(void);`

Returns A long integer that contains the number of records not yet accessed from the list.

Discussion After calling the List() method and before calling the `Next()` method, the `More()` method returns the total number of records in the list. All records have been accessed when the `More()` method returns 0.

Example See "Using the EcxLog Class" on page 244.

See also The `Next()` method on page 250.

Next( )

Associates the object with the next record in the list.

Syntax `EcxDocument& Next(void);`

Returns A reference to this document object.

Discussion The `Next()` method sets the fields in the object to match those in the next record in the list. The `Next()` method decrements the number of records not yet accessed, which is returned by the `More()` method.

CAUTION Do not call the `Next()` method if the `More()` method returns a value less than 1; the results are unpredictable.

Example See "Using the EcxLog Class" on page 244.

See also The `More()` method on page 250.

RetrieveLog( )

Retrieves log information.

Syntax
```
EcxLog& RetrieveLog( const unsigned&trkId,
                     const char* sndrMBName,
                     const char* rcvrMBName,
                     const long fromdt,
                     const long todt,
                     const short stateBitmap);
```

Parameters The `RetrieveLog()` method has the following parameters:

`sndrMBName`	A pointer to a character string that specifies the sender member name.
`rcvrMBName`	A pointer to a character string that specifies the receiver name.
`fromdt`	A long integer that specifies the initial ("from") date.
`todt`	A long integer that specifies the final ("to") date
`stateBitmap`	Data state. Valid values: 0 = unknown 1 = readyForPurge 2 = purged 3 = readyForArchive 4 = archived 5 = readyForRestore 6 = restored

Returns A pointer to this `RetrieveLog` object.

Example See "Using the EcxLog Class" on page 244

SetLogin( )

Allows the object to access the database.

Syntax
```
EcxTracking& SetLogin(EcxLogin& login);
```

Parameters The `SetLogin()` method has the following parameters:

`login`	A reference to a valid `EcxLogin` object

Returns A reference to this tracking object.

Discussion If you do not use the form of the constructor that accepts a login object, you must call the `SetLogin()` method before accessing this object.

Example See "Using the EcxLog Class" on page 244.

See also The `EcxLog` constructor on page 246. The `EcxLogin` class on page 127.

Chapter 16

The EcxFtpClient Class

This chapter describes the `EcxFtpClient` class. The `EcxFtpClient` is an FTP Client API. The EcxFtpClient class defines methods you can use to send and receive files via FTP.

This chapter contains the following sections:

About the EcxFtpClient Class

The `EcxFtpClient()` class is an FTP Client API which defines methods you can use to send and receive files via FTP. The `EcxFtpClient()` class is based on the RFC 959 FTP protocol.

Before you will be able to perform any FTP operations, you must first create the `EcxFtpClient()` object and call the `init()` method. You can then run FTP commands using the `RunCommand()` method.

Methods Summary list:

Constructor and destructor	
`EcxFtpClient(void)`	Creates an `EcxFtpClient` object.
`virtual ~EcxFtpClient()`	Destroys an `EcxFtpClient` object.
Initializing the FTP Client API	
`Init`	Initializes the FTP client API

Accessing Entry Information	
`GetListCount()`	Retrieves the number of files in the current directory listing
`GetFirstListEntry()`	Retrieves the first file in the directory listing
`GetNextListEntry()`	Retrieves the next file in the directory listing
Accessing FTP Replies	
`GetReplyCode()`	Retrieves the last reply code
`GetReplyMsg()`	Retrieves the last reply message
`IsReplyGood`	Indicates whether the last FTP command executed was successful or not
Running Commands	
`RunCommand`	Runs a command

Using the EcxFtpClient Class

The following sections show how to:

- List files in the current directory
- Retrieve the names of files in the current directory listing
- Send and receive files

Listing Files in the Current Directory

The following example shows how to list all of the files in the current directory. The `RunCommand()` method runs the `ls` and `dir` commands to generate a directory listing.

```
int main(int argc, char * argv[])
{
   int  retval = -1;

   //
   // List of ftp commands that we would be running using the Ecxpert
   // ftp client API. We basically login to the remote machine, run
   // 'ls' and 'dir' commands and dump the output on the console.
   //
```

```
char *          FtpCommands[] =
                {
                   "open myhost.myserver.com",
                   "user actraadm actraadm",
                   "ls /tmp",
                   "dir /tmp",
                   "quit",
                   ""
                };

const char *    pListEntry = 0;

EcxInit         EcxInitObj;

EcxFtpClient *  pFtpClientObj = 0;


do
{
    if ( EcxInitObj.Errnum() != 0 )
    {
        printf("Failed to initialize EcxInit object.\n");
        break;
    }

    if ( (pFtpClientObj = new EcxFtpClient) == 0 )
    {
        printf("No memory to create Ecxpert ftp client object.\n");
        break;
    }

    if ( pFtpClientObj->Init("ecx.ini").Errnum() )
    {
        printf("Failed to setup Ecxpert ftp client object.\n");
        break;
    }

    for ( int i = 0; strlen(FtpCommands[i]) != 0; ++i )
    {
        printf("\nExecuting Ftp command - %s\n", FtpCommands[i]);

        if ( pFtpClientObj->RunCommand(FtpCommands[i]).Errnum() )
        {
            printf("Error: %ld - Could not execute command.\n",
                   pFtpClientObj->Errnum());
            break;
        }

        printf("Ftp reply code = %d\n", pFtpClientObj->GetReplyCode());
        printf("Ftp reply message = %s\n", pFtpClientObj->GetReplyMsg());

        if ( pFtpClientObj->IsReplyGood() != TRUE )
        {
            printf("Command could not be executed successfully.\n");
        }
        else
```

```
            {
                //
                // Display the output of the ls/dir command
                //
                printf("Remote directory consists of %d entries.\n\n",
                       pFtpClientObj->GetListCount());

                pListEntry = pFtpClientObj->GetFirstListEntry();

                while( pListEntry != 0 )
                {
                    printf("%s\n", pListEntry);
                    pListEntry = pFtpClientObj->GetNextListEntry();
                }
            }
        }

        retval = pFtpClientObj->Errnum();
    }
    while( 0 );

    if ( pFtpClientObj )
        delete pFtpClientObj;

    return(retval);
}
```

Retrieving File Names

The following example shows how to retrieve file names from the directory listing. The `GetFirstListEntry()` and `GetNextListEntry()` methods retrieve retrieve the first and all subsequent file names from the directory listing.

```
int main(int argc, char * argv[])
{
    long    retval = -1;

    char    szTmpBuff[2048];

    const char *    pListEntry = 0;

    EcxInit         EcxInitObj;

    EcxFtpClient *  pFtpClientObj = 0;


    do
    {
        if ( EcxInitObj.Errnum() != 0 )
```

```
        {
            printf("Failed to initialize EcxInit object.\n");
            break;
        }

        if ( (pFtpClientObj = new EcxFtpClient) == 0 )
        {
            printf("No memory to create Ecxpert ftp client object.\n");
            break;
        }

        if ( pFtpClientObj->Init("ecx.ini").Errnum() )
        {
            printf("Failed to setup Ecxpert ftp client object.\n");
            break;
        }

        do
        {
            printf("ecxftp> ");

            gets(szTmpBuff);

            if ( pFtpClientObj->RunCommand(szTmpBuff).Errnum() )
            {
                printf("Error: %ld - Could not execute command.\n",
                       pFtpClientObj->Errnum());
                break;
            }

            printf("Ftp reply code = %d\n", pFtpClientObj->GetReplyCode());
            printf("Ftp reply message = %s\n", pFtpClientObj->GetReplyMsg());

            if ( pFtpClientObj->IsReplyGood() != TRUE )
            {
                printf("Command could not be executed successfully.\n");
            }
            else if ( pFtpClientObj->GetListCount() > 0 )
            {
                pListEntry = pFtpClientObj->GetFirstListEntry();

                while( pListEntry != 0 )
                {
                    printf("%s\n", pListEntry);
                    pListEntry = pFtpClientObj->GetNextListEntry();
                }
            }
        }
        while( strcmp(szTmpBuff, "quit") != 0 );

        retval = pFtpClientObj->Errnum();
    }
    while( 0 );

    if ( pFtpClientObj )
        delete pFtpClientObj;
```

```
   return(retval);
}
```

Transferring Files

The following example shows how to send and receive files using the EcxFtpClient API. The `RunCommand()` method runs the FTP `get` and `put` commands to transfer an ascii file and a binary file.

```
int main(int argc, char * argv[])
{
   long    retval = -1;

   //
   // List of ftp commands that we would be running using the Ecxpert
   // ftp client API. We basically login to the remote machine and
   // run get and put commands to transfer an ascii file and a binary file.
   //
   char *          FtpCommands[] =
                   {
                      "open flatline.mcom.com",
                      "user smani2 smani2",
                      "get remote-ascii-file local-ascii-file",
                      "put local-ascii-file remote-ascii-file.bak",
                      "binary",
                      "get remote-binary-file local-binary-file",
                      "put local-binary-file remote-binary-file.bak",
                      "quit",
                      ""
                   };

   EcxInit         EcxInitObj;

   EcxFtpClient *  pFtpClientObj = 0;

   do
   {
       if ( EcxInitObj.Errnum() != 0 )
       {
           printf("Failed to initialize EcxInit object.\n");
           break;
       }

       if ( (pFtpClientObj = new EcxFtpClient) == 0 )
       {
           printf("No memory to create Ecxpert ftp client
              object.\n");
           break;
```

```
        }

        if ( pFtpClientObj->Init("ecx.ini").Errnum() )
        {
            printf("Failed to setup Ecxpert ftp client object.\n");
            break;
        }

        for ( int i = 0; strlen(FtpCommands[i]) != 0; ++i )
        {
            printf("\nExecuting Ftp command - %s\n", FtpCommands[i]);

            if ( pFtpClientObj->RunCommand(FtpCommands[i]).Errnum() )
            {
                printf("Error: %ld - Could not execute command.\n",
                       pFtpClientObj->Errnum());
                break;
            }

            printf("Ftp reply code = %d\n", pFtpClientObj->GetReplyCode());
            printf("Ftp reply message = %s\n", pFtpClientObj->GetReplyMsg());

            if ( pFtpClientObj->IsReplyGood() != TRUE )
            {
                printf("Command could not be executed successfully.\n");
            }
        }

        retval = pFtpClientObj->Errnum();
    }
    while( 0 );

    if ( pFtpClientObj )
        delete pFtpClientObj;

    return(retval);
```

EcxFtpClient Class Reference

Interface `ecxftpclient.h`

Superclasses `EcxBase`

Subclasses None

Friend Classes None

Syntax `class EcxFtpClient : public EcxBase { ... };`

Constructor and Destructor

EcxFtpClient( )

Creates an `EcxFtpClient` object.

Syntax `EcxFtpClient(void);`

Example See "Using the EcxFtpClient Class" on page 254.

~EcxFtpClient( )

Destroys an `EcxFtpClient` object.

Syntax `virtual ~EcxFtpClient();`

Example See "Using the EcxFtpClient Class" on page 254.

Methods

This section describes the methods of the `EcxFtpClient` class.

GetListCount ( )

Retrieves the number of files in the current directory.

Syntax `virtual int GetListCount(void)`

Returns The number of files in the current directory.

Discussion After running the `ls` or `dir` command, this method retrieves the number of files in the directory listing.

Example See "Listing Files in the Current Directory" on page 254.

GetFirstListEntry ( )

Retrieves the name of the first file in the directory listing.

Syntax `virtual const char* GetFirstListEntry(void)`

Returns A pointer to a character string that contains the name of the first file in the directory listing.

Discussion After running the `ls` or `dir` command, this method retrieves the first file in the directory listing.

Example See "Listing Files in the Current Directory" on page 254.

GetNextListEntry ( )

Retrieves the name of the next file in the directory listing.

Syntax `virtual const char* GetNextListEntry(void)`

Returns A pointer to a character string that contains the name of the next file in the directory listing.

Example See "Listing Files in the Current Directory" on page 254.

GetReplyCode ( )

Retrives the reply code for the last command executed.

Syntax `virtual int GetReplyCode(void)`

Returns A pointer to an integer representing the reply code for the last command executed.

Example See "Using the EcxFtpClient Class" on page 254.

GetReplyMsg ( )

Retrieves the reply message for the last command executed.

Syntax `virtual const char* GetReplyMsg(void)`

Returns A pointer to a character string that contains the reply message for the last command executed.

Example See "Using the EcxFtpClient Class" on page 254.

Init ( )

Initializes the FTP client API.

Syntax `virtual EcxFtpClient& Init(const char* pEcxIniFileName)`

Parameters The `Init()` method has the following parameters:

`pEcxIniFileName`	A pointer to a character string that contains the full path to the ECXpert initialization file

Returns A reference to this `EcxFtpClient` object.

Discussion This method must be called before you can call the `RunCommand()` method.

Example See "Using the EcxFtpClient Class" on page 254.

IsReplyGood ( )

Indicates whether the last FTP command executed was successful or not.

Syntax `virtual int IsReplyGood(void)`

Returns Returns a 0 or 1. A value of 0 indicates that the last FTP command failed, and a value of 1 indicates that the last FTP command executed successfully.

Example See "Using the EcxFtpClient Class" on page 254.

RunCommand ( )

Runs a command.

Syntax `virtual EcxFtpClient& RunCommand(const char* pCmdString)`

Parameters The `RunCommand()` method has the following parameters:

`pCmdString`	A character string that contains the FTP client command to be run

Returns A reference to this `EcxFtpClient` object.

Example See "Using the EcxFtpClient Class" on page 254.

Chapter 17

The EcxService Class

This chapter describes the `EcxService` class, which represents service records in an iPlanet ECXpert database. This chapter contains the following sections:

About the EcxService Class

The `EcxService()` class represents service records in an iPlanet ECXpert database. Only administrators can add, change, or delete a service record. A user must be logged in to the database before accessing a record.

Methods Summary list:

Constructor and destructor	
`EcxService(void)`	Creates an `EcxService` object.
`virtual ~EcxService(void)`	Destroys an `EcxService` object.
Allowing database access	
`SetLogin`	Allows the object to access the database.

Adding, retrieving, changing and deleting service records	
`Add`	Adds a service record to the database.
`Change`	Changes a service record in the database.
`Delete`	Deletes a service from the database.
`Get`	Retrieves a service record from the database.
Listing service records	
`List`	Retrieves a list of service records from the database
`More`	Determines whether more records are left in the list.
`Next`	Associates the object with the next record in the list.
Resetting an object's state	
`Clear`	Clears the state associated with an object, including its list
Accessing key fields	
`Id`	Determines or specifies the ID number of the service.
Accessing other fields	
`Name`	Determines or specifies the name of the service.
`Type`	Determines or specifies the service type.
`PathName`	Determines or specifies the path name to the service code file.
`EntryName`	Determines or specifies the entry name of the service.
`MaxThread`	Determines or specifies the maximum number of threads the service can have.
`Param`	Determines or specifies the service description.
`ObjPerm`	Determines or specifies the record's access permissions.
`ModByGroup`	Determines the group that last modified the record.
`ModByUser`	Determines the user that last modified the record.
`ModDt`	Determines the date the record was last modified.

Using the EcxService Class

The following sections show how to:

- Create a service object
- Add a service
- List all services
- Modify a service
- Delete a service

Creating a Service Object

The following example shows how to create a Service object.

```
ECXService * pService = NULL;

if((pService = new EcxService())->Errnum())  {
  cout << “EcxServiceObjectError:” << endl;
  cout << “\tErrnum: “ << pService->Errnum() << endl;
  cout << “\tErrmsg: “ << pService->Errmsg() << endl;
  return(NULL);

}

if((pService->SetLogin(*pLogin)).Errnum())  {
  cout << “EcxService.SetLogin() Failed:” << endl;
  cout << “\tErrnum: “ << pService->Errnum() << endl;
  cout << “\tErrmsg: “ << pService->Errmsg() << endl;
  return(NULL);

return (pService);
```

Adding a Service

The following example shows how to add a service.

```
pService->Clear();

pService->Name(“Test service”);
pService->Type(10);
```

```
pService->PathName("TestPathName");
pService->EntryName("TestEntryName");
pService->MaxThread(5);
pService->Param("Test param");
pService->ObjPerm(755);

if((pService->Add()).Errnum())  {
  cout << "EcxService.add() Failed" << endl;
  cout << "\tErrnum: " << pService->Errnum() << endl;
  cout << "\tErrmsg: " << pService->Errmsg() << endl;
  return(NULL);
}

id = pService->Id();

cout << "*** Added service: " << id << endl;

return(0);
```

Listing All Services

The following example shows how to generate a list of all services.

```
pService->Clear();

If((pService->List()).Errnum())  [
  cout << "EcxService.List() Failed:" << endl;
  cout << "\tErrnum: " << pService->Errnum() << endl;
  cout << "\tErrmsg: " << pService->Errmsg() << endl;
  return (pService->Errnum());
}

cout << "*** Listing Services" << pService ->More();
cout << " records found. ***" << endl;

while (pService->More())  {
  cout << pService->Id()          << ":";
  cout << pService->Name()        << ":";
  cout << pService->Type()        << ":";
  cout << pService->PathName()       << ":";
  cout << pService->EntryName()      << ":";
  cout << pService->MaxThread()      << ":";
  cout << pService->Param()          << ":";
  cout << pService->ObjPerm()        << ":";
  cout << pService->ModByGroup()     << ":";
  cout << pService->ModByUser()      << ":";
  cout << pService->ModDt()          << ":";
  pService->Next(;
```

```
}

return(0);
```

Modifying a Service

The following example shows how to modify a service.

```
pService->Clear();
pService->Id(id);

if((pService->Get()).Errnum())  {
  cout << “EcxService.Get() Failed” << endl;
  cout << “\tErrnum: “ << pService->Errnum() << endl;
  cout << “\tErrmsg: “ << pService->Errmsg() << endl;
  return(pService->Errnum());
}

pservice->Type(20);

if((pService->Change()).Errnum())  {
  cout << “EcxService.Change() Failed:” << endl;
  cout << “\tErrnum: “ << pService->Errnum() << endl;
  cout << “\tErrmsg: “ << pService->Errmsg() << endl;
  return(pService->Errnum());
}

return(0);
```

Deleting a Service

The following example shows how to delete a service.

```
pService->Clear();
pService->Id(id);

if((pService->Delete()).Errnum())  {
  cout << “EcxService.Delete() Failed” << endl;
  cout << “\tErrnum: “ << pService->Errnum() << endl;
  cout << “\tErrmsg: “ << pService->Errmsg() << endl;
  return(pService->Errnum());
}
```

```
cout << "*** Deleted service: " << id << endl;
return(0);
```

EcxServiceClass Reference

Interface `ecxservice.h`

Superclasses `EcxBase`

Subclasses None

Friend Classes None

Syntax `class EcxService : public EcxBase { ... };`

Class Variables

The following class variables allow you to identify the member as an administrator:

Syntax `static int ADMINISTRATOR;`

`ADMINISTRATOR`	Administrator

Constructor and Destructor

EcxService(void)

Creates an `EcxService` object.

Syntax `EcxService(void);`
`EcxService(EcxLogin&login)`

Example See "Using the EcxService Class" on page 265.

~EcxService(void)

Destroys an `EcxService` object.

Syntax `virtual ~EcxService(void);`

Example See "Using the EcxService Class" on page 265.

Methods

This section describes the methods of the `EcxService` class.

Add ( )

Adds a service record to the database.

Syntax `Ecxservice& Add(void);`

Returns A reference to this service object.

Discussion You must be an administrator and be logged in before calling this method. You must specify the service's ID number in the object, by calling the `Id()` method, before calling this method.

Example See "Adding a Service" on page 265.

See also The `Id()` method on page 271.

Change( )

Changes a service record in the database.

Syntax `EcxService& Change(void);`

Returns A reference to this service object.

Discussion You must be an administrator and be logged in before calling this method. This method updates the last record retrieved by calling the object's `Get()`, `List()`, or `Next()` method. You must specify the service's ID number in the object, by calling the `Id()` method, before calling this method.

CAUTION If you do not call the object's `Get()`, `List()`, or `Next()` method first, the object's ID number field, which is set by calling the `Id()` method, specifies the record that is changed. In this case, the record is completely overwritten using the object's fields. Any fields not set in the object will be replaced by 0 or NULL in the database.

Example See "Modifying a Service" on page 267.

See also The `Get()` method on page 270. The `List()` method on page 271. The `Next()` method on page 273. The `Id()` method on page 271.

Clear( )

Clears the state associated with an object, including its list.

Syntax `void Clear(void);`

Example See "Adding a Service" on page 265.

Delete( )

Deletes a service from the database.

Syntax `EcxService& Delete(void);`

Returns A reference to this service object.

Discussion You must be an administrator and be logged in before calling this method. You must specify the service's ID number in the object, by calling the `Id()` method, before calling this method.

Example See "Deleting a Service" on page 267.

See also The `Id()` method on page 271.

EntryName ( )

Determines or specifies the entry name of the service.

Syntax `const char * Name() const;`

void EntryName(const char*);

Returns The first form of the method returns a pointer to a character string that contains the entry name of the service.

Discussion Use the first form of the method to determine the service entry name. Use the second form to specify the service entry name.

Example See "Adding a Service" on page 265.

Get( )

Retrieves a service record from the database.

Syntax `EcxService& Get(void);`

Returns A reference to this service object.

Discussion You must specify the service's ID number in the object, by calling the `Id()` method, before calling this method.

Example See "Adding a Service" on page 265.

See also The `Id()` method on page 271.

Id ( )

Determines or specifies the ID number of the service.

Syntax `unsigned int Id()const;`

void Id(const unsigned int)

Returns The first form of the method returns an unsigned integer that contains the ID number of the service.

Discussion Before you call the `Add()`, `Change()`, `Delete()`, or `Get()` methods, you must first specify the service's ID number in the object by calling the `Id()` method. Use the first form of the method to determine the service's ID number. Use the second form to specify the service's ID number.

Example See "Adding a Service" on page 265.

List( )

Retrieves a list of service records from the database.

Syntax `EcxService& List(void);`

Returns A reference to this service object.

Example See "Listing All Services" on page 266.

MaxThread ( )

Determines or specifies the maximum number of threads the service can have.

Syntax `unsigned int MaxThread() const;`

`void MaxThread(const unsigned int);`

Returns The first form of the method returns an unsigned integer that contains the maximum number of threads.

Discussion Use the first form of the method to determine the maximum number of threads. Use the second form to specify the maximum number of threads.

Example See "Adding a Service" on page 265.

ModByGroup( )

Determines the group that last modified the record.

Syntax `const char* ModByGroup() const;`

Returns A pointer to a character string that contains the group.

ModByUser( )

Determines the user that last modified the record.

Syntax `const char* ModByUser() const;`

Returns A pointer to a character string that contains the user name.

ModDt( )

Determines the date the record was last modified.

Syntax `const char* ModDt() const;`

Returns A pointer to a character string that contains the date.

More ( )

Determines whether more records are left in the list.

Syntax `long More(void);`

Returns A long integer that contains the number of records not yet accessed from the list.

Discussion After calling the `List()` method and before calling the `Next()` method, the `More()` method returns the total number of records in the list. All records have been accessed when the `More()` method returns 0.

Example See "Listing All Services" on page 266.

See also The `List()` method on page 271. The `Next()` method on page 271.

Name( )

Determines or specifies the name of the service.

Syntax `const char* Name() const;`

```
void Name(const char* name);
```

Parameters The `Name()` method has the following parameters:

`name`	A pointer to a character string that contains the service's name.

Returns The first form of the method returns a pointer to a character string that contains the name.

Discussion Use the first form of the method to determine the service's name. Use the second form to specify the name.

Example See "Adding a Service" on page 265.

Next( )

Associates the object with the next record in the list.

Syntax `EcxService& Next(void);`

Returns A reference to this member object.

Discussion The `Next()` method sets the fields in the object to match those in the next record in the list. The `Next()` method decrements the number of records not yet accessed, which is returned by the `More()` method.

CAUTION Do not call the `Next()` method if the `More()` method returns a value less than 1; the results are unpredictable.

Example See "Listing All Services" on page 266.

See also The `More()` method on page 272.

ObjPerm( )

Determines or specifies the record's access permissions.

Syntax

```
unsigned int ObjPerm() const;
void ObjPerm(const unsigned int permissions);
```

Parameters The `ObjPerm()` method has the following parameters:

`permissions`	An unsigned integer that specifies the access permissions.

Returns The first form of the method returns an unsigned integer that contains the permissions.

Discussion Use the first form of the method to determine the record's access permissions. Use the second form to specify the permissions. The `ObjPerm()` method does not modify the database.

Example See "Adding a Service" on page 265.

Param ( )

Determines or specifies the service description.

Syntax

```
const char * Param() const;
void Param(const char*);
```

Returns The first form of the method returns a pointer to a character string that contains the service description.

Discussion Use the first form of the method to determine the service description. Use the second form to specify the service description.

PathName ( )

Determines or specifies the path name to the service code file.

Syntax

```
const char * Name() const;
void PathName(const char*);
```

Returns The first form of the method returns a pointer to a character string that contains the path name.

Discussion Use the first form of the method to determine the path name to the service code file. Use the second form to specify the path name to the service code file.

SetLogin( )

Allows the object to access the database.

Syntax

```
EcxService& SetLogin(EcxLogin& login);
```

Parameters The `SetLogin()` method has the following parameters:

`login`	A reference to a valid `EcxLogin` object

Returns A reference to this service object.

Discussion If you do not use the form of the constructor that accepts a login object, you must call the `SetLogin()` method before using this object.

Example See "Creating a Service Object" on page 265.

See also The `EcxService` constructor on page 268. The `EcxLogin` class on page 127.

Type( )

Determines or specifies the type of service.

Syntax

```
unsigned int Type() const;
void Type(const unsigned int type);
```

Parameters The `Type()` method has the following parameters:

`type`	An unsigned integer that specifies whether the member is an administrator.

Returns The first form of the method returns an unsigned integer that contains the type.

Discussion You can use any of the following values:

Constant	Value	Description
`STunknown`	0	unknown
`STinternal`	1	internal service (e.g. parse, translate)
`STscript`	2	external script file
`STexe`	3	external executable file
`STdll`	4	a function in a shared library (i.e. DLL or .so)

Example See "Adding a Service" on page 265.

See also "Class Variables" on page 268.

~EcxService(void)

Chapter 18

The EcxServiceList Class

This chapter describes the `EcxServiceList` class, which represents service list records in an iPlanet ECXpert database. This chapter contains the following sections:

About the EcxServiceList Class

The `EcxServiceList` class defines methods you can use to

Methods Summary list:

Constructor and destructor	
`EcxServiceList(void)`	Creates an `EcxServiceList` object.
`virtual ~EcxServiceList(void)`	Destroys an `EcxServiceList` object.
Allowing database access	
`SetLogin`	Allows the object to access the database.
Adding, retrieving, changing and deleting service list records	
`Add`	Adds a service list record to the database.
`Change`	Changes a service list record in the database.
`Delete`	Deletes a service list from the database.
`Get`	Retrieves a service list record from the database.

Listing service list records	
`List`	Retrieves a list of service list records from the database
`More`	Determines whether more records are left in the list.
`Next`	Associates the object with the next record in the list.
Resetting an object's state	
`Clear`	Clears the state associated with an object, including its list.
Accessing key fields	
`ServiceListName`	Determines or specifies the service list name
`SeqNum`	Determines or specifies the sequence number of the service in the service list.
Accessing other fields	
`SndrMBName`	Determines or specifies the sending member name.
`RcvrMBName`	Determines or specifies the receiving member name.
`TypeName`	Determines or specifies the service file type name OR service data object type name.
`SVRId`	Determines or specifies the service ID.
`SVRName`	Determines or specifies the service name.
`ServiceParams`	Determines or specifies the service parameters.
`ErrorHandler`	Determines the name of user-specified service for error handler.
`Desc`	Determines or specifies the service description.
`ObjPerm`	Determines or specifies the record's access permissions.
`ModByGroup`	Determines the group that last modified the record.
`ModByUser`	Determines the user that last modified the record.
`ModDt`	Determines the date the record was last modified.

Using the EcxServiceList Class

The following sections show how to:

- Create a service list object
- Add a service list
- List all service lists
- Modify a service list
- Delete a service list

Creating a Service List Object

The following example shows how to create a ServiceList object.

```
ECXServiceList * pServiceList = NULL;

if((pServiceList = new EcxServiceList())->Errnum())  {
  cout << “EcxServiceListObjectError:” << endl;
  cout << “\tErrnum: “ << pServiceList->Errnum() << endl;
  cout << “\tErrmsg: “ << pServiceList->Errmsg() << endl;
  return(NULL);

}

if((pServiceList->SetLogin(*pLogin)).Errnum())  {
  cout << “EcxServiceList.SetLogin() Failed:” << endl;
  cout << “\tErrnum: “ << pServiceList->Errnum() << endl;
  cout << “\tErrmsg: “ << pServiceList->Errmsg() << endl;
  cout << endl;
  delete pServiceList;
  return(NULL);

return (pServiceList);
```

Adding a Service List

The following example shows how to add a service list.

```
pServiceList->Clear();

pService->ServiceListName(“slname”);
pService->SeqNum(seqNum);
pService->SndrMBName(“ectest1”);
pService->RcvrMBName(“ectest2”);
pService->TypeName(“Test Type”);
pService->SVRId(201);
pService->SVRName(Parse);
pService->ServiceParams(“Test Service Params”);
pService->ErrorHandler(“Test Error Handler”);
pService->Desc(“Test Desc”);
pService->ObjPerm(755);

if((pService->Add()).Errnum())  {
  cout << “EcxServiceList.add() Failed” << endl;
  cout << “\tErrnum: “ << pServiceList->Errnum() << endl;
  cout << “\tErrmsg: “ << pServiceList->Errmsg() << endl;
  return(NULL);
}

id = pService->Id();

cout << “*** Added serviceList: “ << slname << “, “ << seqNum << endl;

return(0);
```

Listing All Service Lists

The following example shows how to generate a list of all service lists.

```
pServiceList->Clear();

If((pServiceList->List()).Errnum())  [
  cout << “EcxServiceList.List() Failed:” << endl;
  cout << “\tErrnum: “ << pServiceList->Errnum() << endl;
  cout << “\tErrmsg: “ << pServiceList->Errmsg() << endl;
  return (pServiceList->Errnum());
}

cout << “*** Listing serviceLists” << pServiceList->More();
cout << “ records found. ***” << endl;

while (pServiceList->More())  {
```

```
  cout << pServiceList->ServiceListName()         << ":";
  cout << pServiceList->SeqName()                 << ":";
  cout << pServiceList->SndrMBName()              << ":";
  cout << pServiceList->RcvrMBName()              << ":";
  cout << pServiceList->TypeName()                << ":";
  cout << pServiceList->SVRId()                   << ":";
  cout << pServiceList->SVRName()                 << ":";
  cout << pServiceList->ServiceParams()           << ":";
  cout << pServiceList->ErrorHandler()            << ":";
  cout << pServiceList->Desc()                    << ":";
  cout << pServiceList->ObjPerm()                 << ":";
  cout << pServiceList->ModByGroup()              << ":";
  cout << pServiceList->ModByUser()               << ":";
  cout << pServiceList->ModDt()                   << ":";
  pService->Next(;
}

return(0);
```

Modifying a Service List

The following example shows how to modify a service list.

```
pServiceList->Clear();
pServiceList->ServiceListName(slname);
pServiceList->SeqNum(seqNum);

if((pServiceList->Get()).Errnum())  {
  cout << "EcxServiceList.Get() Failed" << endl;
  cout << "\tErrnum: " << pServiceList->Errnum() << endl;
  cout << "\tErrmsg: " << pServiceList->Errmsg() << endl;
  return(pServiceList->Errnum());
}

pServiceList->TypeName("Changed Type");

if((pServiceList->Change()).Errnum())  {
  cout << "EcxServiceList.Change() Failed:" << endl;
  cout << "\tErrnum: " << pServiceList->Errnum() << endl;
  cout << "\tErrmsg: " << pServiceList->Errmsg() << endl;
  return(pServiceList->Errnum());
}

cout << "*** Changed serviceList: " << slname << ", " << seqNum <<
endl;

return(0);
```

Deleting a Service List

The following example shows how to delete a list of all service lists.

```
pServiceList->Clear();
pServiceList->Id(id);
pServiceList->SeqNum(seqNum)

if((pServiceList->Delete()).Errnum())  {
  cout << “EcxServiceList.Delete() Failed” << endl;
  cout << “\tErrnum: “ << pServiceList->Errnum() << endl;
  cout << “\tErrmsg: “ << pServiceList->Errmsg() << endl;
  return(pServiceList->Errnum());
}

cout << “*** Deleted serviceList: “ << slname << “, “ << seqNum << endl;

return(0);
```

EcxServiceList Class Reference

Interface `ecxservice.h`

Superclasses `EcxBase`

Subclasses None

Friend Classes None

Syntax `class EcxFtpClient : public EcxBase { ... };`

Class Variables

The following class variables allow you to identify the member as an administrator:

Syntax `static int ADMINISTRATOR;`

`ADMINISTRATOR`	Administrator

Constructor and Destructor

EcxServiceList(void)

Creates an `EcxFtpClient` object.

Syntax `EcxServiceList(void);`
`EcxServiceList(EcxLogin&login)`

Example See "Using the EcxServiceList Class" on page 279.

~EcxServiceList(void)

Destroys an `EcxFtpClient` object.

Syntax `virtual ~EcxServiceList(void);`

Example See "Using the EcxServiceList Class" on page 279.

Methods

This section describes the methods of the `EcxFtpClient` class.

Add ( )

Adds a service list record to the database.

Syntax `EcxServiceList& Add(void);`

Returns A reference to this service list object.

Discussion You must be an administrator and be logged in before calling this method. You must specify the service list name in the object, by calling the `ServiceListName()` method, and specify the sequence number of the service in the service list, by calling the `SeqNum()` method, before calling this method.

Example See "Adding a Service List" on page 280.

See also The `ServiceListName()` method on page 288. The `SeqNum()` method on page 288.

Change( )

Changes a service list record in the database.

Syntax `EcxServiceList& Change(void);`

Returns A reference to this service list object.

Discussion This method updates the last record retrieved by calling the object's `Get()`, `List()`, or `Next()` method. You must be an administrator and be logged in before calling this method. You must specify the service list name in the object, by calling the `ServiceListName()` method, and specify the sequence number of the service in the service list, by calling the `SeqNum()` method, before calling this method.

Warning If you do not call the object's `Get()`, `List()`, or `Next()` method first, the object's name and sequence number fields, which are set by calling the `ServiceListName()` method and the `SeqNum()` method, specify the record that is changed. In this case, the record is completely overwritten using the object's fields. Any fields not set in the object will be replaced by 0 or NULL in the database.

Example See "Modifying a Service List" on page 281.

See also The `Get()` method on page 285. The `List()` method on page 286. The `Next()` method on page 287. The `ServiceListName()` method on page 288. The `SeqNum()` method on page 288.

Clear( )

Clears the state associated with an object, including its list.

Syntax `void Clear(void);`

Example See "Listing All Service Lists" on page 280.

Delete( )

Deletes a service list from the database.

Syntax `EcxServiceList& Delete(void);`

Returns A reference to this service list object.

Discussion You must be an administrator and be logged in before calling this method. You must specify the service list name in the object, by calling the `ServiceListName()` method, and specify the sequence number of the service in the service list, by calling the `SeqNum()` method, before calling this method.

Example See "Deleting a Service List" on page 282.

See also The `ServiceListName()` method on page 288. The `SeqNum()` method on page 288.

Desc ( )

Determines or specifies the service description.

Syntax `const char * Desc() const;`

`void Desc(const char*);`

Returns The first form of the method returns a pointer to a character string that contains the service list description.

Discussion Use the first form of the method to determine the service list description. Use the second form to specify the service list description.

Example See "Adding a Service List" on page 280.

ErrorHandler ( )

Determines the name of user-specified service for error handler.

Syntax `const char * ErrorHandler() const;`

`void ErrorHandler (const char* );`

Returns The first form of the method returns a pointer to a character string that contains the name of user-specified service for error handler.

Discussion Use the first form of the method to determine the name of user-specified service for error handler. Use the second form to specify the name of user-specified service for error handler.

Example See "Adding a Service List" on page 280.

Get( )

Retrieves a service list record from the database.

Syntax `EcxServiceList& Get(void);`

Returns A reference to this service list object.

Discussion You must specify the service list name in the object, by calling the `ServiceListName()` method, and specify the sequence number of the service in the service list, by calling the `SeqNum()` method, before calling this method.

Example See "Modifying a Service List" on page 281.

See also The `ServiceListName()` method on page 288. The `SeqNum()` method on page 288.

List( )

Retrieves a list of service list records from the database.

Syntax `EcxServiceList& List(void);`

Returns A reference to this service list object.

Discussion If you specify the service list's name in the object by calling the `ServiceListName()` method first, only the record matching with the specified name will be retrieved. After calling the `List()` method, the member object contains fields from the first record from the list.

Example See "Listing All Service Lists" on page 280.

See also The `ServiceListName()` method on page 288.

ModByGroup( )

Determines the group that last modified the record.

Syntax `const char* ModByGroup() const;`

Returns A pointer to a character string that contains the group.

ModByUser( )

Determines the user that last modified the record.

Syntax `const char* ModByUser() const;`

Returns A pointer to a character string that contains the user name.

ModDt( )

Determines the date the record was last modified.

Syntax `const char* ModDt() const;`

Returns A pointer to a character string that contains the date.

More ( )

Determines whether more records are left in the list.

Syntax `long More(void);`

Returns A long integer that contains the number of records not yet accessed from the list.

Discussion After calling the `List()` method and before calling the `Next()` method, the `More()` method returns the total number of records in the list. All records have been accessed when the `More()` method returns 0.

Example See "Listing All Service Lists" on page 280.

See also The `List()` method on page 286. The `Next()` method on page 287.

Next( )

Associates the object with the next record in the list.

Syntax `EcxServiceList& Next(void);`

Returns A reference to this member object.

Discussion The `Next()` method sets the fields in the object to match those in the next record in the list. The `Next()` method decrements the number of records not yet accessed, which is returned by the `More()` method.

CAUTION Do not call the `Next()` method if the `More()` method returns a value less than 1; the results are unpredictable.

Example See "Listing All Service Lists" on page 280.

See also The `More()` method on page 286.

ObjPerm( )

Determines or specifies the record's access permissions.

Syntax `unsigned int ObjPerm() const;`

`void ObjPerm(const unsigned int permissions);`

Parameters The `ObjPerm()` method has the following parameters:

`permissions`	An unsigned integer that specifies the access permissions.

Returns The first form of the method returns an unsigned integer that contains the permissions.

Discussion Use the first form of the method to determine the record's access permissions. Use the second form to specify the permissions. The `ObjPerm()` method does not modify the database.

Example See "Adding a Service List" on page 280.

RcvrMBName ( )

Determines or specifies the receiving member name.

Syntax

```
const char * RcvrMBName() const;
void RcvrMBName (const char* );
```

Returns The first form of the method returns a pointer to a character string that contains the receiving member name.

Discussion Use the first form of the method to determine the receiving member name. Use the second form to specify the receiving member name. Because it is the foreign key, the receiving member name must exist in the database.

Example See "Listing All Service Lists" on page 280.

SeqNum

Determines or specifies the sequence number of the service in the service list.

Syntax

```
unsigned int SeqNum() const;
void SeqNum (const unsigned int);
```

Returns The first form of the method returns an unsigned integer that contains the sequence number of the service in the service list.

Discussion Before you call the `Add()`, `Change()`, `Delete()`, or `Get()` methods, you must first specify the service's sequence number within the service list in the object by calling the `SeqNum()` method. Use the first form of the method to determine the sequence number of the service in the service list. Use the second form to specify the sequence number of the service in the service list.

Example The `Add()` method on page 283. The `Change()` method on page 283. The `Delete()` method on page 284. The `Get()` method on page 285. The `ServiceListName()` method on page 288.

ServiceListName ( )

Determines or specifies the service list name.

Syntax

```
const char * ServiceListName() const;
void ServiceListName (const char* );
```

Returns The first form of the method returns a pointer to a character string that contains the service list name.

Discussion Before you call the `Add()`, `Change()`, `Delete()`, or `Get()` methods, you must first specify the service list name in the object by calling the `ServiceListName()` method. Use the first form of the method to determine the service list name. Use the second form to specify the service list name.

Example The `Add()` method on page 283. The `Change()` method on page 283. The `Delete()` method on page 284. The `Get()` method on page 285. The `SeqNum()` method on page 288.

ServiceParams ( )

Determines or specifies the service parameters.

Syntax

```
const char * ServiceParams() const;
void ServiceParams (const char* );
```

Returns The first form of the method returns a pointer to a character string that contains the service parameters.

Discussion Use the first form of the method to determine the service list name. Use the second form to specify the service parameters.

Example See "Listing All Service Lists" on page 280.

SetLogin( )

Allows the object to access the database.

Syntax

```
EcxServiceList& SetLogin(EcxLogin& login);
```

Parameters The `SetLogin()` method has the following parameters:

`login`	A reference to a valid `EcxLogin` object

Returns A reference to this service list object.

Discussion If you do not use the form of the constructor that accepts a login object, you must call the `SetLogin()` method before using this object.

Example See "Creating a Service List Object" on page 279.

See also The `EcxServiceList` constructor on page 283. The `EcxLogin` class on page 127.

SndrMBName ( )

Determines or specifies the sending member name.

Syntax

```
const char * SndrMBName() const;
void SndrMBName (const char* );
```

Returns The first form of the method returns a pointer to a character string that contains the sending member name.

Discussion Use the first form of the method to determine the sending member name. Use the second form to specify the sending member name. Because it is the foreign key, the sending member name must exist in the database.

Example See "Listing All Service Lists" on page 280.

SVRId ( )

Determines or specifies the service ID.

Syntax

```
unsigned int SVRId() const;
void SvrId (const unsigned int);
```

Returns The first form of the method returns an unsigned integer that contains the service ID.

Discussion Use the first form of the method to determine the sequence number of the service in the service list. Use the second form to specify the service ID.

Example See "Listing All Service Lists" on page 280.

SVRName ( )

Determines or specifies the service name.

Syntax

```
const char * SVRName() const;
void SVRName(const char* );
```

Returns The first form of the method returns a pointer to a character string that contains the service name.

Discussion Use the first form of the method to determine the service name. Use the second form to specify the service name.

Example See "Listing All Service Lists" on page 280.

TypeName ( )

Determines or specifies the service file type name OR service data object type name.

Syntax `const char * TypeName() const;`

`void TypeName (const char* );`

Returns The first form of the method returns a pointer to a character string that contains the service ID.

Discussion Use the first form of the method to determine the service ID. Use the second form to specify the service ID.

Example See "Listing All Service Lists" on page 280.

~EcxServiceList(void)

Chapter 19

The ECXpert Java API Classes

This chapter describes the following:

- methods for each Java API class
- environment variables needed by the Java API, and
- API implementation examples

The following ECXpert Java API classes are included:

- JEcxBase
- JEcxInit
- JEcxSubmit
- JEcxLogin
- JEcxMember
- JEcxAddresses
- JEcxPartnership
- JEcxPartnerId
- JEcxDocument
- JEcxDocumentId
- JEcxTracking
- JEcxLog
- JEcxFTPClient
- JEcXService
- JEcxServiceList

Overview

The ECXpert Java API class methods described also provide the command syntax, parameter descriptions, and return values. For detailed information on a particular method, please consult the corresponding method documentation in the C++ class of the ECXpert SDK earlier in this guide.

In cases where the method references are documented elsewhere in this guide, a section and page reference are provided for more details..

NOTE An asterisk (*) after a method indicates that a note for that method is included at the end of the method list.

Directory Structure and Source Files

The ECXpert Java API directory structure branches off of the root directory for the ECXpert SDK: *$NSBASE/NS-apps*/`ECXpert/ecxsdk`. The location of the source files and their descriptions are listed in Table 19-1.

Table 19-1 Location and Description of Java API Source Files

Subdirectory or File	Description of Contents
$NSBASE/NS-apps/ECXpert/ecxsdk/jni	The ECXpert Java API root directory.
$NSBASE/NS-apps/ECXpert/ecxsdk/jniecxsdkjni.jar	The Jar file which contains ECXpert Java API class files and all the example class files.
${NSBASE}/NS-apps/ECXpert/ecxsdkexamples	The ECXpert Java API example source code directory
${NSBASE}/NS-apps/ECXpert/ecxsdkexamples/db	The example source code directory containing the ECXpert Java API database access portion.

Table 19-1 Location and Description of Java API Source Files *(Continued)*

Subdirectory or File	Description of Contents
${NSBASE}/NS-apps/ECXpert/ecxsdkexamples/ftp	The example source code directory containing the ECXpert Java API ftp portion.
${NSBASE}/NS-apps/ECXpert/ecxsdkexamples/submit	The example source code directory containing ECXpert Java API submit portion

JEcxBase

Syntax `class JEcxBase extends java.lang.Object`

The JEcxBase class is the base class from which all ECXpert API classes are derived. For example, ECXpert's JEcxMember class is derived from the JEcxBase class. The JEcxBase class is an abstract class, therefore you should never need to instantiate a JEcxBase object. The JEcxBase class defines methods that are common to the ECXpert API classes you use to interact programmatically with the ECXpert System. The class provides methods that allow you to get and clear the error number and error message corresponding to the last error reported by ECXpert. For example, you can create an object from a class derived from JEcxBase and call that object's `getErrnum()` method to determine whether or not an error occurred. You can call the object's `clearErr()` method to reset the error condition to the "no error" state. Keeping in mind that an Exception is thrown whenever an error occurs in an API class, the best place to call these methods would be within the 'catch' or 'finally' blocks.

Methods Summary list:

Error handling	
`getErrnum()`	Retrieves or sets the last error.
`clearErr()`	Clears the last error that occurred.
`getErrmsg`	Returns error message string (java.lang.string).

For additional details about each method in the JEcxBase Class, refer to Chapter 6, "The EcxBase Class."

JEcxInit

Syntax `class JEcxInit extends java.lang.Object`

The JEcxInit class initializes a connection to the ECXpert database for your application. When using any of the Java API database classes, the user must instantiate a JEcxInit object and call it's `ecxInit()` method.

Method Summary:

Constructor and Destructor	
`ecxInit()`	Creates an `EcxInit` object.
`~ecxInit()`	Destroys an `EcxInit` object.

Example

```
import jni.base.*;
import jni.db.*;

public class myApp {

public static void main (String args[]) {
    try {
      JEcxInit jInit = new JEcxInit();
      jInit.ecxInit();

      JEcxLogin jLogin = new JEcxLogin();
      jLogin.login(args[0], args[1]);
      jLogin.logout();
      System.out.println("Successful login for user " + args[0]);
    }
    catch (Exception e) {
      System.out.println("this is a bad place to be =( ");
      System.err.println(e.getMessage());
    }
  }
}
```

For additional details about the JEcxInit class, refer to Chapter 7, "The EcxInit Class."

JEcxSubmit

Syntax `public class JEcxSubmit extends jni.base.JEcxBase`

The JEcxSubmit class defines methods that you use to submit a file to ECXpert. You can use these methods to provide a file submission capability within your application instead of requiring the user to execute a command or use ECXpert's HTML interface to submit an object.

You may create objects from the `JEcxSubmit` class and use them directly, or you may define a subclass of the `JEcxSubmit` class and create objects from the derived class. For example, you might define a subclass that handles much of the application logic associated with files to be submitted to ECXpert. Objects derived from your subclass would inherit the ability to submit files to ECXpert.

You call methods to specify this information. For example, you call the object's `setSender()` method to specify the sender's member ID. You must specify the files that you wish to submit to ECXpert. You build a submission list by calling the object's `addFile()` method to add a file to the list. You specify the following information when you add a file:

- Document name
- Document type, such as EDI or XML

You can add as many files as you want. If you add more than one file, the files become part of a single multi-part file. When you finish adding the files to the submission list, you can call the object's `submit()` method to submit the files.

By default, ECXpert moves the files being submitted to the directory specified by the repository entry in the configuration file's tcpip-connector section. Moving a file (copying the file and deleting the source file after copying) is the most efficient way to submit files if your application executes on the same server as ECXpert.

You can also submit files to ECXpert using a TCP/IP connection. You specify whether or not to use a TCP/IP connection when you call the object's `submit()` method. Using a TCP/IP connection causes ECXpert to stream the contents of the files through a socket to the server. This is a useful technique if your application

runs on a remote computer and the files being submitted are relatively small. If you want to submit large files from a remote computer, you should consider using a protocol such as FTP to copy the files to the server and then submit them from the server.

NOTE If you stream data through a TCP/IP connection, the source file is not deleted after the data has been streamed to the server.

After you submit a file, you should check for errors. If no error occurred, you can call the object's `getFirstTrackingID()` method to determine the tracking ID of the first file submitted and the object's `getNextTrackingID()` method to determine the tracking ID for each additional file in the list.

Methods Summary list:

Retrieving submission information	
`getDeliveryMethod`	Gets the delivery method.
`getEcxIniFileName`	Gets the full pathname of ECXpert's configuration file.
`getMapName`	Gets the map name.
`getPassword`	Gets the sender's password
`getSender`	Gets the sender's member ID.
Setting submission information	
`setSender()`	Sets the sender's member ID.
`setRecipient()`	Sets the recipient's member ID.
`setPassword()`	Sets the sender's password.
`setEcxIniFileName()`	Sets the full pathname of ECXpert's configuration file.
`setMapName()`	Sets the map name.
`setDeliveryMethod()`	Sets the delivery method.

Manipulating the submission list	
`addFile()`	Adds a file to the submission list.
`clearFileList()`	Clears the submission list.
`getFirstTrackingID()`	Retrieves the tracking ID for the first file in the object's submission list.
`getNextTrackingID()`	Retrieves the tracking ID for the next file in the object's submission list.
Submitting files	
`submit()`	Submits objects to ECXpert for processing.

For additional details about each method in the JEcxSubmit class, refer to Chapter 8, "The EcxSubmit Class."

JEcxLogin

Syntax `public class JEcxLogin extends jni.base.JEcxBase`

Objects of the JEcxLogin class represent connections to ECXpert. To log in to the database, you can create a JEcxLogin object and call the object's login method. When you no longer need the connection to ECXpert, you should call the object's logout method.

Method Summary:

Login/Logout	
`login()`	.Logs in to the database.
`logout()`	Logs out of the database.
Member Determination	
`memberType()`	Determines the type of member currently logged in to ECXpert (e.g., an administrator).

Example

```
public void loginToEcx(String user, String password) throws Exception
{
  JEcxLogin ecxLogin = null;
  boolean success = true;

  try
  {
    ecxLogin = new JEcxLogin();
    ecxLogin.login (user, password);
    System.out.println(ecxLogin.memberType());
  }
  catch (Exception e)
  {
      success = false;
      System.err.println(e.getMessage());
      System.err.println("Error number " + ecxLogin.getErrnum());
      System.err.println("Error message " + ecxLogin.getErrmsg());
  }
  finally {
      try { if (success) ecxLogin.logout(); }
      catch (Exception exp) {
        System.out.println("Warning: " + exp.getMessage());
      }
  }
}
```

For additional details about each method in the JEcxLogin class, refer to Chapter 9, "The EcxLogin Class."

JEcxMember

Syntax `public class JEcxMember extends jni.base.JEcxBase`

The JEcxMember class represents member records in an ECXpert database. Administrators can manipulate any member record for their trading partnerships; non-administrators can only change contact information in their own record. A user must be logged-in to the database before accessing a record.

Method Summary

Allowing database access	
`setLogin()`	Allows the object to access the database.
Adding, retrieving, changing and deleting member records	
`add()`	Adds a member record to the database.
`get()`	Retrieves a member record from the database.
`change()`	Changes a member record in the database.
`delete()`	Deletes a member from the database.
Listing member records	
`list()`	Retrieves a list of member records from the database.
`more()`	Determines whether more records are left in the list.
`next()`	Associates the object with the next record in the list.
Resetting an object's state	
`clear()`	Clears the state associated with an object, including its list.
Accessing key fields	
`getName()`	Obtains the name of the member.
Accessing contact information	
`getContactName()`	Obtains the name of the contact person for this member.
`getContactCompany()`	Obtains the contact's company.
`getContactAddress1()`	Obtains the first line of the contact's address.
`getContactAddress2()`	Obtains the second line of the contact's address.
`getContactCity()`	Obtains the contact's city.
`getContactState()`	Obtains the contact's state.
`getContactZip()`	Obtains the contact's zip or postal code.
`getContactCountry()`	Obtains the contact's country.
`getContactPhone()`	Dbtains the contact's phone number.
`getContactFax()`	Obtains the contact's fax number.
`getContactEmailId()`	Obtains the contact's e-mail address.

Accessing other fields	
`getDescription()`	Gets the member's description.
`getType()`	Gets the type of member.
`getParentName()`	Gets the name of the parent member.
`getIsGroup()`	Obtains whether the member is a group or individual.
`getActive()`	Obtains whether the member is active.
`getPassword()`	Obtains the member's password.
`getTrusted()`	Obtains whether the member is trusted.
`getObjPerm()`	Obtains the record's access permissions.
`getModByGroup()`	Obtains the group that last modified the record.
`getModByUser()`	Obtains the user that last modified the record.
`getModDt()`	Obtains the date the record was last modified.
Setting key fields	
`setName()`	Sets the name of the member.
Setting other fields	
`setDescription()`	Sets the member's description.
`setType()`	Sets the type of member.
`setIsGroup()`	Sets the member as a group or individual.
`setActive()`	Sets the member as active or not active.
`setPassword()`	Sets the member's password.
`setTrusted()`	Sets the member as trusted or not trusted.
`setObjPerm()`	Sets the record's access permissions.
Setting contact information	
`setContactName()`	Sets the name of the contact person for this member.
`setContactCompany()`	Sets the contact's company.
`setContactAddress1()`	Sets the first line of the contact's address.
`setContactAddress2()`	Sets the second line of the contact's address.
`setContactCity()`	Sets the contact's city.
`setContactState()`	Sets the contact's state.
`setContactZip()`	Sets the contact's zip or postal code.

`setContactCountry()`	Sets the contact's country.
`setContactPhone()`	Sets the contact's phone number.
`setContactFax()`	Sets the contact's fax number.
`setContactEmailId()`	Sets the contact's e-mail address.

Example

```
// This example sets the Miscellaneous information about a member.
//The Contact* variables are assumed to be supplied from somewhere else

public void updateContactInfo(JEcxLogin ecxLogin, String userName)
throws Exception
{
  JEcxMember ecxMember = null;
  try
  {
    ecxMember = new JEcxMember();
    ecxMember.setLogin(ecxLogin);
    ecxMember.clear();
    ecxMember.setName(userName);
    ecxMember.get();

    ecxMember.setContactName(contactName);
    ecxMember.setContactAddress1(contactAddress1);
    ecxMember.setContactAddress2(contactAddress2);
    ecxMember.setContactCity(contactCity);
    ecxMember.setContactState(contactState);
    ecxMember.setContactCountry(contactCountry);
    ecxMember.setContactZip(contactZip);
    ecxMember.setContactFax(contactFax);
    ecxMember.setContactPhone(contactPhone);
    ecxMember.setContactEmailId(contactEmail);

  ecxMember.change();
  }
  catch (Exception e)
  {
      System.err.println(e.getMessage());
      System.err.println("Error number " + ecxMember.getErrnum());
      System.err.println("Error message " + ecxMember.getErrmsg());
  }
}
```

For additional details about each method in the JEcxMember class, refer to Chapter 12, "The EcxMember Class."

JEcxAddresses

Syntax `public class JEcxAddresses extends jni.base.JEcxBase`

The JEcxAddresses class represents trading address records in the ECXpert database. Administrators can manipulate any address record; non-administrators can add and delete only their own address records. A user must be logged in to the database before accessing a record.

Methods Summary list:

Allowing database access	
`setLogin()`	Allows the object to access the database.
Adding and deleting address records	
`add()`	Adds an address record to the database.
`delete()`	Deletes an address record from the database.
Listing address records	
`list()`	Retrieves a list of address records from the database.
`more()`	Determines whether more records are left in the list.
`next()`	Associates the object with the next record in the list.
Resetting an object's state	
`clear()`	Clears the state associated with an object, including its list.
Accessing key fields	
`setMember()`	Sets a member.
`setQual()`	Sets a member's trading address qualifier.
`setQualId()`	Sets a member's trading address.

For additional details about each method in the JEcxAddresses class, refer to Chapter 10, "The EcxAddresses Class."

JEcxPartnership

Syntax `public class JEcxPartnership extends jni.base.JEcxBase`

The JEcxPartnership class represents the following kinds of records in the ECXpert database:

- document types
- EDI standards information
- partnership groups
- partnerships

A record in the view represents a partnership record whose ID matches a standards information ID, a group ID and a document type ID and whose group type matches the document type. Only administrators can add, change, or delete records using this view. An administrator can retrieve any record from the view; a non administrator can only retrieve records from the view that includes the user as either a sender or receiver. A user must be logged in to the database before accessing a record through the view.

Methods Summary list:

Allowing database access	
`setLogin()`	Allows the object to access the database.
Adding, retrieving, changing and deleting partnership view-related records	
`add()`	Adds partnership view-related records to the database.
`get()`	Retrieves partnership view-related records from the database.
`change()`	Changes partnership view-related records in the database.
`delete()`	Deletes partnership view-related records from the database.
Listing partnership records	
`list()`	Retrieves a list of partnership view-related records from the database.
`more()`	Determines whether more records are left in the list.

`next()`	Associates the object with the next record in the list.
Resetting an object's state	
`clear()`	Clears the state associated with an object, including its list.
Accessing key fields	
`getPartnerId()`	Obtains the partnership ID.
`getdocType()`	Obtains the kind of EDI document.
`getGroupType()`	Obtains the kind of EDI documents in the group.
Accessing partnership information	
`getSenderName()`	Obtains the sender's member name.
`getSenderQual()`	Obtains the sender's trading address qualifier.
`getSenderQualId()`	Obtains the sender's trading address.
`getSenderCertificateType()`	Obtains the sender's certificate type.
`getReceiverName()`	Obtains the receiver's member name.
`getReceiverQual()`	Obtains the receiver's trading address qualifier.
`getReceiverQualId()`	Obtains the receiver's trading address
`getReceiverCertificateType()`	Obtains the receiver's certificate type.
`getActive()`	Obtains whether the partnership is active.
`getSecurity()`	Obtains the kind of security.
`getDescription()`	Obtains the partnership's description.
Accessing standards information	
`getStandardName()`	Obtains the name of the EDI standard.
`getStandardVersion()`	Obtains the standard's version number.
`getStandardRelease()`	Obtains the standard's release number.
`getIntchngLastControlNumber()`	Obtains the last interchange control number generated.
`getIntchngLock()`	Obtains whether the document has been read at the interchange level.
`getIntchngGenerateAck()`	Obtains whether to generate interchange acknowledgments flags.

`getIntchngAckWaitPeriod()`	Obtains the number of minutes to wait before the acknowledgment becomes overdue.
`getTestProductionFlag()`	Obtains whether the partnership is used for testing or production.
`getSegmentTerminator()`	Obtains the segment terminator character.
`getElementSeparator()`	Obtains the data element terminator character.
`getSubElementSeparator()`	Obtains the data subelement terminator character.
`getDecimalPointCharacter()`	Obtains the decimal point character.
`getReleaseCharacter()`	Obtains the release character.
`getOutStandard()`	Obtains the interchange standard user wishes to appear in bundled EDI documents.
`getOutVersion()`	Obtains the interchange version user wishes to appear in bundled EDI documents
`getOutRelease()`	Obtains the interchange release user wishes to appear in bundled EDI documents.
`getGenOptEnv()`	Obtains the enveloping options.
Accessing group information	
`getGroupLastControlNumber()`	Obtains the last group control number generated.
`getGroupLock()`	Obtains whether the document has been read at the group level.
`getGroupGenerateDocAck()`	Obtains the to generate group acknowledgments flags
`getSndrAppQual()`	Obtains the sending member main trading address.
`getSndrAppCode()`	Obtains the application sender code.
`getRcvrAppQual()`	Obtains the receiving member main trading address.
`getRcvrAppCode()`	Obtains the application receiver code.
Accessing document type specific information	
`getDocPriority()`	Obtains the document processing priority.
`getMapName()`	Obtains the map file name.
`getMapDirection()`	Obtains the document translation type.

`getAckExpected()`	Obtains the number of minutes to wait before an acknowledgment becomes overdue.
`getDocLastControlNumber()`	Obtains the last document control number generated.
`getDocLock()`	Obtains the read status of the document.
`getPrimaryXportType()`	Obtains the primary transport protocol.
`getPrimaryXportParam()`	Obtains the primary transport protocol parameter.
`getSecondaryXportType()`	Obtains the secondary transport protocol.
`getSecondaryXportParam()`	Obtains the secondary transport protocol parameter.
`getSendType()`	Obtains when the document is to be sent.
`getDeleteWaitPeriod()`	Obtains the number of days to retain documents before deleting them.
`getArchiveWaitPeriod()`	Obtains the number of days to retain documents before archiving them.
`getPreEnveloped()`	Obtains whether documents are preenveloped.
Setting partnership information	
`setenderName()`	Sets the sender's member name.
`setSenderQual()`	Sets the sender's trading address qualifier.
`setSenderQualId()`	Sets the sender's trading address.
`setSenderCertificateType()`	Sets the sender's certificate type.
`setReceiverName()`	Sets the receiver's member name.
`setReceiverQual()`	Sets the receiver's trading address qualifier.
`setReceiverQualId()`	Sets the receiver's trading address
`setReceiverCertificateType()`	Sets the receiver's certificate type.
`setActive()`	Sets whether the partnership is active.
`setSecurity()`	Sets the kind of security.
`setDescription()`	Sets the partnership's description.
Setting standards information	
`setStandardName()`	Sets the name of the EDI standard.
`setStandardVersion()`	Sets the standard's version number.

`setStandardRelease()`	Sets the standard's release number.
`setIntchngLastControlNumber()`	Sets the last interchange control number generated.
`setIntchngLock()`	Sets the document has been read at the interchange level.
`setIntchngGenerateAck()`	Sets generate interchange acknowledgments flags.
`setIntchngAckWaitPeriod()`	Sets the number of minutes to wait before the acknowledgment becomes overdue.
`setTestProductionFlag()`	Sets the partnership use for testing or production.
`setSegmentTerminator()`	Sets the segment terminator character.
`setElementSeparator()`	Sets the data element terminator character.
`setSubElementSeparator()`	Sets the data subelement terminator character.
`setDecimalPointCharacter()`	Sets the decimal point character.
`setReleaseCharacter()`	Sets the release character.
`setOutStandard()`	Sets the interchange standard user wishes to appear in bundled EDI documents.
`setOutVersion()`	Sets the interchange version user wishes to appear in bundled EDI documents
`setOutRelease()`	Sets the interchange release user wishes to appear in bundled EDI documents.
`setGenOptEnv()`	Sets the enveloping options.
Setting group information	
`setGroupLastControlNumber()`	Sets the last group control number generated.
`setGroupLock()`	Sets the read status of the document at the group level.
`setGroupGenerateDocAck()`	Sets the to generate group acknowledgments flags
`setSndrAppQual()`	Sets the sending member main trading address.
`setSndrAppCode()`	Sets the application sender code.
`setRcvrAppQual()`	Sets the receiving member main trading address.
`setRcvrAppCode()`	Sets the application receiver code.

Setting document type specific information	
`setDocPriority()`	Sets the document processing priority.
`setMapName()`	Sets the map file name.
`setMapDirection()`	Sets the document translation type.
`setAckExpected()`	Sets the number of minutes to wait before an acknowledgment becomes overdue.
`setDocLastControlNumber()`	Sets the last document control number generated.
`setDocLock()`	Sets the read status of the document.
`setPrimaryXportType()`	Sets the primary transport protocol.
`setPrimaryXportParam()`	Sets the primary transport protocol parameter.
`setSecondaryXportType()`	Sets the secondary transport protocol.
`setSecondaryXportParam()`	Sets the secondary transport protocol parameter.
`setSendType()`	Sets the document send type..
`setDeleteWaitPeriod()`	Sets the number of days to retain documents before deleting them.
`setArchiveWaitPeriod()`	Sets the number of days to retain documents before archiving them.
`setPreEnveloped()`	Sets preenveloped condition of documents..
Setting partnership information	
`setenderName()`	Sets the sender's member name.
`setSenderQual()`	Sets the sender's trading address qualifier.

Example

```
public static void whoSendsMeDocuments(JEcxLogin ecxLogin, String userName)
throws Exception
{
      JEcxPartnership ecxPtnship = null;

      try {
      ecxPtnship = new JEcxPartnership();
      ecxPtnship.setLogin(ecxLogin);
      ecxPtnship.clear();
      ecxPtnship.setSenderName(userName);
      ecxPtnship.list();
```

```
      //retrieve partnership information
      while(ecxPtnship.more() > 0)
      {
        System.out.println ("user " + userName + " receives "
          + ecxPtnship.getDocType() + " type documents from "
          + ecxPtnship.getSenderName());

        ecxPtnship.next();
      }
    }
    catch (Exception e)
    {
      System.err.println(e.getMessage());
      System.err.println("Error number " + ecxPtnship.getErrnum());
      System.err.println("Error message "
          + ecxPtnship.getErrmsg());
    }
}
```

For additional details about each method in the JEcxPartnership class, refer to Chapter 11, “Partnership-Related Classes.”

JEcxPartnerId

Syntax `public class JEcxPartnerId extends jni.base.JEcxBase`

The JEcxPartnerId class represents a key from which partnership views can be retrieved from the database. You must create an JEcxPartnerId object before you can call the partnership's `get()` and get `PartnerId()` methods. A partner ID consists of the following values:

- document type
- partnership ID
- standard ID
- group ID

In general, values for a partnership ID and a standard ID are the same for each record in the view.

Methods Summary list:

Setting key values	
`setValues()`	Sets the values associated with a partnership view key.
Obtaining key values	
`getDocType()`	Obtains the document type in the key.
`getPartnershipID()`	Obtains the partnership ID in the key.
`getStandardID()`	Obtains the standard ID in the key.
`getGroupId()`	Obtains the group ID in the key.

For additional details about each method in the JEcxPartnerId class, refer to Chapter 11, "Partnership-Related Classes."

JEcxDocument

Syntax `public class JEcxDocument extends jni.base.JEcxBase`

The JEcxDocument class represents documents sent to the logged-in user via ECXpert. You can retrieve these document records and access information that identifies them, such as the file name that contains the document's content.

Methods Summary list:

Setting database access	
`setLogin()`	Sets the object to access the database.
Retrieving and listing document records	
`get()`	Retrieves a document record from the database.
`list()`	Retrieves a list of document records from the database.
`more()`	Determines whether more records are left in the list.
`next()`	Associates the object with the next record in the list.
`delete`	Deletes document records from the database.

Resetting an object's state	
`clear()`	Clears the state associated with an object, including its list.
Accessing key fields	
`getDocId()`	Obtains the document ID.
Accessing document information	
`getFileName()`	Obtains the name of the file associated with the document.
`getSecondaryTitle()`	Obtains the secondary title.
`getSecondaryValue()`	Obtains the secondary value.
`getSenderName()`	Obtains or specifies the sender's member name.
`getState()`	Obtains the document's state.
`getTitle()`	Obtains the document's title.
`getValue()`	Obtains the document's value.
`getXportParam()`	Obtains the transport parameter.
`getXportType()`	Obtains the transport protocol.
`getFilename()`	Obtains the name of the file associated with the document.
`getCreationDate()`	Obtains the date the document was created.
`getModifyDate()`	Obtains the most recent document modification date.
`getDocType()`	Obtains the document type.
`getStandard()`	Obtains the document's EDI standard.
`getVersion()`	Obtains the document's EDI version.
`getRelease()`	Obtains the document's EDI standard release number.
`getCardCount()`	Obtains the number of cards associated with the document.
`getDataState()`	Obtains the state the document data is in.
`getRead()`	Obtains the read status of the document.
Accessing card-level information	
`getCardIOType()`	Obtains the card input/output type.
`getCardFlags()`	Accesses information about what card flags have been set.

`getTrackState()`	Obtains the document's tracking state.
`getTranslatedFileName()`	Accesses the name of the translated file.
`setReadyForPurge()`	Sets the document to "ready to be purged" state.
Setting document information	
`setExportType()`	Sets the transport protocol.
`setSenderName`	Sets the sender's member name.
`setRead()`	Sets the read status of the document.
Setting card-level information	
`setTrackState`	Sets the document's tracking state.

Example

```
public void listDocuments (JEcxLogin ecxLogin) throws Exception

JEcxDocument jDoc = null;

  try
  {

    System.out.println (jDoc.more() + " documents found");
    while (jDoc.more() > 0) // loop through each document
    {
      System.out.println ("Filename = " + jDoc.getFileName(1));
      jDoc.next();
    }
  }
  catch (Exception e)
  {
    System.err.println(e.getMessage());
    System.err.println("Error number " + jDoc.getErrnum());
    System.err.println("Error message " + jDoc.getErrmsg());
  }
}
```

For additional details about each method in the JEcxDocument class, refer to Chapter 13, "Document-Related Classes."

JEcxDocumentId

Syntax `public class JEcxDocumentId extends jni.base.JEcxBase`

The JEcxDocumentId class represents a key from which documents can be retrieved from the database. You must create a JEcxDocId object before you can call the document's `get()` and `getDocId()` methods.

A document ID key consists of the following values:

- tracking ID
- interchange ID
- group ID
- document ID

Methods Summary list:

Setting key values	
`setValues()`	Sets the values associated with a document's key.
Determining key values	
`getDocumentID()`	Obtains the document ID in the key.
`getTrackingID()`	Obtains the tracking ID in the key.
`getInterchangeID()`	Obtains the interchange ID in the key.
`getGroupID()`	Obtains the group ID in the key.

Example

```
*/
A JEcxDocument object is required before this method can be used.
See the example of the JEcxDocument class.
*/

public void ShowDocId (JEcxDocument docObj) throws Exception
{
JEcxDocId docIDObj = null;
try
  {
    docIDObj = new JEcxDocId();
    docObj.getDocId(docIDObj.getPeerPointer());
```

```
    System.out.println ("Document Id is " +
      docIDObj.getDocumentId());
    System.out.println ("Tracking ID is " +
      docIDObj.getTrackingId());
    System.out.println ("Interchange ID is " +
      docIDObj.getInterchangeId());
    System.out.println ("Group ID is " + docIDObj.getGroupId());
  }
  catch (Exception e)
  {
    System.err.println(e.getMessage());
    System.err.println("Error number " + docIDObj.getErrnum());
    System.err.println("Error message " + docIDObj.getErrmsg());
  }
}
```

For additional details about each method in the JEcxDocId class, refer to Chapter 13, “Document-Related Classes.”

JEcxTracking

Syntax `public class JEcxTracking extends jni.base.JEcxBase`

The JEcxTracking class represents documents sent from the logged-in user via ECXpert. You can retrieve the tracking status of a document using a JEcxTracking object.

Methods Summary list:

Setting database access	
`setLogin()`	Allows the object to access the database.
Listing document records	
`list()`	Retrieves a list of document records from the database.
`more()`	Determines whether more records are left in the list.
`next()`	Associates the object with the next record in the list.
`get()`	Retrieves document ID records from the database.
`delete()`	Deletes a document record.

Resetting an object's state	
`clear()`	Clears the state associated with an object, including its list.
Accessing document information	
`getSecondaryTitle()`	Obtains the secondary title.
`getSecondaryValue()`	Obtains the secondary value.
`getReceiverName()`	Obtains receiver's member name.
`getState()`	Obtains the document's state.
`getTitle()`	Obtains the document's title.
`getValue()`	Obtains the document's value.
`getProgress()`	Obtains the document's progress.
`getFileName()`	Accesses the file name of the document.
`getStandard()`	Obtains the document's EDI standard.
`getVersion()`	Obtains the document's EDI standard version number.
`getRelease()`	Obtains the document's EDI standard release number.
`getTranslatedFileName()`	Accesses the name of the translated file.
`getCreationDate()`	Accesses the date the document was created.
`getModifyDate()`	Accesses the date the document was last modified.
`getDocType()`	Obtains the document type.
`getDataState()`	Obtains the state the data is in.
`setReadyForPurge()`	Sets the status for readiness for the document to be purged.

Example

```
public static void listOutboundDocuments (JEcxLogin ecxLogin, String userName)
                  throws Exception
{
  JEcxTracking jTrack = null;

  try
  {
    jTrack = new JEcxTracking();
    jTrack.setLogin(ecxLogin);
    jTrack.clear();
    jTrack.list ("", 0, 0, 0, userName);
```

```
    System.out.println (jTrack.more() + " documents found");
    while (jTrack.more() > 0) // loop through each document
    {
      System.out.println ("Sent " + jTrack.getDocType() +
        " type document to " +jTrack.getReceiverName());
      jTrack.next();
    }
  }
  catch (Exception e)
  {
    System.err.println(e.getMessage());
      System.err.println("Error number " + jTrack.getErrnum());
      System.err.println("Error message " + jTrack.getErrmsg());
  }
}
```

For additional details about each method in the JEcxTracking class, refer to Chapter 14, "The EcxTracking Class."

JEcxLog

Syntax `public class JEcxLog extends jni.base.JEcxBase`

The JEcxLog class represents entries in the ECXpert log. You can use a JEcxLog object to add an entry to the log or list log entries.

Methods Summary list:

Setting database access	
`setLogin()`	Sets the object to access the database.
Logging an event	
`logEvent()`	Adds an entry to the log.
Resetting an object's state	
`clear()`	Clears the state associated with an object.

Accessing log information	
`next`	Associates the object with the next record in the list.
`more`	Determines whether more records are left in the list.
`retrieveLog`	Retrieves log information.
`getELId`	Obtains the ID number of the event in the event log.
`getELEventId`	Obtains the ID number of the event in the event log.
`getELCategory`	Obtains the category of the event in the event log.
`getELSeverity`	Obtains the severity of the event in the event log.
`getELEventShortMsg`	Obtains the short message associated with the event in the event log.
`getElTrkId`	Obtains the tracking ID of the event in the event log.
`getElIntgId`	Obtains the interchange identifier.
`getELGrpId`	Obtains the group ID of the event in the event log.
`getElDocId`	Obtains the ID number of the document in the event log.
`getElTDId`	Obtains the document-level internal tracking ID associated with the event.

For additional details about each method in the JEcxLog class, refer to Chapter 15, "The EcxLog Class."

JEcxFtpClient

Syntax `public class JEcxFtpClient extends jni.base.JEcxBase`

The JEcxFtpClient class is a FTP Client API which defines methods you can use to send and receive files via FTP. The JEcxFtpClient class is based on the RFC 959 FTP protocol.

Before you will be able to perform any FTP operations, you must first create the `JEcxFtpClient()` object and call the init() method. You can then run FTP commands using the `runCommand()` method.

Methods Summary list:

Initializing the FTP Client API	
`init`	Initializes the FTP client API
Accessing Entry Information	
`getListCount()`	Retrieves the number of files in the current directory listing
`getFirstListEntry()`	Retrieves the first file in the directory listing
`getNextListEntry()`	Retrieves the next file in the directory listing
Accessing FTP Replies	
`getReplyCode()`	Retrieves the last reply code
`getReplyMsg()`	Retrieves the last reply message
`isReplyGood`	Indicates whether the last FTP command executed was successful or not
Running Commands	
`runCommand`	Runs a command

For additional details about each method in the JEcxFtpClient class, refer to Chapter 16, "The EcxFtpClient Class."

JEcxService

Syntax `public class JEcxService extends jni.base.JEcxBase`

The JEcxService class represents service records in an ECXpert database. Only administrators can add, change, or delete a service record. A user must be logged-in to the database before viewing a service record.

Methods Summary list:

Allowing database access	
`setLogin`	Allows the object to access the database.

Adding, retrieving, changing and deleting service records	
`add`	Adds a service record to the database.
`change`	Changes a service record in the database.
`delete`	Deletes a service from the database.
`get`	Retrieves a service record from the database.
Listing service records	
`list`	Retrieves a list of service records from the database
`more`	Determines whether more records are left in the list.
`next`	Associates the object with the next record in the list.
Resetting an object's state	
`clear`	Clears the state associated with an object, including its list
Accessing key fields	
`getId`	Obtains the ID number of the service.
Accessing other fields	
`getName`	Obtains the name of the service.
`getType`	Obtains the service type.
`getPathName`	Obtains the path name to the service code file.
`getEntryName`	Obtains the entry name of the service.
`getMaxThread`	Obtains the maximum number of threads the service can have.
`getParam`	Obtains the service description.
`getObjPerm`	Obtains the record's access permissions.
`getModByGroup`	Obtains the group that last modified the record.
`getModByUser`	Obtains the user that last modified the record.
`getModDt`	Obtains the date the record was last modified.
Setting key fields	
`setID`	Sets the ID number of the service.
Setting other fields	
`setName`	Sets the name of the service.
`setType`	Sets the service type.
`setPathName`	Sets the path name to the service code file.
`setEntryName`	Sets the entry name of the service.

`setMaxThread`	Sets the maximum number of threads the service can have.
`setParam`	Sets the service description.
`setObjPerm`	Sets the record's access permissions.

For additional details about each method in the JEcxService class, refer to Chapter 17, "The EcxService Class."

JEcxServiceList

Syntax `public class JEcxServiceList extends jni.base.JEcxBase`

The JEcxServiceList class represents service list records in the ECXpert database. Only administrators can add, change, or delete a service list record. A user must be logged-in to the database before viewing a service list record.

Methods Summary list:

Allowing database access	
`setLogin`	Allows the object to access the database.
Adding, retrieving, changing and deleting service list records	
`add`	Adds a service list record to the database.
`change`	Changes a service list record in the database.
`delete`	Deletes a service list from the database.
`get`	Retrieves a service list record from the database.
Listing service list records	
`list`	Retrieves a list of service list records from the database
`more`	Determines whether more records are left in the list.
`next`	Associates the object with the next record in the list.
Resetting an object's state	
`clear`	Clears the state associated with an object, including its list.

Accessing key fields	
`getServiceListName`	Obtains the service list name
`getSeqNum`	Obtains the sequence number of the service in the service list.
Accessing other fields	
`getSndrMBName`	Obtains the sending member name.
`getRcvrMBName`	Obtains the receiving member name.
`getTypeName`	Obtains the service file type name OR service data object type name.
`getSVRId`	Obtains the service ID.
`getSVRName`	Obtains the service name.
`getServiceParams`	Obtains the service parameters.
`getErrorHandler`	Obtains the name of user-specified service for error handler.
`getDesc`	Obtains the service description.
`getObjPerm`	Obtains the record's access permissions.
`getModByGroup`	Obtains the group that last modified the record.
`getModByUser`	Obtains the user that last modified the record.
`getModDt`	Obtains the date the record was last modified.
Setting key fields	
`setServiceListName`	Sets the service list name
`setSeqNum`	Sets the sequence number of the service in the service list.
Setting other fields	
`setSndrMBName`	Sets the sending member name.
`setRcvrMBName`	Sets the receiving member name.
`setTypeName`	Sets the service file type name OR service data object type name.
`setSVRId`	Sets the service ID.
`setSVRName`	Sets the service name.
`setServiceParams`	Sets the service parameters.
`setErrorHandler`	Sets the name of user-specified service for error handler.
`setDesc`	Sets the service description.
`setObjPerm`	Sets the record's access permissions.

For additional details about each method in the JEcxServiceList class, refer to Chapter 17, "The EcxService Class."

Setting Up The Environment

Before running the ECXpert Java API examples indicated in the last step below, the following environment variables must be set up. Commands are stated for the C shell operating environment.

1. Set LD_LIBRARY_PATH as follows.

   ```
   setenv LD_LIBRARY_PATH ${NSBASE}/NS-apps/ECXpert/lib
   ```

   Set BDGHOME as follows.

   ```
   setenv BDGHOME ${NSBASE}/NS-apps/ECXpert
   ```

2. If you are using JDK 1.1.6 (or higher), set CLASSPATH as follows.

   ```
   setenv CLASSPATH
   .:/usr/java1.1/lib/classes.zip:${NSBASE}/NS-apps/ECXpert/ecxsdk/
   jni/ecxsdkjni.jar
   ```

3. Verify that the JDK is installed in the `/usr/java1.1` directory.

4. Run the following example:

   ```
   java jni.examples.db.j_membertest
   ```

jexport Utility Implementation Example

The jexport utility is designed to export the contents of the ECXpert database using the Java SDK. The output is formatted for use with the traditional ECX import utility, described in the *iPlanet ECXpert Administrator's Guide*. This utility might be used to:

- back up the contents of your ECX DB
- create a reproducable test or QA environment on several machines
- migrate DB contents from one machine to another
- promote specific memberships and partnerships from test to production

Usage

```
java jexport <admin> <password> [-m <member> | -s]
```

- if the 'm' flag is used, the following information is exported:
  - - member entry for the member specified on the command line
  - - partnership entries for any partnership that the member is part of
  - - member entries for all members that share a partnership
- the 's' flag is used to export all services and service lists. This option is exclusive of all other options.
- in a new ECX installation, the <admin> / <password> will be "ECX" / "ECX".

Known Issues and Limitations

The following issues and limitations apply when using the import utility.

- The field delimiter is currently hard-coded to a comma (',').
- The only import operation supported at this time is 'insert'. However, the output could be modified to perform an update or delete, or other functions.
- Exporting and importing the included ECX services (such as parse, translate, gateway, and so forth) is not recommended since the Service ID cannot be specified.
- The getPassword() method in the JEcxMember class returns encrypted data. As a result, each member's password will be set to (hard-coded as) "PASSWORD".

Chapter 20

Specifying Billing Data to be Captured by the Billing Utility

This chapter describes the guidelines used to capture billing data from the ECXpert database. The following topics are covered:

- Overview
- Billing Data DTD Model
- Billing Data Utility

Overview

In order to capture billing data, a billing code is specified in the Partnership UI for the partnership that is to have a report on billing data for sender, receiver, and document types over some predetermined time interval. Refer to the topic "Partnership Information."

When a billing code is specified, the billing data is captured according to the information contained within the billing data DTD described in the next section.

Once captured, the billing data utility is used at the command line to prepare the ECXpert Billing Data report. This data utility amounts to running a perl script, as described in the Billing Data Utility section following the billing data DTD section.

Billing Data DTD Model

Billing data is captured and presented in XML format by a perl script. In order for the perl script to run correctly, the billing data must be prepared to include the partnership data, range attributes for tracking IDs to be included, and the dates for which the capture begins and ends. This information is specified in the billing data DTD which follows below. The billing data DTD exists in the following directory:

```
<!-- ECXpert 3.5 ECXBillingData DTD
     Copyright (C) 2001 iPlanet.com
                                        -->
  <?xml version="1.0"?>
  <!DOCTYPE ECXBillingData
    <!ELEMENT ECXBillingData (Partners*)>
    <!ATTLIST ECXBillingData
      ReportDate    CDATA          #IMPLIED
      MinTrk        CDATA          #IMPLIED
      MaxTrk        CDATA          #IMPLIED
      MinDate       CDATA          #IMPLIED
      MaxDate       CDATA          #IMPLIED>

<!ELEMENT Partners (Item+)>
<!ATTLIST Partners
      Sndr              CDATA       #REQUIRED
      Rcvr              CDATA       #REQUIRED
      Type              CDATA       #REQUIRED
      BillingCode       CDATA       #REQUIRED>

<!ELEMENT Item/>
<!ATTLIST Item
      Doc               ID           #REQUIRED
      ReceiveDate       CDATA        #REQUIRED
      Size              CDATA        #REQUIRED
      Priority   ( h | m | l | s )    #REQUIRED>
     >
```

This model tells that a ECXBillingData report consists of an ECXBillingData element, which has a required ReportDate and optional range attributes MinTrk, MaxTrk, MinDate, and MaxDate. The contents of BillingData is made up of zero or a number of Partners elements. Each Partners element specifies sender, receiver, document type and billing code, and is followed by a number of (at least one) document Item. The document Item shows ID, ReceiveDate, Size and processing Priority of the document.

Here is an example of an XML file conforms to the above DTD:

```
<ECXBillingData ReportDate="Fri Sep  8 11:13:19 PDT 2000" MinTrk="3"
MaxDate="07/06/00">
<Partners Sndr="jim1" Rcvr="smani1" Type="810" BillingCode="BC000">
  <Item Doc="0000000006-000001-001-00000001" ReceiveDate="07/05/00 14:33"
    Size="729"Priority="m"/>
  <Item Doc="0000000006-000001-001-00000002" ReceiveDate="07/05/00
    14:33" Size="904" Priority="m"/>



</Partner>
<Partners Sndr="jim1" Rcvr="smani1" Type="850" BillingCode="BC000">
  <Item Doc="0000000003-000001-001-00000001" ReceiveDate="07/05/00 14:33"
    Size="729" Priority="m"/>
  <Item Doc="0000000003-000001-001-00000002" ReceiveDate="07/05/00 14:33"
    Size="904" Priority="m"/>
  <Item Doc="0000000004-000001-001-00000001" ReceiveDate="07/05/00 14:33"
    Size="729" Priority="m"/>
</Partner>
<Partners Sndr="jim1" Rcvr="tom" Type="850" BillingCode="BC000">
  <Item Doc="0000000005-000001-001-00000001" ReceiveDate="07/05/00 14:33"
    Size="729" Priority="m"/>
</Partner>
</ECXBillingData>
```

Billing Data Utility

The utility is a perl script. It takes some boundary parameters, such as minimal or maximum TRKId, minimal or maximum receiving date, then searches the database and generates the XML data file. The syntax of the script is as follows.

```
billing_code -ecx ecx_home -pwd ecx_ora_usr_password [-trk
[min_trk][:max_trk]]
    [-date [min_date][:max_date]] -o|-a output_fn
```

The boundary of the TRKId can be specified as *-trk mintrkid:maxtrkid* where *mintrkid* or *maxtrkid* can be omitted if the user does not want to have the restriction on minimum or maxmium TRKId: *-trk mintrkid* or *-trk :maxtrkid*. The -date option follows the similar pattern.

The script can either overwrite (-o) or append (-a) the output file.

Chapter 21

Customizing Reports

This chapter describes how you can use the Actuate Report System to create custom reports for use with ECXpert. This chapter contains the following sections:

Overview

The Actuate Report System is a very powerful database reporting tool. Actuate comes with hundreds of pages of documentation. This chapter does not attempt to cover much of the information provided by Actuate; rather, this chapter provides just enough information to get started using Actuate with an ECXpert database. You should find Actuate easier to use after reading this chapter.

You will probably find that you need some knowledge of the SQL Select statement if you want to do anything complicated. Although Actuate builds a Select statement for you when you specify the fields you want to display in your report, you still need to know how to interpret the Select statement.

You will also need to refer to the ECXpert database schema described in "ECXpert Database Schema" on page 369. The schema identifies the fields you can use to create the report and the relationships between tables.

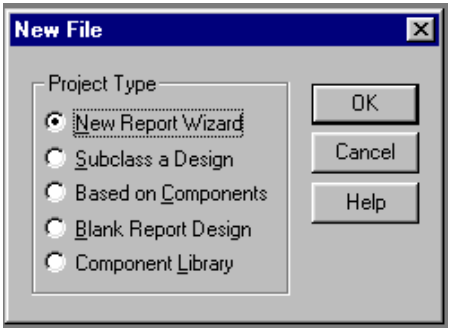
There are many strategies for creating reports and learning how to interpret the data in the ECXpert database. The strategy shown in this chapter is to first create a report that uses an individual table, then create a report that uses multiple tables and groups data. If you follow this strategy, you will learn how easy it is to use Actuate's basic features. You will also become familiar with the contents of the database tables that you are interested in. When you are ready to create your own multiple-table reports, you will be familiar with both Actuate and the data from which your report is prepared.

CAUTION The ECXpert Version 3.6 database schema on which you build your reports is subject to change in future versions of the ECXpert System. You should consider the potential reimplementation effort associated with an upgrade to the database when deciding how much effort you want to invest creating custom reports.

Starting a New Report

You create reports with Actuate's Developer Workbench. After you start the Workbench, choose New from the File menu. You are prompted for the kind of project. Choose New Report Wizard to create your report, as shown in Figure 21-1.

Figure 21-1 Choosing the project type

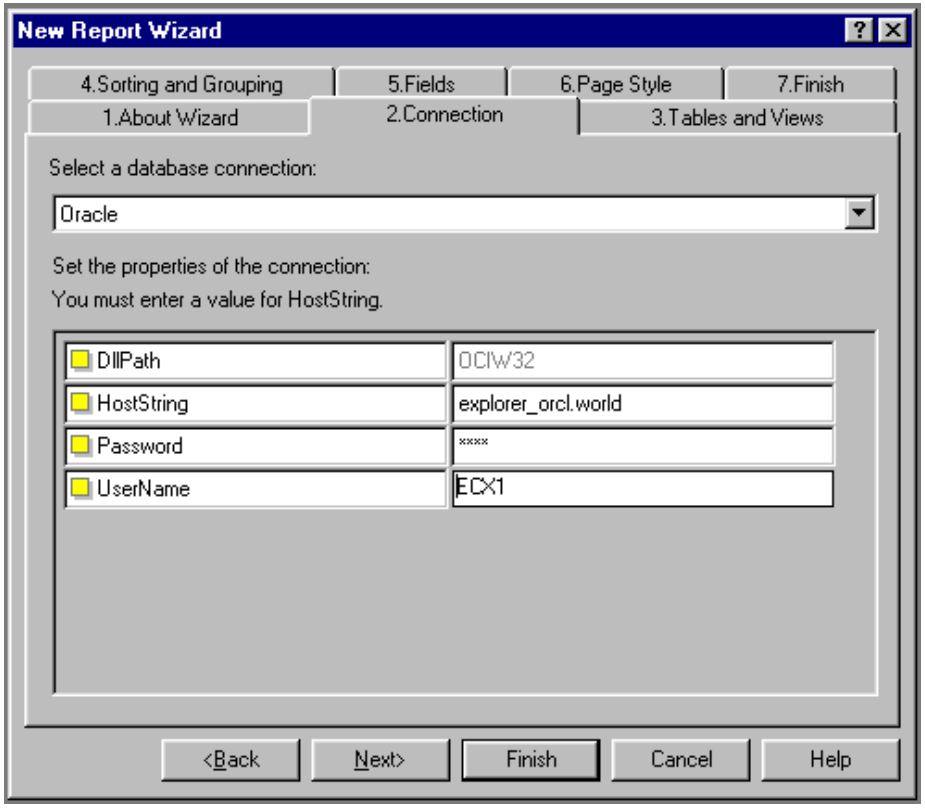


After you choose OK, the wizard specification box appears. You can fill in all of the sections; however, you need not fill in any. You may find it convenient to fill in Section 2, "Connection," as shown in Figure 21-2. This section allows you to specify the kind of database connection (Oracle), the default user name, password, and host.

NOTE Filling in these connection parameters does not connect you to the database. To ensure that your configuration is correct, you can run Oracle's SQL*Plus or a standard report provided with ECXpert using the connection parameters.

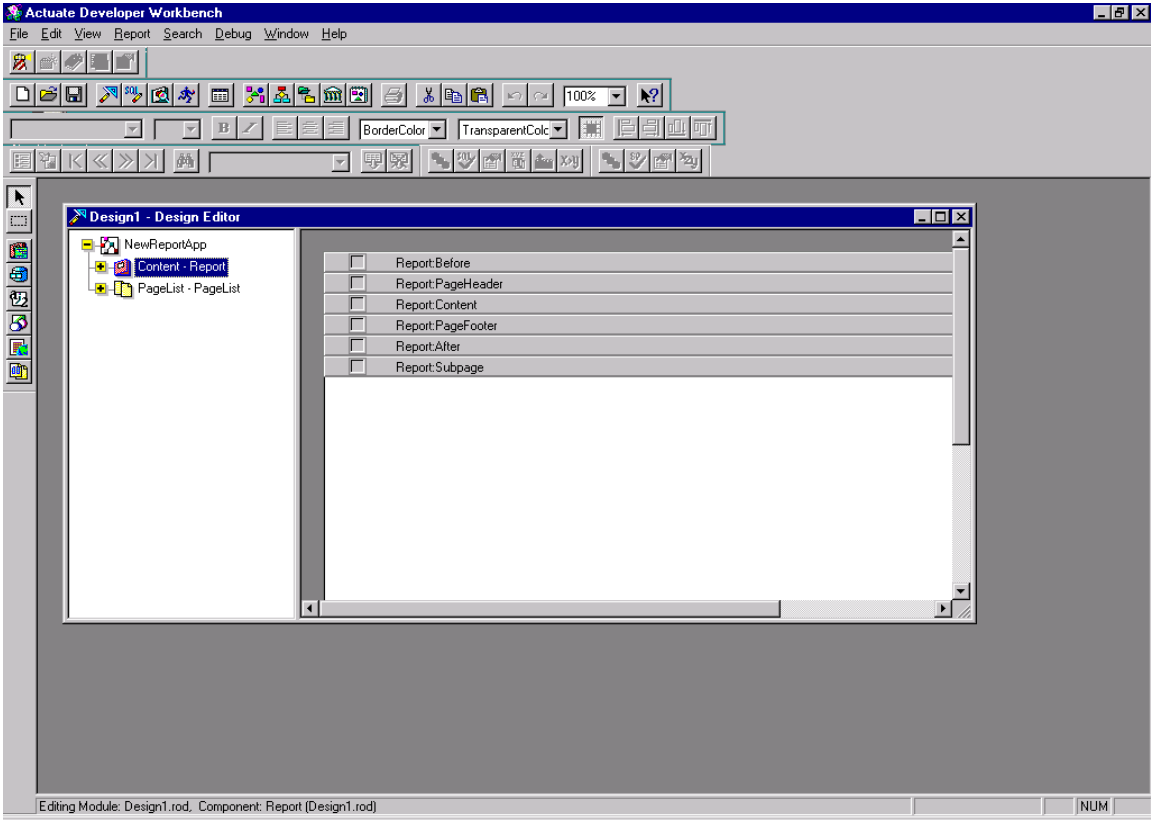
When you finish with the wizard specification, choose Finish.

Figure 21-2 New report wizard



When the New Report Wizard box closes, you are placed in the Design Editor, as shown in Figure 21-3. This editor has two parts; the structure pane on the left and the layout pane on the right. The structure pane shows all of the objects created by the New Report Wizard. You do not need to work with them yet. Just select one of them, such as NewReportApp, and then choose Data Source from the View menu to start describing the SQL Select statement you want your report to execute.

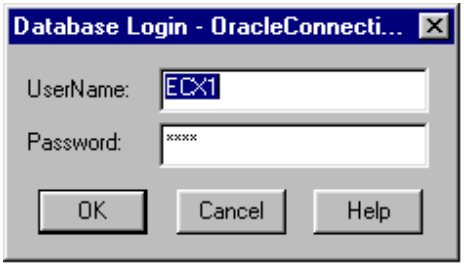
Figure 21-3 The design editor



Building a Query

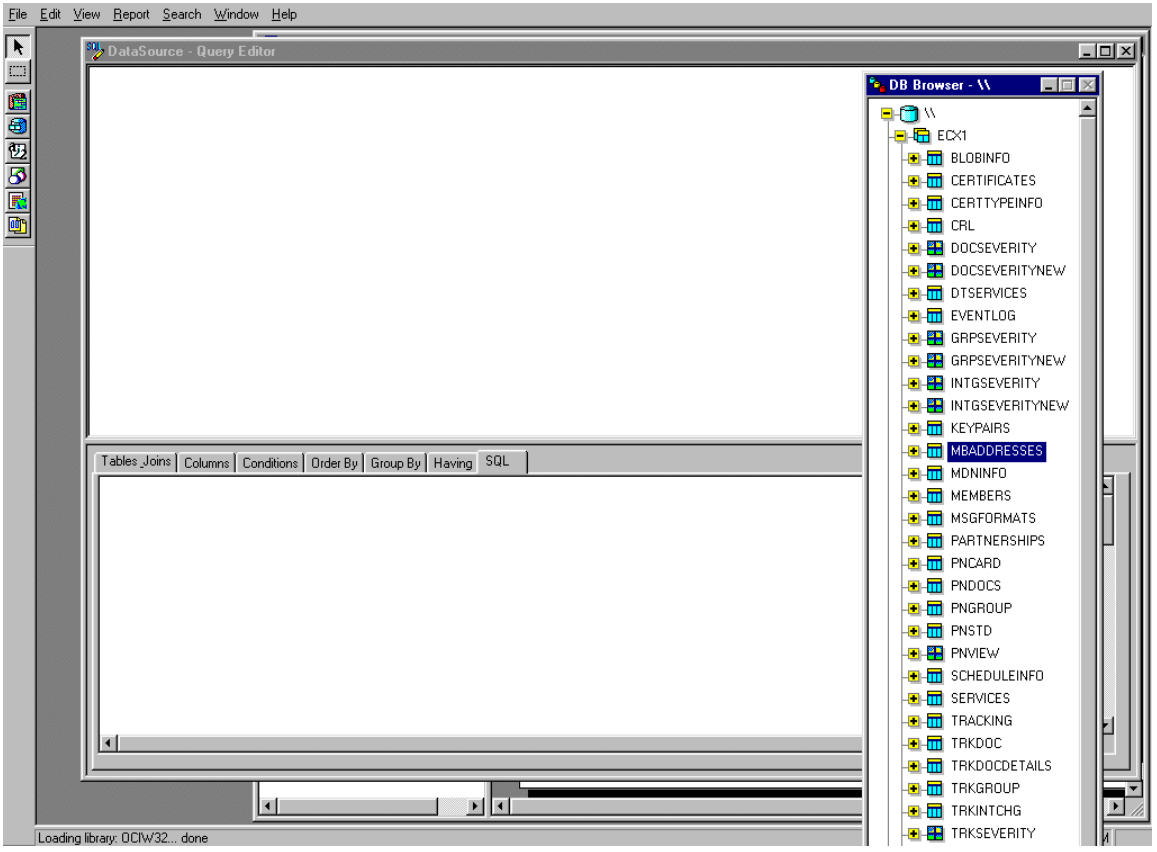
You can build a SQL Select statement to drive your report in the Query Editor. If you are not logged into a database, Actuate prompts you for the user name and password using the default values that you specified in the New Report Wizard, as shown in Figure 21-4. You can change the values if you want. If you want to change the host, you must change the Connection object in the structure pane; see Figure 21-24 on page 354. Figure 21-4 shows the login dialog.

Figure 21-4 Login dialog for Oracle



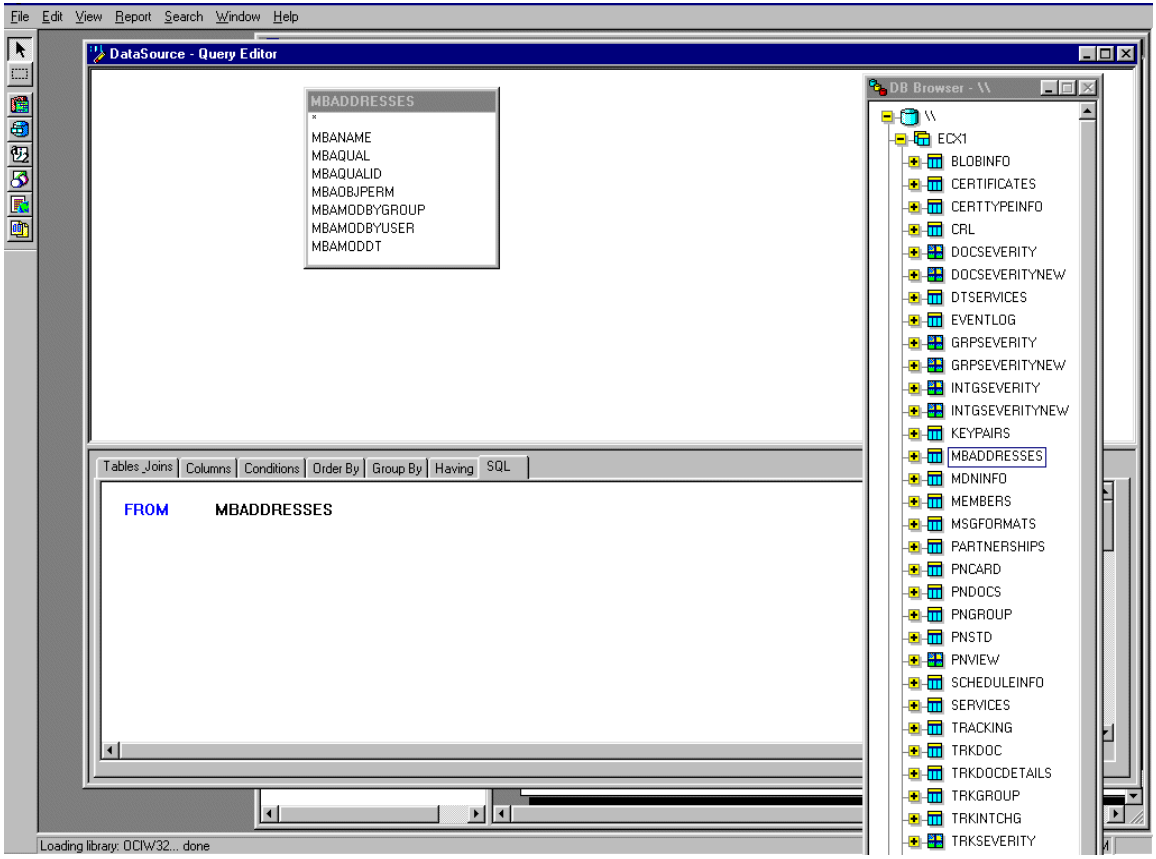
After you log in, the Query Editor appears, as shown in Figure 21-5. It consists of a pane on top for visually representing data and a tab-selected pane on the bottom for entering and viewing the SQL statement specification. A database browsing window is also available for selecting tables.

Figure 21-5 The query editor



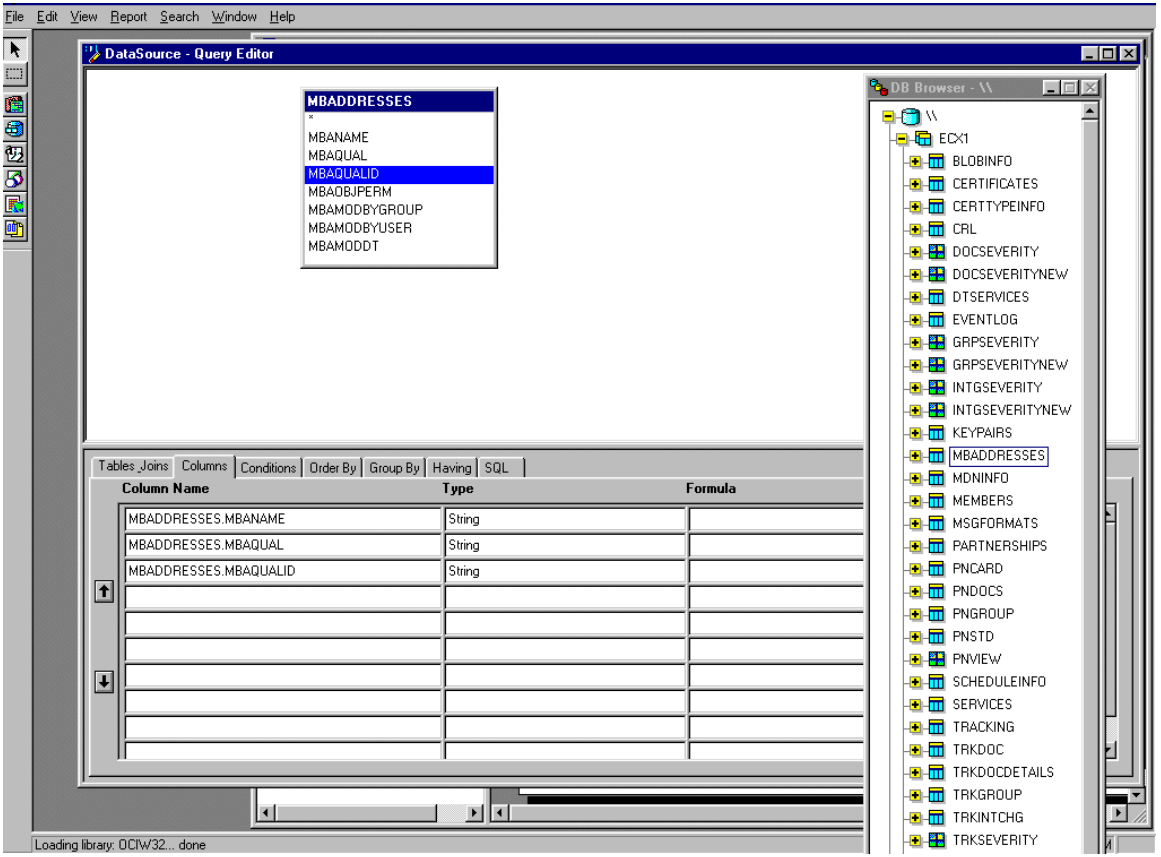
You select the tables you want to use from the database browsing window and drag them into the upper pane. In this part of the example, only MBADDRESSES is used, as shown in Figure 21-6. Dragging a table into the upper pane modifies the From clause in the SQL-tab of the lower pane.

Figure 21-6 Dragging tables to the visual pane



After you drag the tables to the upper pane, select the Columns tab from the lower pane. This allows you to drag columns from the table in the upper pane and drop them under Column Name in the lower pane. You can drag and drop the asterisk (*) if you want to quickly select all columns in the table. Figure 21-7 shows the Query Editor after MBANAME, MBAQUAL, and MBAQUALID have been selected.

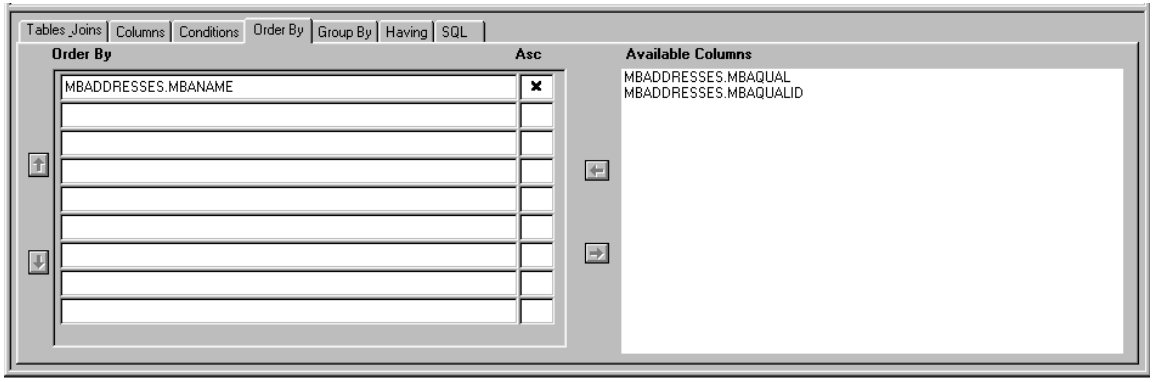
Figure 21-7 Selecting columns



After you have selected your tables from the database browser window and your columns from the upper pane, you can refine your SQL statement by selecting tabs in the lower pane and making further specifications.

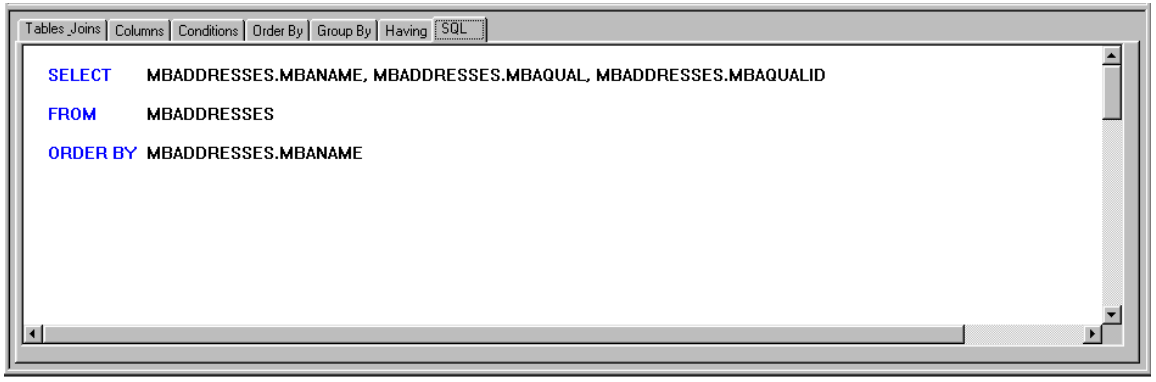
Figure 21-8 shows the lower pane after selecting the OrderBy tab. You can use the pane to specify the Order By clause in your Select statement. In this example, the Select statement is ordered by the MBANAME column.

Figure 21-8 Specifying the Order By clause



When you are finished, you can choose the SQL tab to view the resulting Select statement. Figure 21-9 shows the Select statement that is used in this example.

Figure 21-9 A SQL Select statement



At this point, the Select statement used in the first report has been created. Close the Query Editor to return to the Design Editor.

Laying Out a Report

You use the Design Editor to lay out your report. The following sections show you how to

- create frames for the data you want to display
- set up fields in the frames to display the data
- add headers and footers to your report

Along the way, you will learn how to run the report and view the appearance of your layout.

Creating Frames

A report is divided into various sections. Initially, a report contains the following sections:

- Report:PageHeader for items you want to appear at the top of each page
- Report:Before for items you want to appear only on the first page
- Report:Content for the main content of your report
- Report:After for items you want to appear on the last page
- Report:PageFooter for items you want to appear at the bottom of each page
- Report:Subpage for items you want to appear as a section within a page

Before you can display anything in a report section, you must create a frame and drag it into the section. To create a frame, select the structure tool (third icon from the top) from the left of the structure pane. A structure palette appears. Select and drag a frame structure (fifth icon from the right) from the palette to either the box to the left of the section name in the layout pane or to the corresponding object in the structure pane.

Figure 21-10 shows the structure palette and the Class Name prompt that appears after dropping the form in the appropriate place. Each object is identified by its name; typically, you can accept the default. A discussion of the use of subclasses is beyond the scope of this chapter.

Figure 21-10 Creating a frame

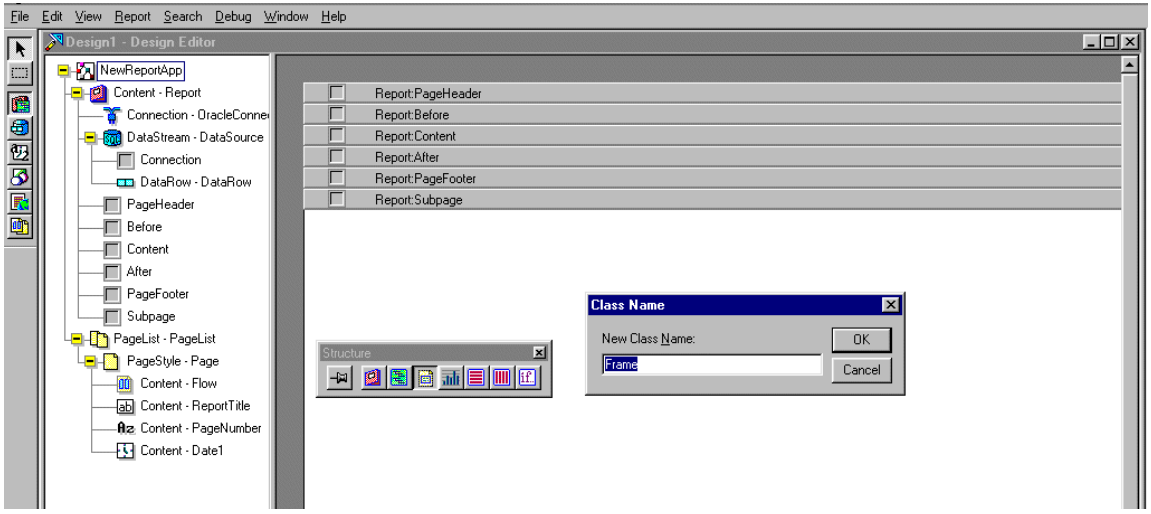
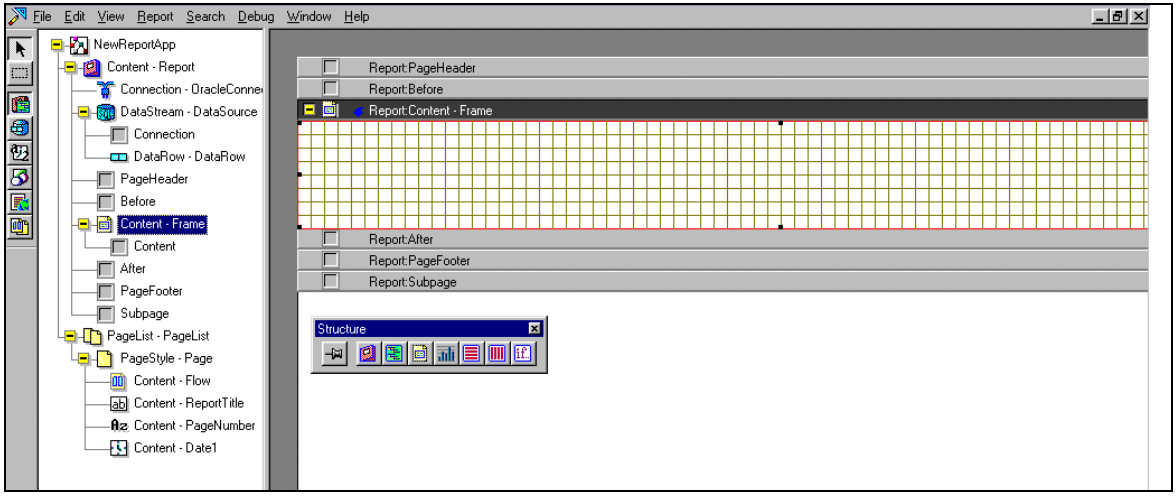


Figure 21-11 shows the frame in the layout pane. Notice that the frame also appears in the structure pane. Everything you place in the layout pane also appears in the structure pane. For any given operation, you can decide which pane is easier to work from.

Figure 21-11 A display frame



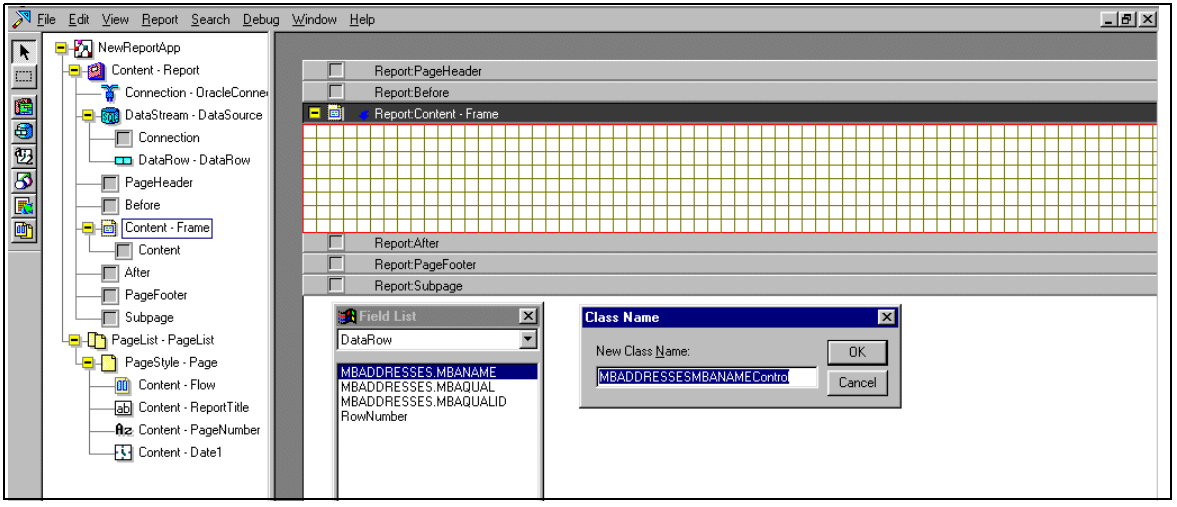
After you create a frame, you can add the items you want to display.

Displaying Data

You can display data in controls. There are several ways to create controls. One way is to use the Field List that Actuate creates when you build your query. To display the Field List, select Field List from the View menu. This menu option toggles whether or not to display the list. You can select one of the fields you specified in your SQL Select statement and drag it onto the frame.

Figure 21-12 shows the Field List and the Class Name prompt that appears after selecting MBADDRESSES.MBANAME from the Field List and dropping it into the frame. After you choose a class name, the control is created in the frame.

Figure 21-12 Using the field list



After you create a control, you can double-click on it to display its Component Editor. The editor shows all the properties of the component.

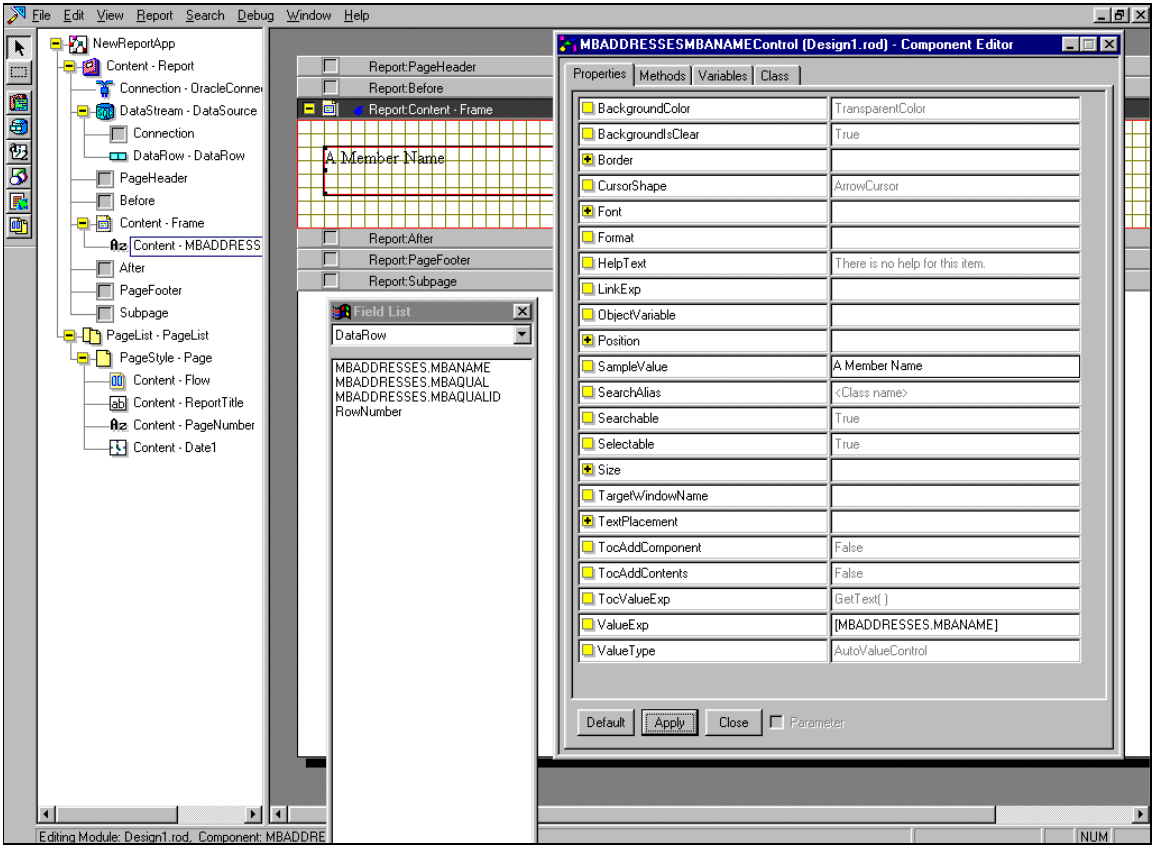
NOTE A Component Editor displays the properties and other attributes of any object, not just controls. Other sections in this chapter show uses of a Component Editor for other kinds of objects.

A Component Editor displays the properties and other attributes of any object, not just controls. Other sections in this chapter show uses of a Component Editor for other kinds of objects.

Figure 21-13 shows the Component Editor for the control. You specify the data to display in the ValueExp property; in this case, it is [MBADDRESSES.MBANAME]. The brackets identify the contents of the property as a column name. The SampleValue property displays a place holder value that appears in the layout pane; in this case, it is "A Member Name," which appears in the field in the layout pane after you select Apply from the Component Editor. You can change other properties, such as the font characteristics and text-placement. You can also change the size and position of the control; however, you may find it easier to do this by selecting the control and sizing or moving it within the frame.

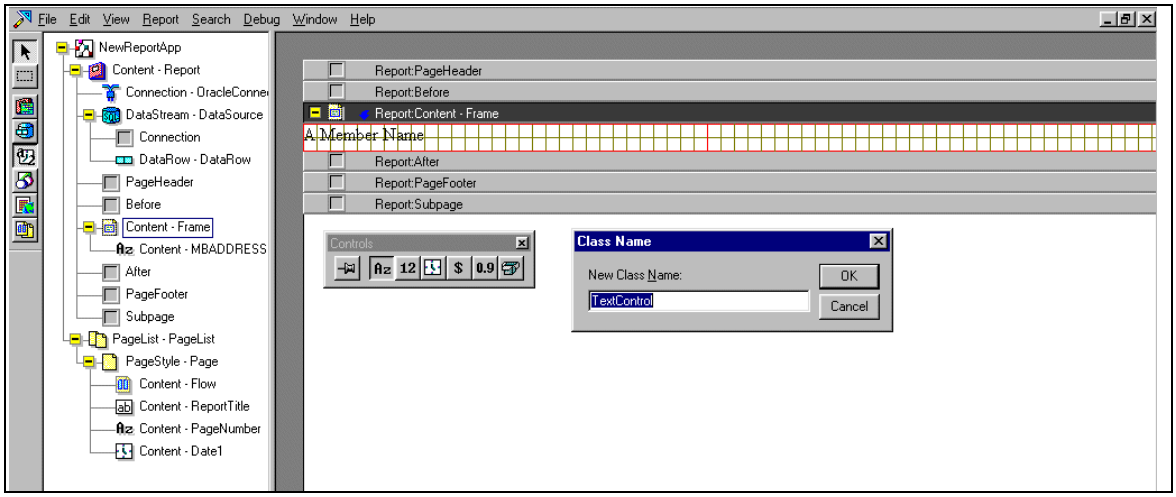
NOTE You can resize a frame in the same way you resize a control; either by changing the size and position in the frame's Component Editor or by selecting and resizing the frame in the layout pane.

Figure 21-13 The component editor



Another way to create a control is to select the control tool (fifth icon from the top) from the left of the structure pane. A Control palette appears. You can select and drag the appropriate kind of control to the frame. Figure 21-14 shows the Control palette and the Class Name prompt after a text control has been selected.

Figure 21-14 Creating a display field

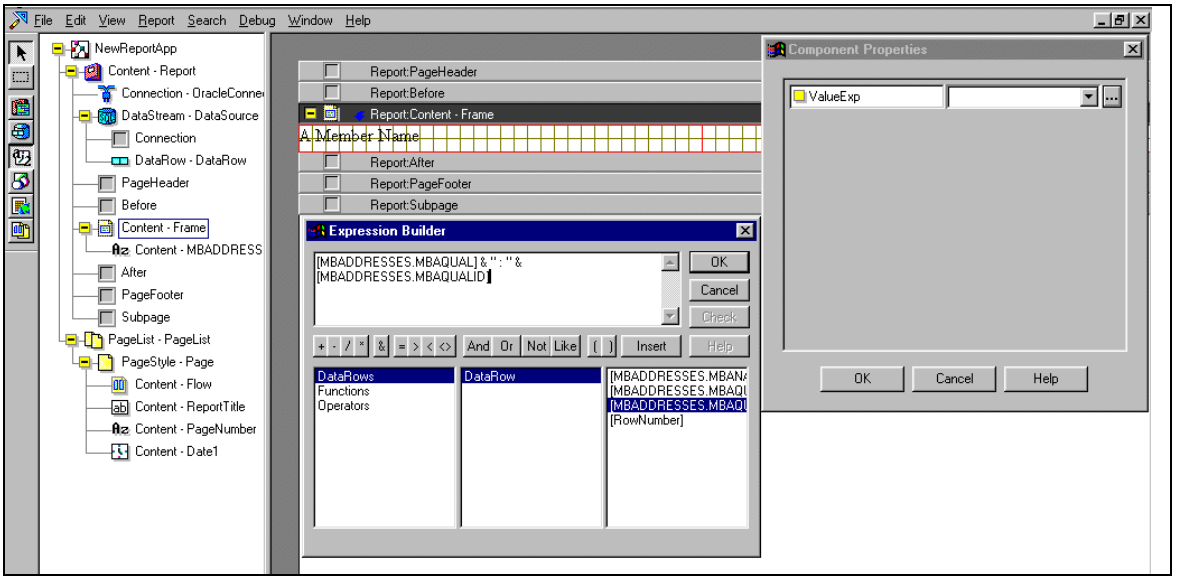


When you create a control using the palette, Actuate prompts you for the value expression. If you want to add static text, you can enter it here within double quotes (“My static text”). You can also choose items from the Field List by selecting the down-arrow icon.

You can create complex expressions, including a combination of text, column names, and functions. To create such an expression, click the ellipses (...) to the right of the down-arrow icon. The Expression Builder appears.

Figure 21-15 shows the Expression Builder after inserting two columns that are concatenated with an intervening colon (“ : “). If you decide to change your expression later, you can open the control’s Component Editor and click the ellipses (...) to the right of the ValExp property; it’s the same property you are prompted for here.

Figure 21-15 Using the expression builder

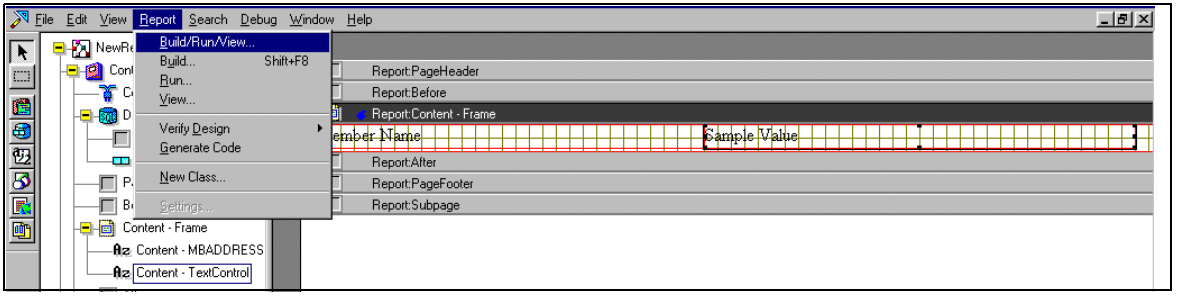


Running a Report

After you have a frame that displays at least one column value from the database, you can run your report. If you want the report to display without data, you must provide additional code that is beyond the scope of this chapter.

To run a report, you must build the report, execute it, and then view the resulting output. You can perform these steps individually from the Report menu, or you can select the Build/Run/View option to perform them all at once. Figure 21-16 shows the Report menu.

Figure 21-16 Building, running, and viewing a report



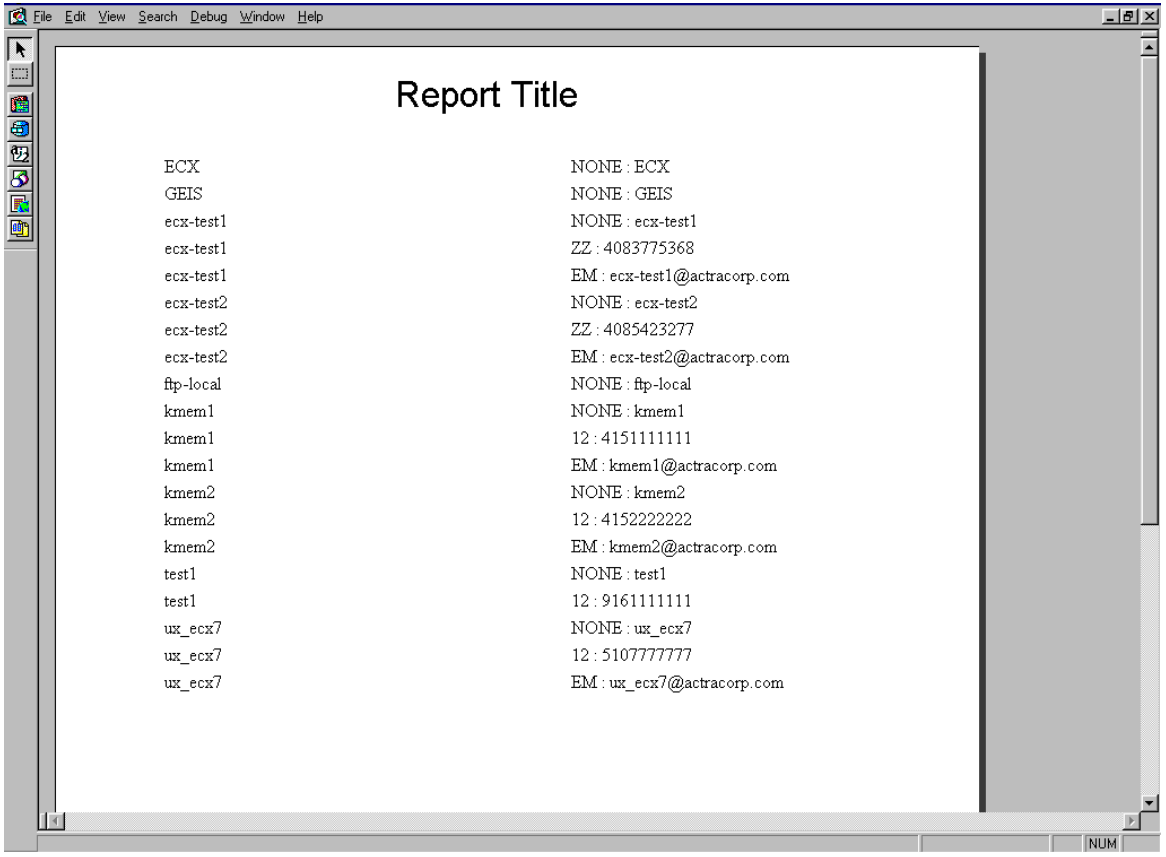
When you run a report, a Requester dialog appears to request values of parameters. Figure 21-17 shows the Requestor dialog. In general, you can ignore the Output Parameters requested by Actuate. For information about adding your own parameters, see "Adding Report Parameters" on page 352.

Figure 21-17 Requester dialog

After you respond with OK to the Requester dialog, the report runs. Figure 21-18 shows the report created thus far.

Along with one line for each row of data, the report shows a generic report title, which is one of the defaults provided by the New Report Wizard. The report also contains page numbers and today's date at the bottom of each page; these are not shown in the figure.

Figure 21-18 The first report



Adding Headers and Footers

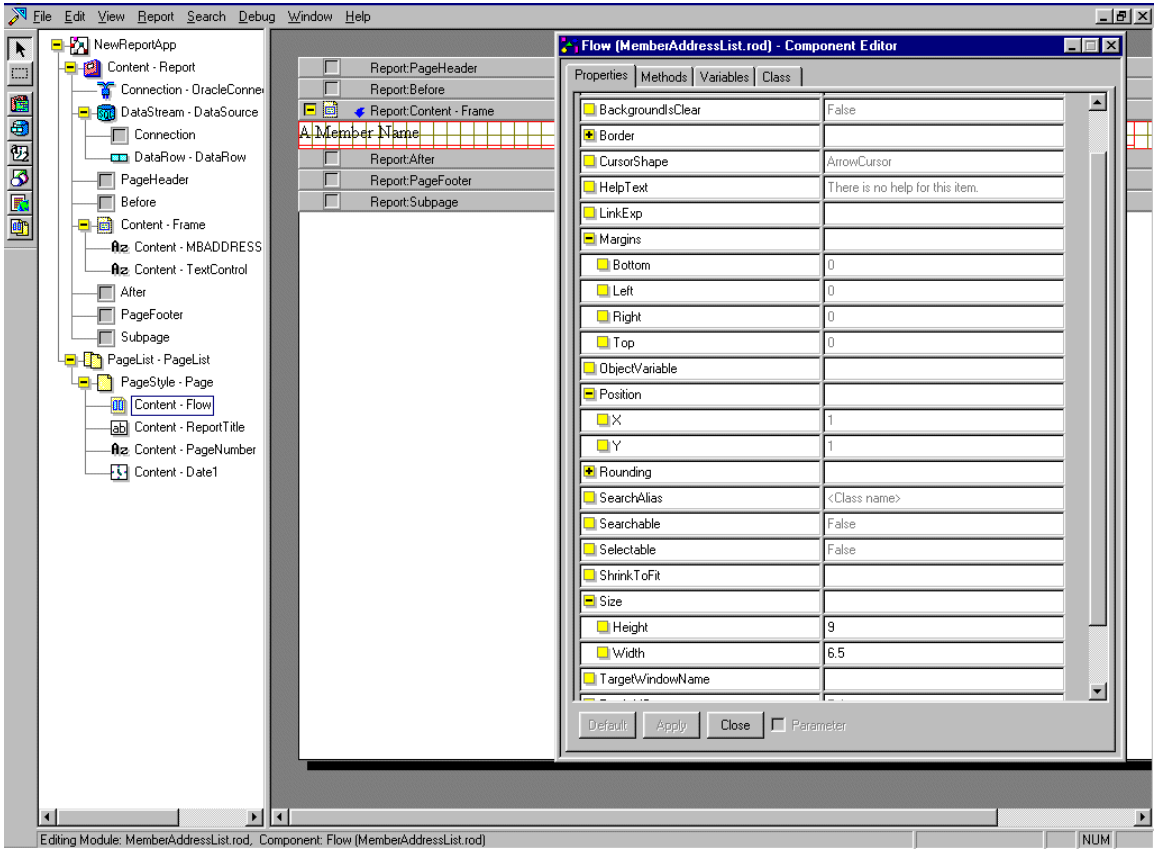
When you use the New Report Wizard to create a report, Actuate creates several objects for you:

- a PageList, which is the container for a page layout
- a Page, which specifies the page size
- a Flow, which defines the printable area of the page
- a text Label for the report title
- text Controls for the page number and the date

You can change the properties of any of these objects in their respective Component Editors; however, these objects are not visible in the layout pane. For example, you can adjust the size property of the flow to effectively change the margins on the page.

Figure 21-19 shows the page list-related objects and the flow's Component Editor.

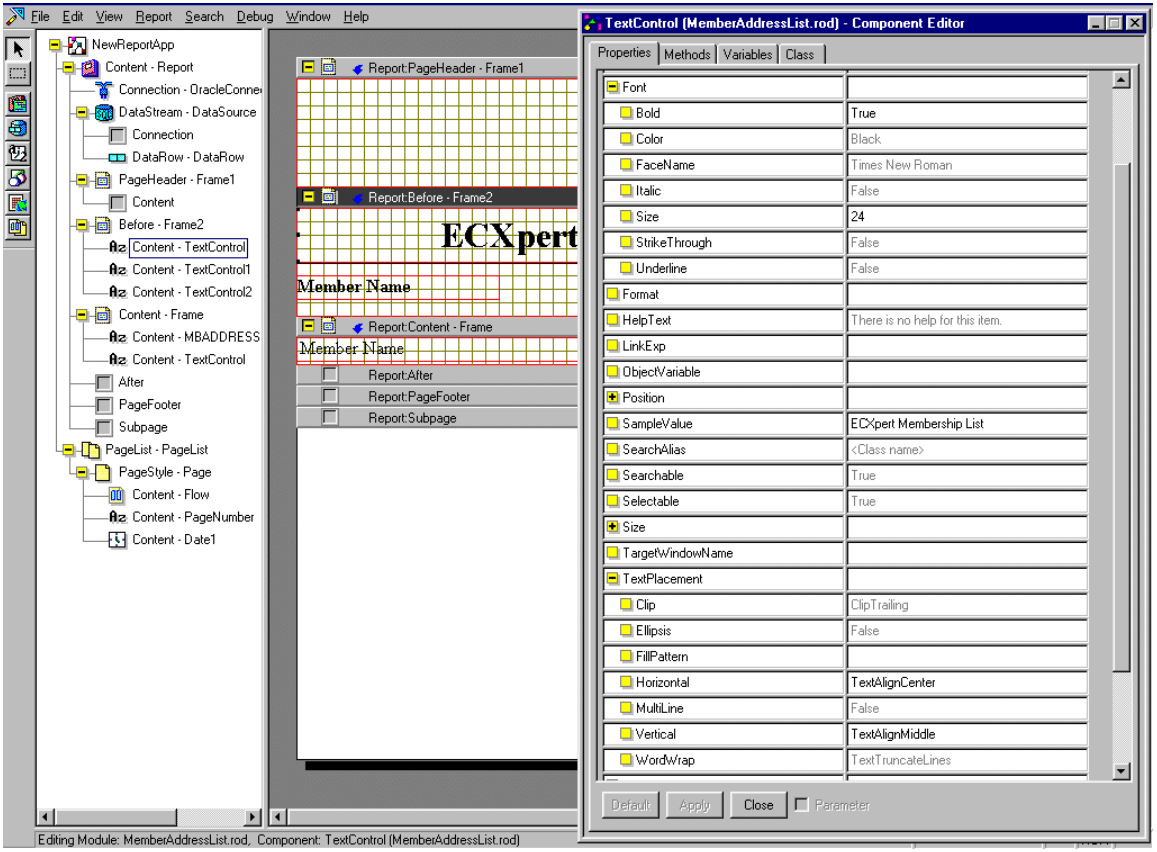
Figure 21-19 Specifying page flow



You can simply modify each page list-object to specify the header and footer for your report; however, you will probably find it easier to create frames and use the layout pane to position them. You can delete an unneeded object by selecting it in either the structure or layout panes and pressing the Delete key.

This example deletes the report title control and replaces it with a frame and related controls in the Report:Before section. Figure 21-20 shows the new report control's Component Editor, in which the font as been changed to 24-point bold and the text has been placed in the center of the control.

Figure 21-20 Adding a report title



The Report:Before section only displays on the first page of the report. You must provide a frame and related controls in the Report:PageHeader section if you want to have a heading on each page. You can copy items from one frame to another by holding the Control key while dragging the object.

To add a footer, you must provide a frame and related controls in the Report:PageFooter section. One way to set up controls in this section, after you create the frame, is to drag the page number and date controls from the page to the frame.

By default, the page footer does not appear on the first page of the report. To change this situation, you must open the Content - Report object's Component Editor and change the Page property's ShowFooterOnLast field to True.

Figure 21-21 shows the layout pane after adding the Report:Before and Report:PageFooter sections. It also shows the Content - Report object's Component Editor after changing the Page property's ShowFooterOnLast field.

Figure 21-21 Adding a page footer

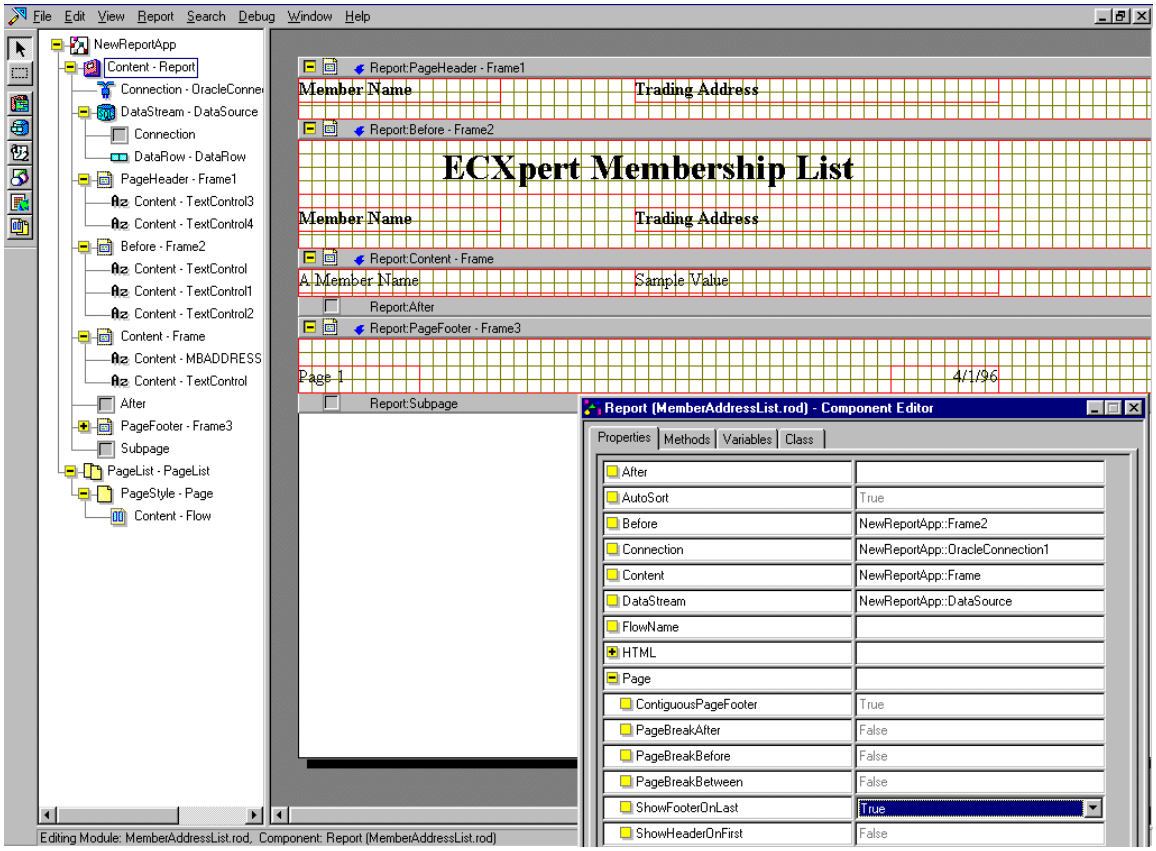
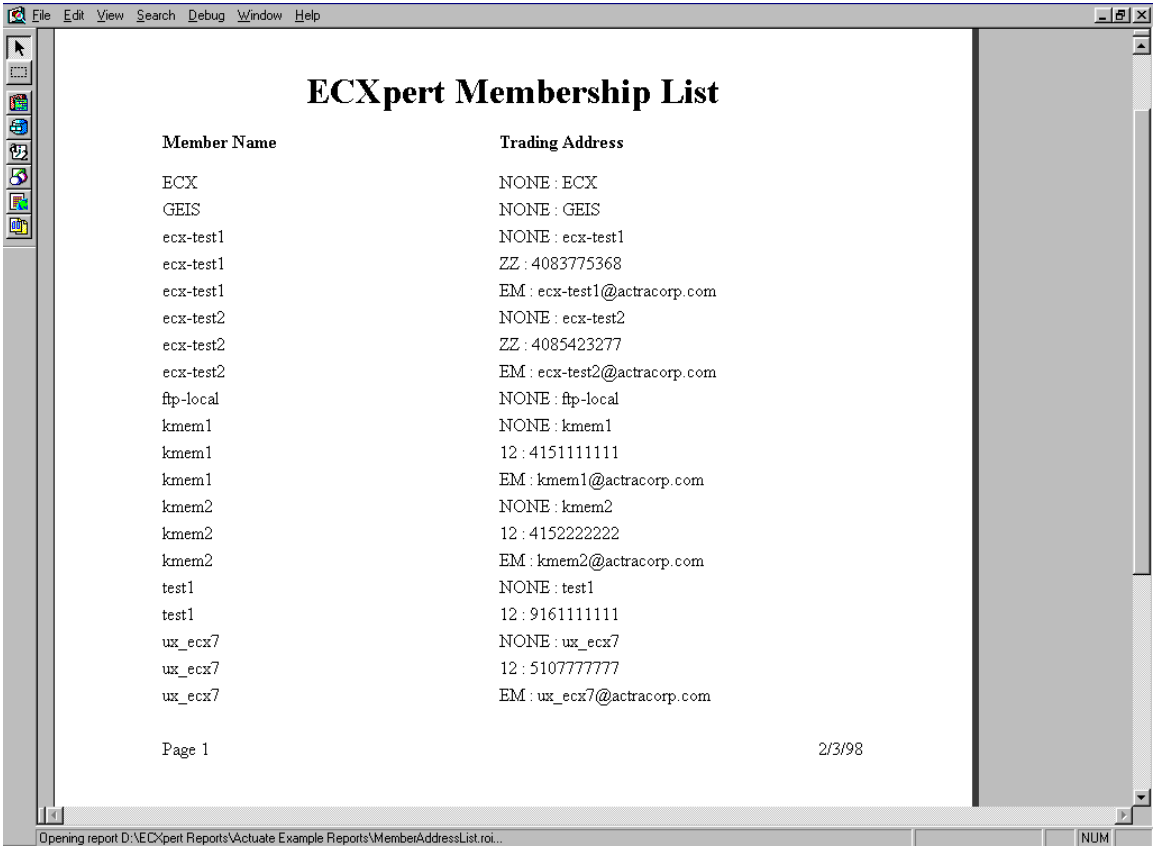


Figure 21-22 shows the report after adding headers and footers. Note that the footer appears right after the last data line. You can set the Page property's ContiguousPageFooter field to False if you want the page footer to appear at the bottom of the last page.

Figure 21-22 A report with headers and footers



Adding Report Parameters

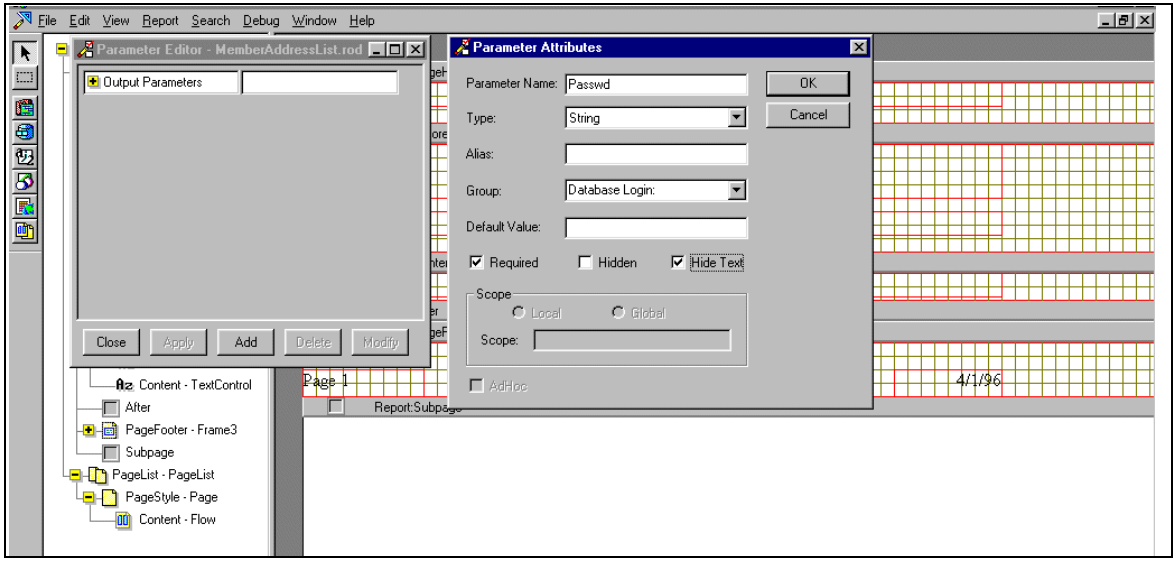
When running a report, the requester dialog appears to prompt for values of parameters that control the report's execution. You create a parameter by selecting Parameters from the View menu, which causes the Parameter Editor to appear. For each parameter, choose Add and fill in the attributes.

The parameter name must be a valid name for programming purposes; as you will see, parameters are used in very simple code you must write. You must provide a data type that represents the kind of data you want to use; for example, String for alphanumeric data, Integer for whole numbers, and Double for decimal numbers. You can provide a group name to display several parameters under the same heading. You can also specify whether a value is required, whether or not it will be displayed at all, and whether what the user types will be displayed.

NOTE The Requester dialog displays groups and non-grouped parameters in alphabetical order and displays grouped parameters in alphabetical order within each group. You can only control the placement of items in the Requester dialog by naming them appropriately.

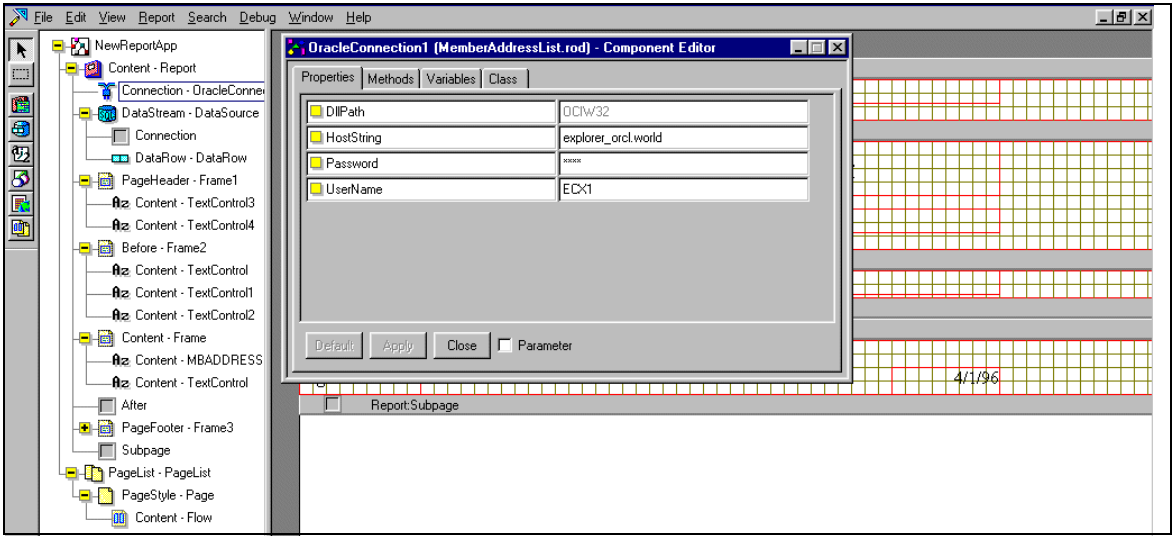
Figure 21-23 shows the Parameter Editor while adding the Passwd field to the "Database Login:" group. A value is required and, because Hide Text is checked, characters typed into the Requester dialog for this parameter will be changed to asterisks (*) when the report is invoked. (Two other parameters created in this example are not shown; they are the LoginName and Server parameters.)

Figure 21-23 Adding a parameter



After you create a parameter, you can use it to set the value of a property. The Passwd, LoginName, and Server parameters in this example are used to set properties of the Connection object, which is shown in Figure 21-24.

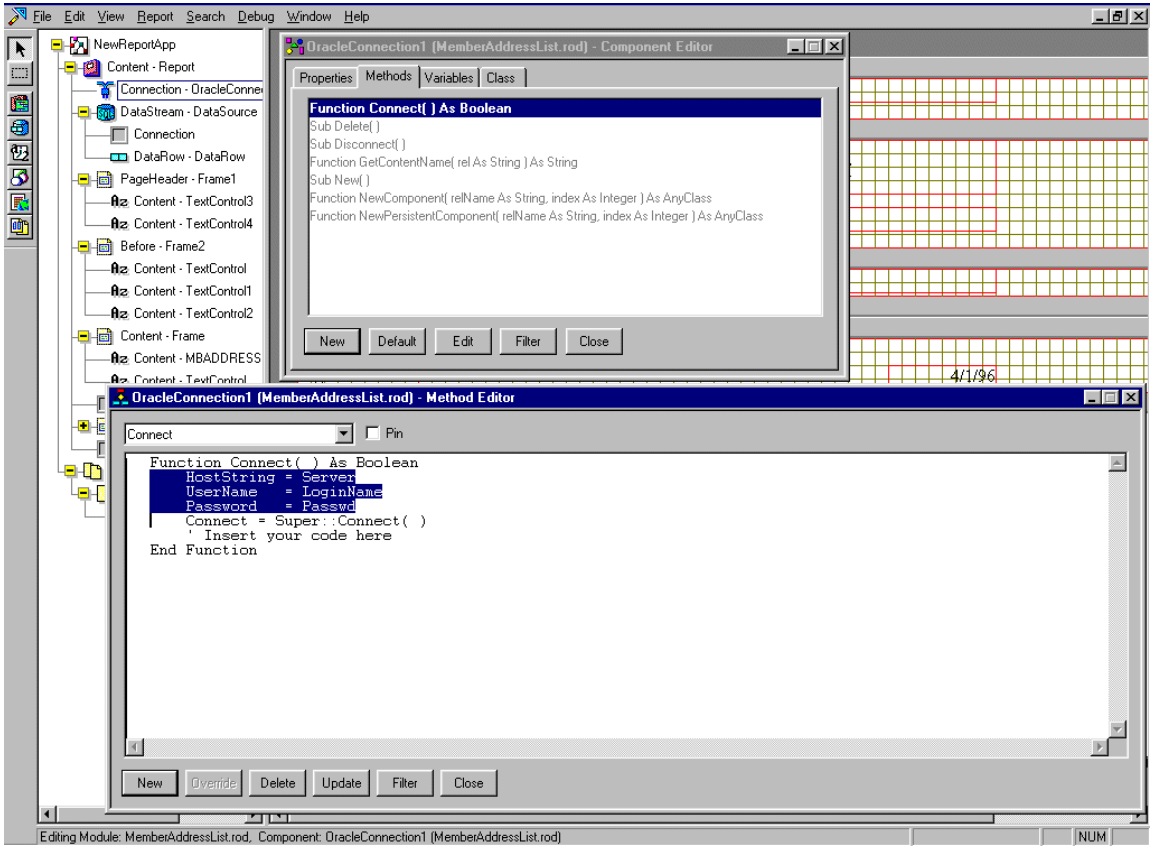
Figure 21-24 The connection component editor



One of the powerful features of Actuate is the ability to customize its operation by overriding various methods. You must override the Connection object's `Connect()` method, for example, to set its properties before Actuate establishes the connection with the database.

Figure 21-25 shows how to set the properties. The parameter values are assigned to the properties. The properties are specified on the left-hand side of the assignment.

Figure 21-25 Setting properties from parameters



NOTE In general, you can assign value to a property in any object programmatically, allowing the property to be set or changed when the report executes. A description of the execution sequence of Actuate methods and when to override specific methods is beyond the scope of this chapter.

Figure 21-26 shows the Requester dialog after the newly added parameter values have been filled in.

Figure 21-26 Entering parameter values

Building Complex Queries

The section "Building a Query" on page 335 describes how to build a query in the Query Editor. This section shows how you can join tables in the Query Editor and how you can create dynamic queries in which the Where clause of a select statement is created when the report executes.

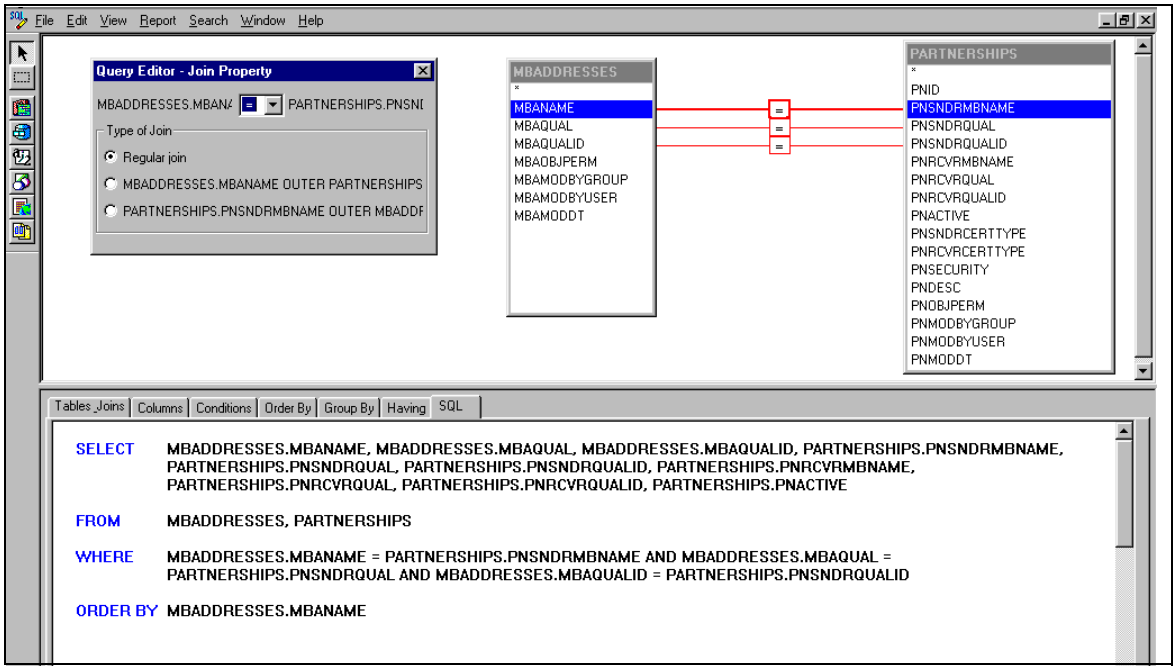
Joining Tables

To join tables, start the Query Editor and choose the column name you want to join in one table. Drag the name to the column you want joined in the other table. This action creates a join based on equality, which specifies selection where ever the column values for the rows are the same.

The action of dragging from one column to another causes a line to appear between the columns with a join icon in the middle. You can double-click the icon to display the join's Property Editor. The Property Editor allows you to specify other relationships between the tables. It also allows you to specify outer joins.

Figure 21-27 shows the MBADDRESSES table joined with the PARTNERSHIPS table on three fields. The Property Editor for the fist join is also shown. The SQL tab in the lower pane shows the resulting Where clause.

Figure 21-27 Joining tables



In some cases it is not possible to describe the Where clause visually. For example, you cannot describe visually the Where clause in the following statement, which is used by the Partnership example being built here:

Code Example 21-1 A complex Where clause

```
SELECT
        MBADDRESSES.MBANAME,
        PARTNERSHIPS.PNSNDRMBNAME, PARTNERSHIPS.PNSNDRQUAL,
        PARTNERSHIPS.PNSNDRQUALID, PARTNERSHIPS.PNRCVRMBNAME,
        PARTNERSHIPS.PNRCVRQUAL, PARTNERSHIPS.PNRCVRQUALID
FROM
        MBADDRESSES MBADDRESSES,
        PARTNERSHIPS PARTNERSHIPS
WHERE
(       MBADDRESSES.MBANAME = PARTNERSHIPS.PNSNDRMBNAME AND
        MBADDRESSES.MBAQUAL = PARTNERSHIPS.PNSNDRQUAL AND
        MBADDRESSES.MBAQUALID = PARTNERSHIPS.PNSNDRQUALID ) OR
(       MBADDRESSES.MBANAME = PARTNERSHIPS.PNRCVRMBNAME AND
        MBADDRESSES.MBAQUAL = PARTNERSHIPS.PNRCVRQUAL AND
        MBADDRESSES.MBAQUALID = PARTNERSHIPS.PNRCVRQUALID )
ORDER BY
        MBADDRESSES.MBANAME ASC
```

In cases such as this one, you can enter the Where clause in the lower part of the lower pane after you choose the Conditions tab. Enter the Where clause exactly as you want it to appear, without the WHERE keyword. Figure 21-28 shows the Where clause under the Conditions tab.

Figure 21-28 Entering a Where clause in the Query Editor

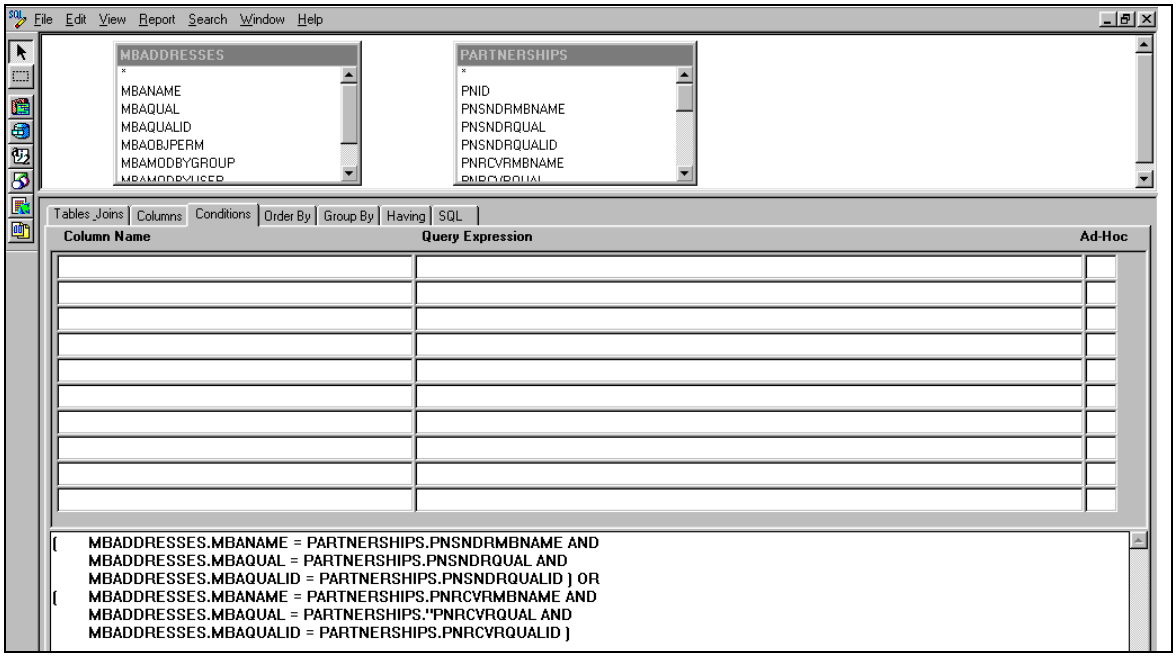
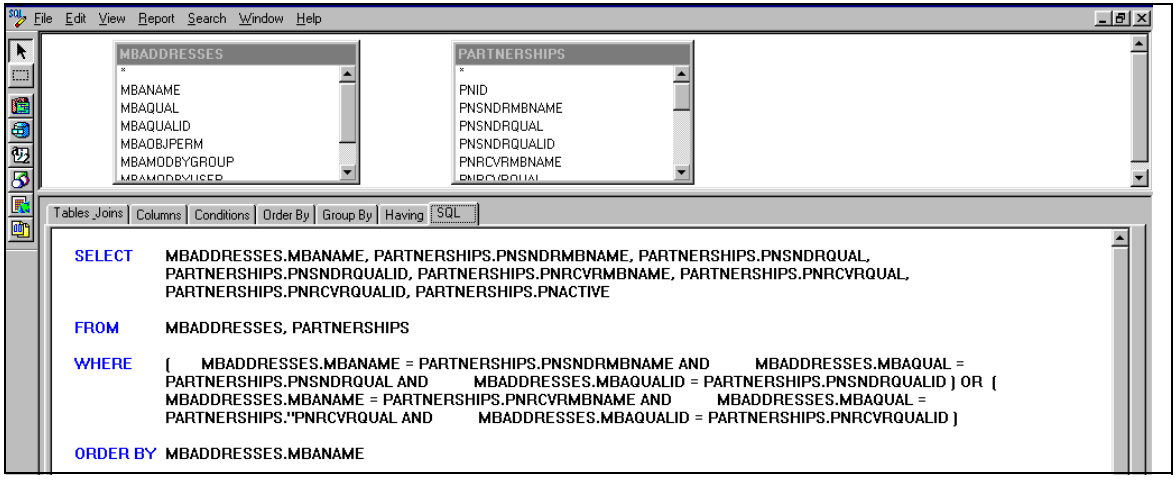


Figure 21-29 shows the resulting Select statement in the lower pane after choosing the SQL tab.

Figure 21-29 The revised Select statement



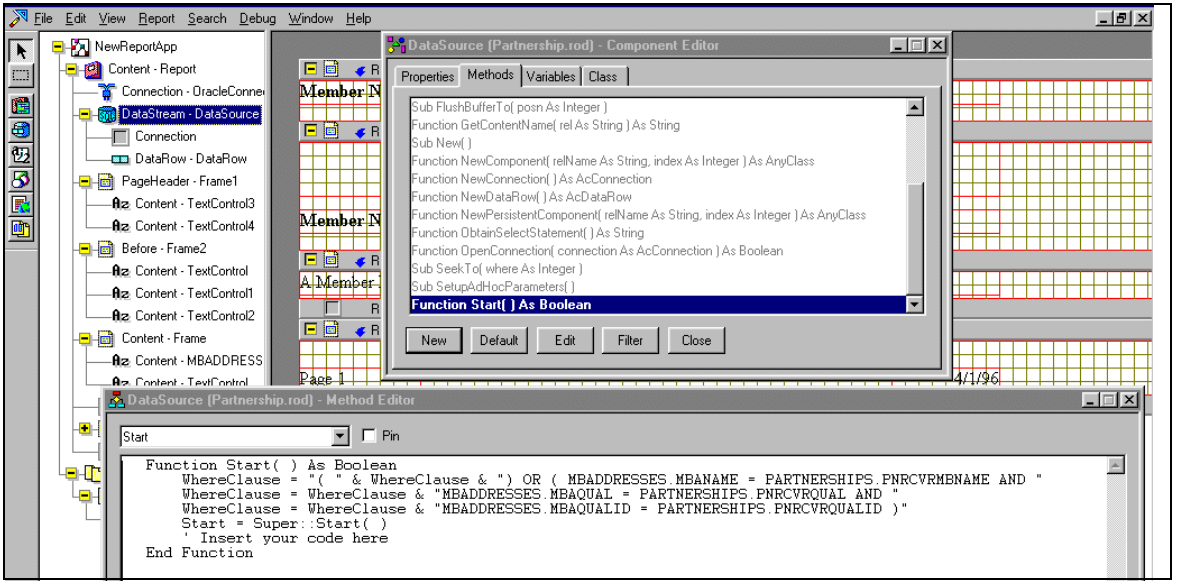
Creating Dynamic Queries

The following example shows an alternative way of specifying the Where clause of a Select statement that you can use to change the query when the report executes. This example dynamically modifies the Where clause shown in Figure 21-27 so that it performs the correct query for the Partnership, as shown in Code Example 21-1. This Where clause in this example does not actually need to be dynamic because it does not require any parameters or control structures (such as “if then, else, end if”); however, this section shows how to set up the clause in a dynamic way.

To create a dynamic Where clause, you must override the DataStream’s object’s `Start()` method. The DataSteam object contains several variables, one of which is WhereClause. You set the WhereClause variable to contain the clause you want to use when you execute the report.

Figure 21-30 shows the WhereClause variable being set so that it modifies the clause in Figure 21-27 to become the one in Code Example 21-1.

Figure 21-30 Specifying the Where clause at runtime



Displaying Groups of Data

Often, you want your report to group data in some meaningful way. This example shows how to group partnerships by member—the member may be either a sender or a receiver. Thus, if member A forms a partnership with member B, the report displays the partnership in a group for member A as well as a group for member B. The Select statement in Code Example 21-1 handles the join requirement. This section shows how to set up the report to display the partnerships grouped by member.

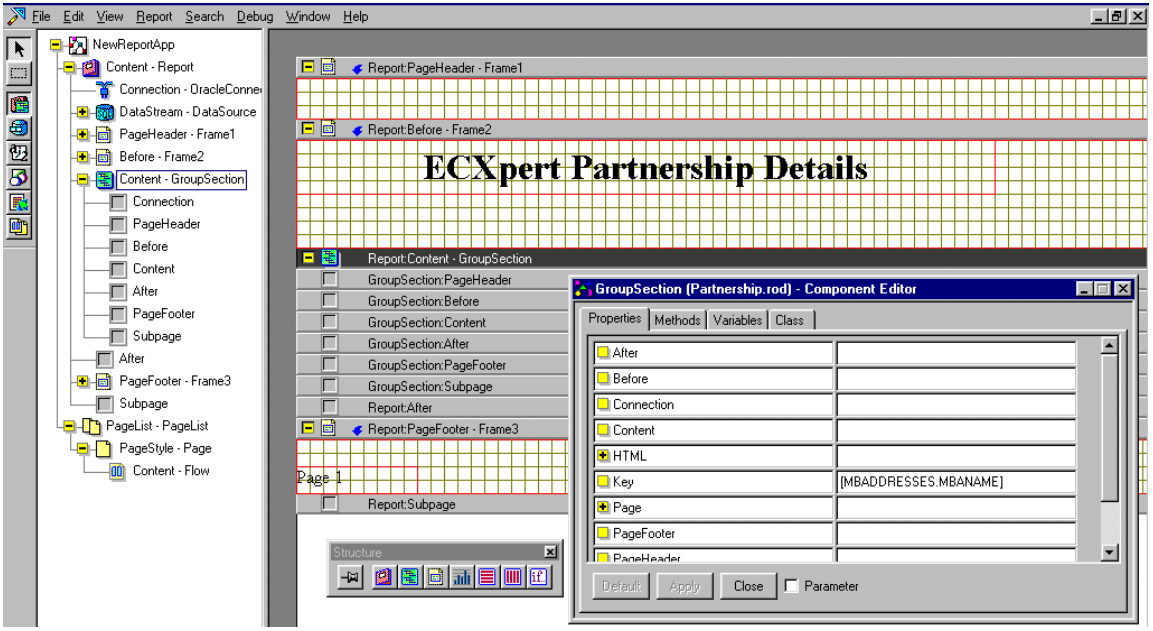
To create a group, you select the Group icon (third icon from the left) from the Structure palette and drag it to the Report:Content section.

CAUTION Unpredictable (and erroneous) results occur if you drop the group object in a Report section other than the Report:Content section.

You must specify a column or variable for the Group section's Key property. A change in the value of the key causes a new group to appear in the report. If you specify a variable for the key, it must be defined in the DataRow object. See "Displaying Row-related Data" on page 364 for more information.

Figure 21-31 shows a the group section's Component Editor in which the key is the member name.

Figure 21-31 Specifying a group key



Within the group, you add frames and controls within the various sections:

- Group:Before contains items to display when the key value changes.
- Group:Content contains items to display as detail lines within the group.
- Group:After contains items to display after the key value has changed but before the next Group:Before section appears in the report.

Figure 21-32 shows a report that displays the member name in the Group:Before section, a partnership row consisting of two lines (one for the sender, the other for the receiver) in the Group:Content section, and a blank line in the Group:After section.

Figure 21-32 Reporting grouped data

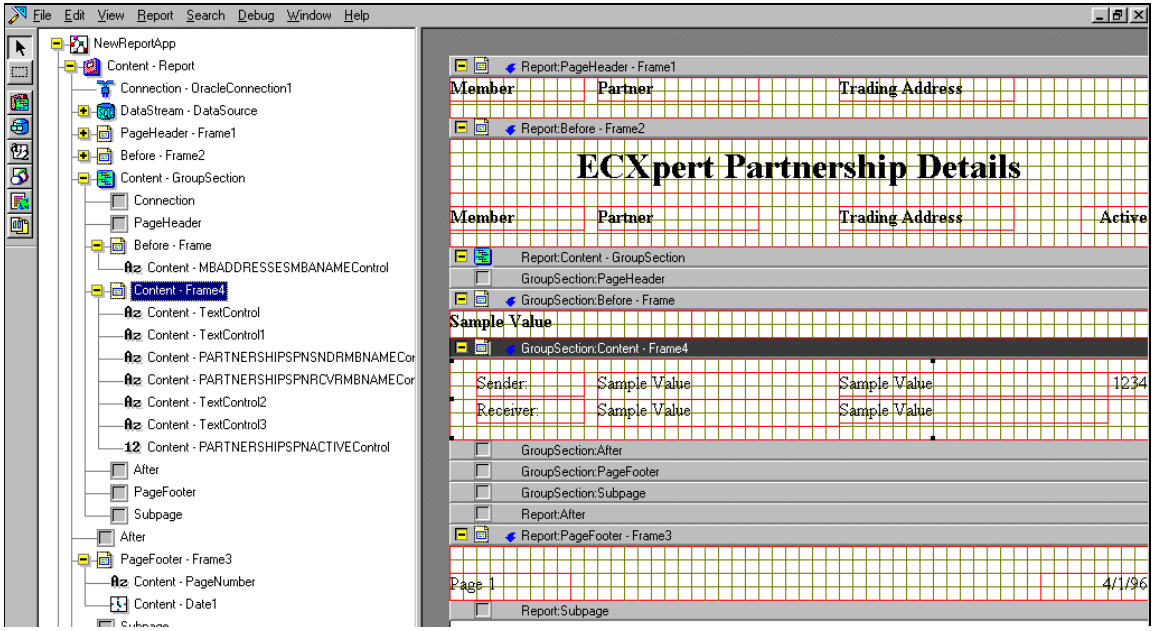


Figure 21-33 shows the output of this report.

Figure 21-33 Output from the grouped data

ECXpert Partnership Details

Member	Partner	Trading Address	Active
ecx-test1			
Sender:	ecx-test1	ZZ : 4085423277	1
Receiver:	ecx-test2	ZZ : 4085423277	
ecx-test2			
Sender:	ecx-test1	ZZ : 4085423277	1
Receiver:	ecx-test2	ZZ : 4085423277	
kmem1			
Sender:	test1	12 : 4151111111	1
Receiver:	kmem1	12 : 4151111111	
Sender:	kmem1	12 : 5107777777	1
Receiver:	ux_ecx7	12 : 5107777777	
kmem2			
Sender:	test1	12 : 4152222222	1
Receiver:	kmem2	12 : 4152222222	

Displaying Row-related Data

The report shown in Figure 21-33 displays a 1 if the partnership is active because it simply displays the integer value stored in the database that represents an active partnership. This section shows how you can create a variable and set its value based on a value in the row. In this example, the variable is created to display a database value in a more meaningful way.

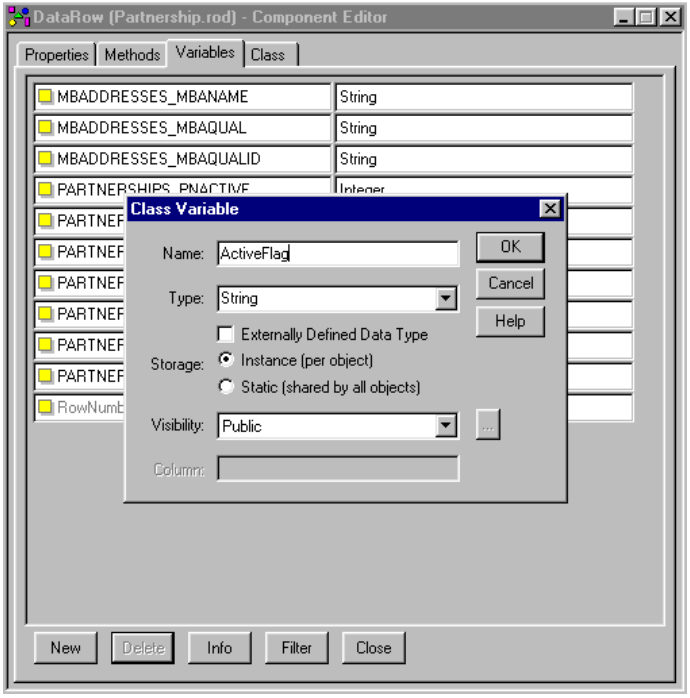
The DataRow object contains a per-instance variable for each column that you specified in your Select statement. These variables are named by concatenating the table name, an underscore character (_), and the column name; for example, PARTNERSHIPS_PNACTIVE is the variable associated with the PNACTIVE column of the PARTNERSHIPS table.

You can add other variables to display additional row-related data. To create a variable, open the DataRow object's Component Editor and choose the Variables tab. Choose New to add a new variable.

A Class Variable prompt appears. In it you specify the variable name, the data type, the kind of variable, and its visibility. (Visibility is beyond the scope of this chapter; choose the default.) For efficiency, specify the type if you know it. If you want Actuate to handle type conversion, specify Variant for the data type. You should specify Instance for a row-related variable; this kind of variable exists for the duration of the row.

Figure 21-34 shows the Class Variable prompt in which the ActiveFlag variable is being created.

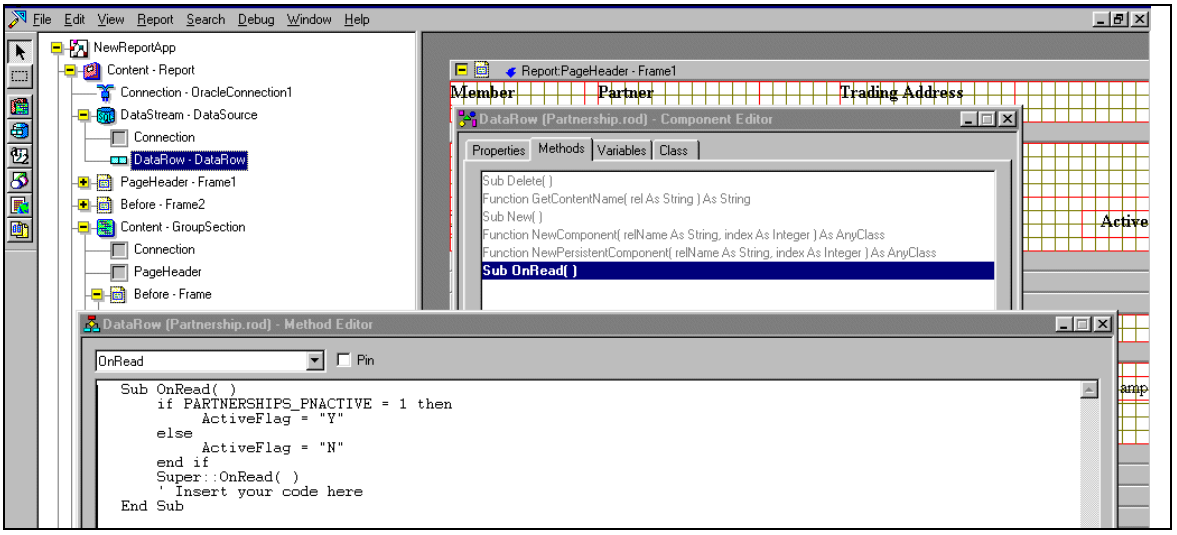
Figure 21-34 Adding a class variable



After you create a variable, you can use it with a column from the database. You specify the relationship between your variable and a column variable by overriding the DataRow object's `OnRead()` method.

Figure 21-35 shows an if-then-else-endif construct that sets ActiveFlag based on the value of the PARTNERSHIPS_PNACTIVE variable.

Figure 21-35 Setting a variable using a row's column value



After you define a variable in the DataRow object, you can use it in the same way as you use a database column. Figure 21-36 shows how to select a row-related variable to display in a text control.

Figure 21-36 Displaying a row-based variable

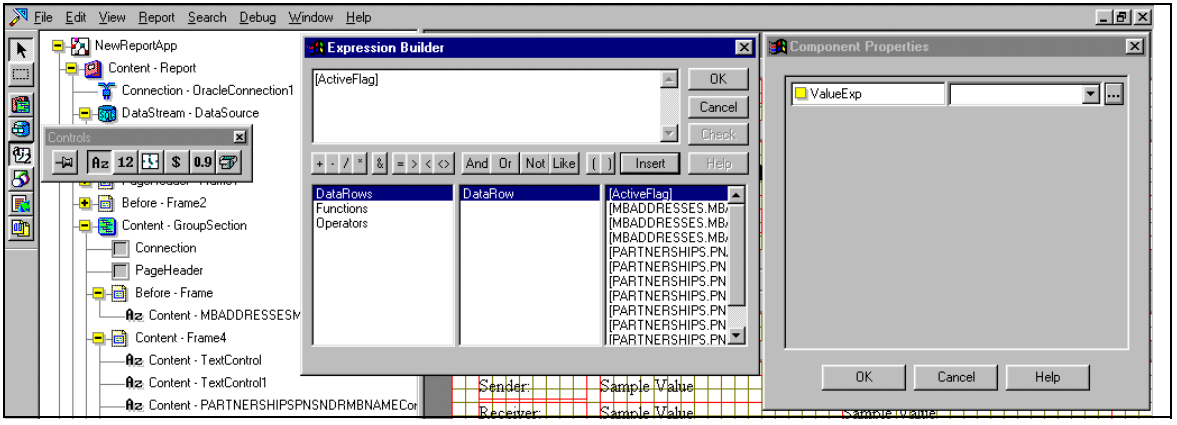
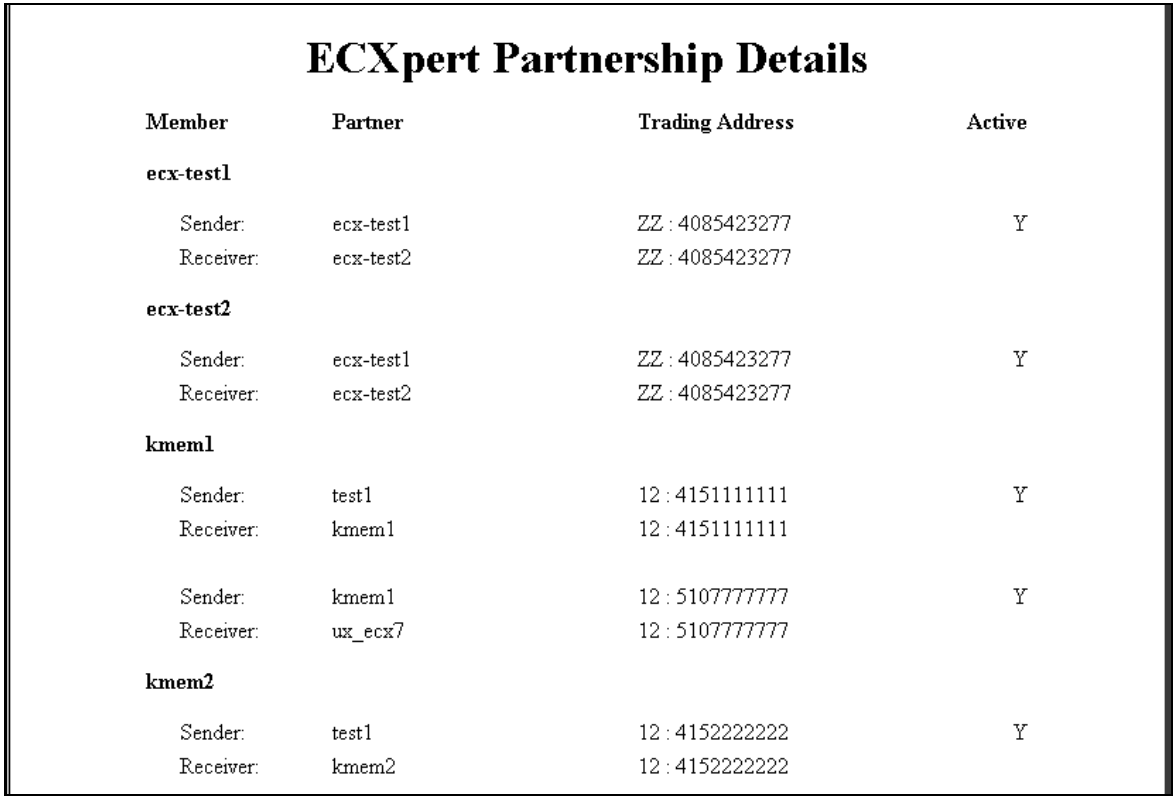


Figure 21-37 shows the report that displays this variable.

Figure 21-37 The second report



ECXpert Partnership Details

Member	Partner	Trading Address	Active
ecx-test1			
Sender:	ecx-test1	ZZ : 4085423277	Y
Receiver:	ecx-test2	ZZ : 4085423277	
ecx-test2			
Sender:	ecx-test1	ZZ : 4085423277	Y
Receiver:	ecx-test2	ZZ : 4085423277	
kmem1			
Sender:	test1	12 : 4151111111	Y
Receiver:	kmem1	12 : 4151111111	
Sender:	kmem1	12 : 5107777777	Y
Receiver:	ux_ecx7	12 : 5107777777	
kmem2			
Sender:	test1	12 : 4152222222	Y
Receiver:	kmem2	12 : 4152222222	

Appendix A

ECXpert Database Schema

This appendix details the table structure of the database underlying the ECXpert System.

The following topics are presented:

Cautions in Using the Database Schema

The database schema for Version 3.6 of the ECXpert System is subject to change in future versions of ECXpert. You should only use the API described in this manual to access the database outside of ECXpert. If you rely on the schema, you should consider the potential reimplementation effort that you could incur as the result of an upgrade to the database.

Extending Table and Rollback Segment Space

You can extend your tablespace size and rollback segment space by following the steps below:

1. Log onto Solaris with your Oracle account. For example:

```
login: oracle
password: oracle
```

2. Launch the Oracle Server Manager utility.

```
# svrmgrl
SVRMGR> connect system/manager
```

NOTE The default password is `manager`; yours may differ.

3. Enlarge the `USERS` and `SYSTEM` default tablespaces.

   For example, if the user default tablespace is `USERS` and the system default tablespace is `SYSTEM`:

```
SVRMGR> alter tablespace USERS
add datafile '/export/app/oracle/product/8.0.4/dbs/usrdataECX20-2.dbf' size
100M;
SVRMGR> alter tablespace SYSTEM
add datafile '/export/app/oracle/product/8.0.4/dbs/systECX20-2.dbf' size 50M;
```

   In the `datafile` command above, change "`size 50M`" to reflect the table space size you want to set. iPlanet recommends you use the following formula to estimate the tablespace size needed for ECXpert:

   - 2.5kB * number of documents received daily * number of days retained

   For example, if you expect to process five documents per day and retain the document information for five days, you should set the table space size to at least 2.5 kB * 5 (documents) * 5 (days retained) = 625kB.

4. Enlarge the rollback segment size.

NOTE For the rollback segment size, estimate 1.5 - 2 times the largest tablespace.

For example, if the user default tablespace is `USERS` and the system default tablespace is `SYSTEM`:

```
SVRMGR> alter tablespace RBS
add datafile '/export/oracle/product/8.0.4/dbs/usrdataRBWG2.dbf' size 300M;
SVRMGR> alter tablespace RBS
add datafile '/export/oracle/product/8.0.4/dbs/systRBWG5.dbf' size 300M;
```

Values of AckState

The AckState column stores the acknowledgment status when Functional Acknowledgments (FAs/997s) or CONTRL messages are requested. The column appears in the TrkIntchg (TIAckState), TrkGroup (TGAckState), and TrkDoc (TDAckState) tables. The actual value of AckState is computed by adding together the applicable combination of the following values shown in Table A-1:

Table A-1 Determining AckState Values

Defined State	Value
ASunknown	0
ASwaiting	1
ASok	2
ASerror	4
ASreject	6
ASpreject	16
ASsent	32
ASsendFailed	64
ASreconciled	128

To understand the usage of these values, we can break the above definitions into three categories:

- **basic state** (Asunknown, ASwaiting)
- **acknowledgment status** (ASok, ASerror, ASreject, ASpreject)
- **acknowledgment flavor** (ASsent, ASsendFailed, ASreconciled)

The acknowledgment status can be added to the acknowledgment flavor to get a complete picture of a document record's corresponding acknowledgment state.

Examples Let's consider some scenarios and see how this would work.

Outbound EDI

In the case of outbound EDI, the map direction is Application to EDI or EDI to EDI. After successful translation, Translate assigns ASwaiting to the document record.

When the 997 or CONTRL is returned in response to this document, this is parsed and the AckState of the gets a flavour of ASreconciled added to the state extracted from the acknowledgment. Thus, if the trading partner rejects this document for whatever reason, the AckState for this document would be ASreconciled added to ASreject.

Inbound EDI

In this case, the map direction is EDI to application. FAgen generates the acknowledgment and assigns an initial status to the document (ASok, ASreject, etc.). When Gateway sends the acknowledgment out, the AckState of the original document is updated with the ASsent or ASsendFailed flavor. Thus, if we reject an inbound EDI document and Gateway succeeds in sending this out, the AckState of this document would be ASsent added to ASreject.

Messages Displayed Table A-2 lists the messages displayed in the Tracking tabs for various values of AckState.

Table A-2 Tracking Messages Associated with Values for AckState

If AckState has...	And...	Message Displayed is...
ASwaiting only added (AckState = ASwaiting)	acknowledgment Overdue Date > current date	Waiting
	acknowledgment Overdue Date <= current date	Overdue
ASreconciled added	ASok has been added to AckState	Reconciled (OK)
	ASerror has been added to AckState	Reconciled (Error)
	ASreject has been added to AckState	Reconciled (Reject)
	ASpreject has been added to AckState	Reconciled (Partial) Reject
	otherwise	Reconciled

Table A-2 Tracking Messages Associated with Values for AckState *(Continued)*

If AckState has...	And...	Message Displayed is...
ASsendFailed added	ASok has been added to AckState	Sent (OK)
	ASerror has been added to AckState	Sent (Error)
	ASreject has been added to AckState	Sent (Reject)
	ASpreject has been added to AckState	Sent (Partial) Reject
	otherwise	Sent
ASsent added	ASok has been added to AckState	Send Failed (OK)
	ASerror has been added to AckState	Send Failed (Error)
	ASreject has been added to AckState	Send Failed (Reject)
	ASpreject has been added to AckState	Send Failed (Partial) Reject
	otherwise	Send Failed
otherwise, if acknowledgment Overdue Date > current date	ASok has been added to AckState	Generated (OK)
	ASerror has been added to AckState	Generated (Error)
	ASreject has been added to AckState	Generated (Reject)
	ASpreject has been added to AckState	Generated (Partial) Reject
otherwise, if acknowledgment Overdue Date <= current date	ASok has been added to AckState	Send-Overdue (OK)
	ASerror has been added to AckState	Send-Overdue (Error)
	ASreject has been added to AckState	Send-Overdue (Reject)
	ASpreject has been added to AckState	Send-Overdue (Partial) Reject

Alphabetical Listing of Tables

The tables in this appendix are ordered by functional groupings. When you know the name of a particular table, you can use the alphabetical listing in Table A-3 to locate it quickly, without reference to the functional groupings.

Table A-3 Tables List by Functional Groupings

Table Name	Functional Grouping	Contents	Page No.
BlobInfo	System	Information about blobs	page 383
Certificates	Keys	Certificate information	page 403
CertTypeInfo	Keys	Certificate information for UI display	page 405
CRL	Keys	Certificate revocation list	page 404
DTServices	System	Service list definitions	page 382
EventLog	Tracking	Log of processing events	page 424
Job	System	Information about scheduled jobs	page 378
KeyPairs	Keys	Public/private key pair information	page 405
MBAddresses	Membership	Member trading addresses	page 386
MsgFormats	System	Text strings for EventLog	page 425
MDNInfo	Tracking	Message Disposition Notification information	page 422
MsgFormats	Tracking	Text strings for EventLog	page 425
Oftp	Tracking	OFTP EERP reconciliation information	page 423
Partnerships	Partnerships	Partnership definitions	page 387
PNCard	Partnerships	Mercator card information	page 400
PNDocs	Partnerships	Partnership document definitions	page 389
PNGroup	Partnerships	Partnership group definitions	page 400
PNStd	Partnerships	EDI standards for partnerships	page 401
Services	System	Service definitions	page 381
Tracking	Tracking	Basic information for submission units (same tracking ID)	page 406
TrkDoc	Tracking	Document-level information	page 414

Table A-3 Tables List by Functional Groupings *(Continued)*

Table Name	Functional Grouping	Contents	Page No.
TrkDocDetails	Tracking	Document card-level information	page 420
TrkGroup	Tracking	Group-level information	page 412
TrkIntchg	Tracking	Interchange-level information	page 409
TrkSegment	Tracking	Document segment-level information	page 420
UniqueKeys	System	Control information for system-generated unique keys	page 383
Versions	System	Information about product and database schema versions	page 380

Schema Overview

Figure A-1 shows the relationship between the membership, partnership, services, and key-related tables in the ECXpert database schema.

Figure A-1 Diagram of database schema for membership, partnerships, services, and certificates

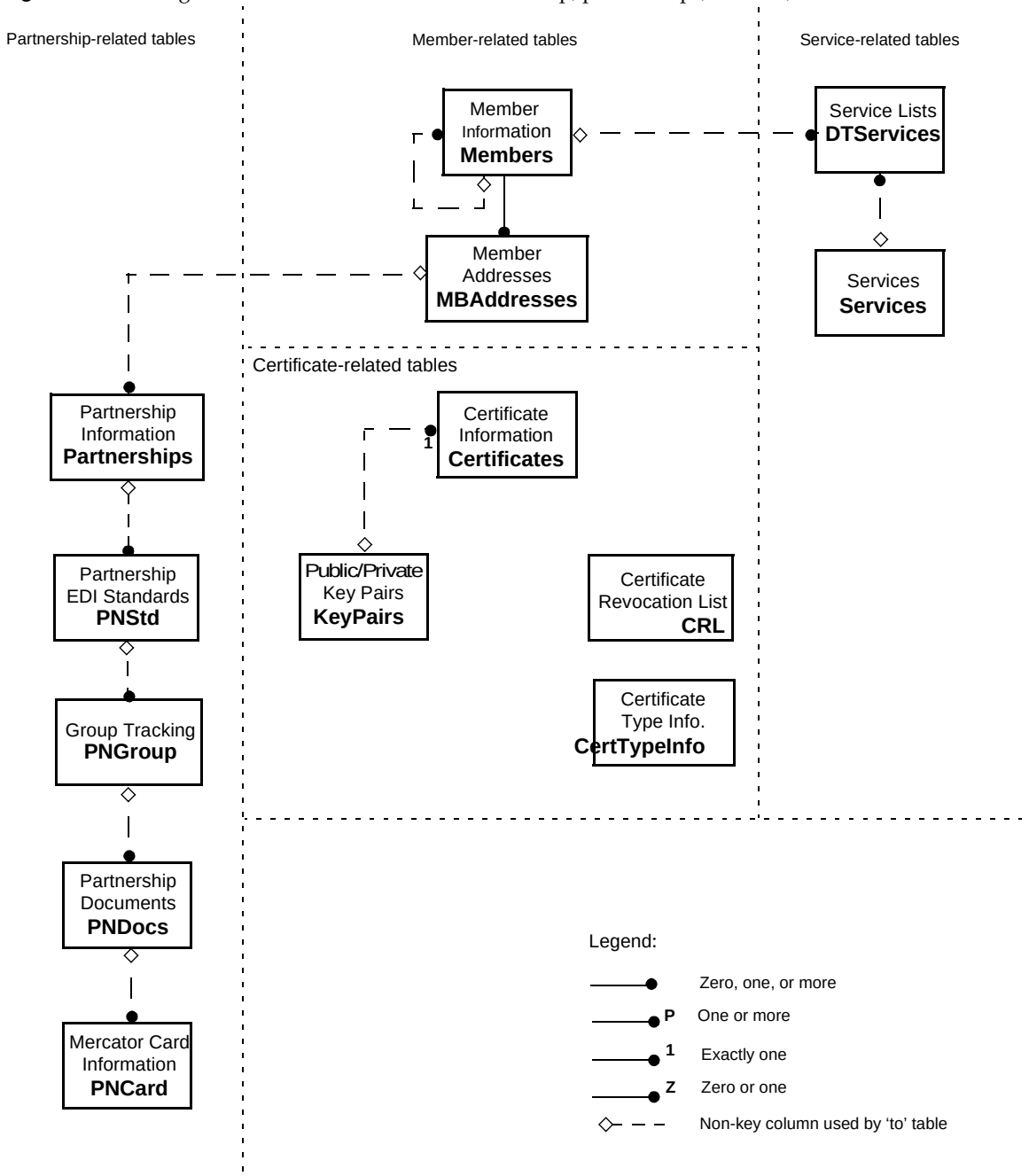
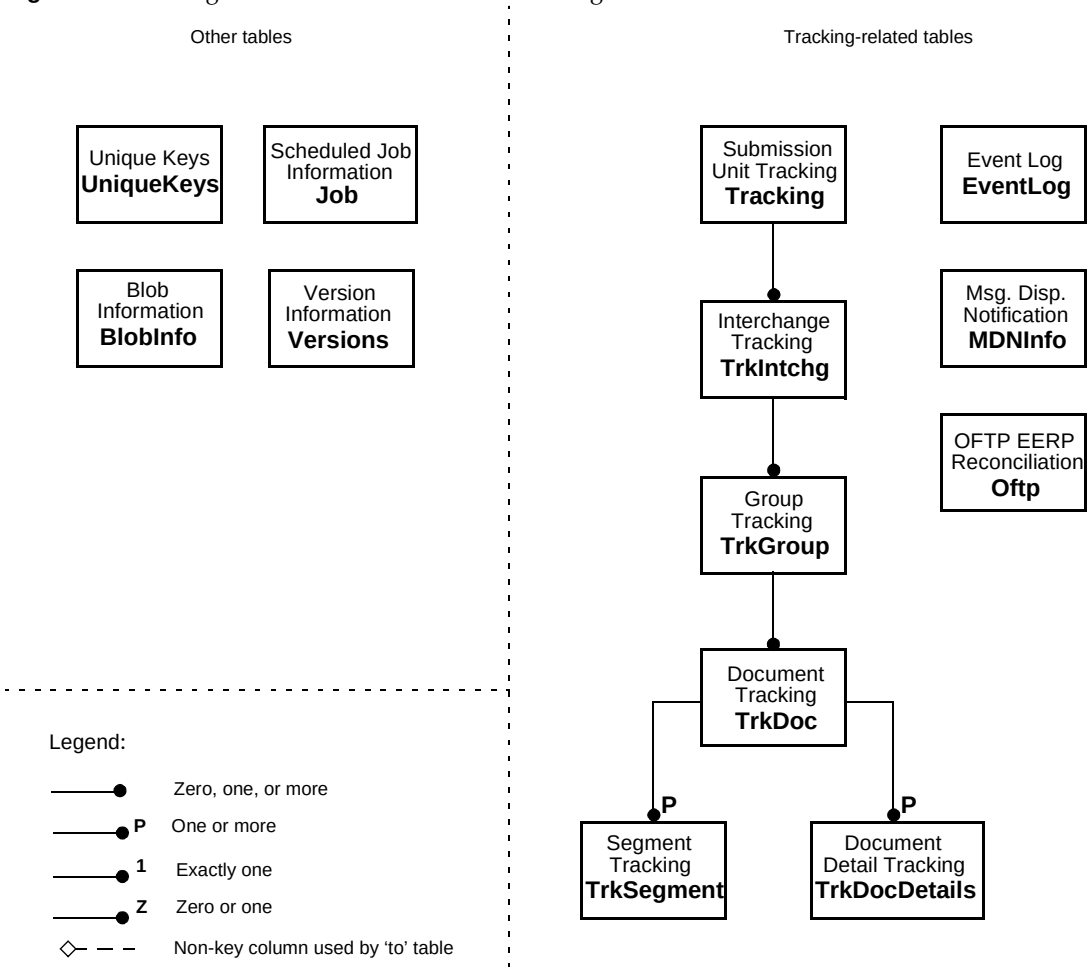


Figure A-2 shows the relationship between the tracking-related tables in the ECXpert database schema. It also shows other tables in the ECXpert database schema.

Figure A-2 Diagram of database schema for tracking and other tables



System-wide Tables

The system-wide group of tables store information that is used throughout the ECXpert System.

Job

Table A-4 stores information about scheduled jobs.

Table A-4 Jobs Table

Name	Req	Type (Len)	Description
JBId [1]	Y	varchar2(60)	Unique ID of scheduled job
JBDescription		varchar2(255)	Description of scheduled job
JBExecType		integer	Type of scheduled job (e.g. script, daemon, etc.)
JBExecName		varchar2(60)	Pathname to an executable or a script, or the section name of an ECXpert server
JBExecArgs		long	Arguments passed to scheduled job
JBExecCfgFile		varchar2(60)	Used internally for daemon
JBExecPktId		integer	Used internally for daemon
JBCriterionId		integer	Blob ID for job running criteria
JBBlkoutId		integer	Blob ID for blackout criteria
JBRepeatFrequency		integer	Seconds between each time the job is to be run
JBRunTotal		integer	Total number of times job is to be run
JBIteration		integer (default 0)	Current iteration of the scheduled job

Table A-4 Jobs Table *(Continued)*

Name	Req	Type (Len)	Description
JBState		integer (default 0)	Current state of the scheduled job. Valid states are: 0 - Job is submitted 1 - Job is waiting for the evaluation of its criteria 2 - Job is ready to run 3 - Job is running 4 - The previous run is done 5 - Job is all done 6 - Job is held (suspended) by user 7 - Job is aborted due to non-recoverable error
JBLastRunRetCode		integer	Return code from last iteration of scheduled job
JBLastRunErrno		integer	Error number from last iteration of scheduled job
JBLastRunErrMsg		varchar2(255)	Error message from last iteration of scheduled job
JBLastRunTime		date	Starting time of last iteration of scheduled job
JBPrevEvalTime		date	Time of most recent evaluation of criteria
JBNextEvalTime		date	Time of next evaluation of criteria
JBLogLevel		integer	Indicates logging level (e.g. warning, error, etc. Valid logging levels are: Lower than 10 - informational 10 - 20 - warning ('(' means exclude while ']' means include) 20 - 30 - error Higher than 30 - no logging
JBModByGroup		varchar2(60)	ID of group modified by
JBModByUser		varchar2(60)	ID of user modified by

Table A-4 Jobs Table *(Continued)*

Name	Req	Type (Len)	Description
JBModDt		date	Modification date. Default: system date.

1. Primary key

Versions

Table A-5 stores information about the current version of ECXpert and the current version of the database schema.

Table A-5 Versions Table

Name	Req	Type (Len)	Description
Product		varchar2(30)	Product name (ECXpert).
ProductVersion		varchar2(20)	Version number of the product
SchemaVersion		varchar2(20)	Version number of the database schema
MBModDt		date	Modification date

Services

Table A-6 stores definitions of individual services that can be combined into service lists in the DTServices table.

Table A-6 Services Table

Name	Req	Type (Len)	Description
SVRId[1]	Y	integer	Service ID. Valid values: 201 =parse 203 =translate 205 = FAGen 207 = OutPrep 209 = OutParse 211 = Routing 213 = Split 704 = Gateway 2001 and above = custom services
SVRName		varchar2(60)	Service name (e.g., parse)
SVRType		integer	Service type. Valid values: 0 = STunknown 1 = STinternal (ECXpert internal service, e.g. parse, xlat) 2 = STscript (ECXpert external script file) 3 = STexe (ECXpert external executable file) 4 = STdll (function in a shared library, e.g. DLL)
SVRPathName		varchar2(255)	Path name to service code file
SVREntryName		varchar2(60)	Entry name
SVRMaxThread		integer	Maximum number of threads
SVRParam		varchar2(255)	Service description
SVRObjPerm		integer	Object permission

Table A-6 Services Table *(Continued)*

Name	Req	Type (Len)	Description
SVRModByGroup		varchar2(60)	ID of group modified by
SVRModByUser		varchar2(60)	ID of user modified by
SVRModDt		date	Modification date. Default: system date.

1. Primary key

DTServices

Table A-7 stores definitions of service lists, built from individual services stored in the Services table.

Table A-7 DT Services Table

Name	Req	Type (Len)	Description
DTSServiceListName [1]	Y	varchar2(60)	Service list name
DTSSeqNum [1, 2]	Y	integer	Order of the service in service list
DTSSchedType	N	Integer	Indicates whether service list is scheduled
DTSSndrMBName [2, 3]	Y	varchar2(60)	Sending member name
DTSRcvrMBName [2, 3]	Y	varchar2(60)	Receiving member name
DTSTypeName [2, 3]	Y	varchar2(60)	Service file type name OR service data object type name
DTSSVRId [3]		integer	Service ID
DTSSVRName		varchar2(60)	Service name
DTSServiceParams		varchar2(255)	Service parameters
DTSErrorHandler		varchar2(60)	Name of user-specified service for error handler
DTSDesc		varchar2(255)	Service description
DTSObjPerm		integer	Object permission
DTSModByGroup		varchar2(60)	ID of group modified by
DTSModByUser		varchar2(60)	ID of user modified by
DTSModDt		date	Modification date. Default: system date.

1. Primary key: *DTSServiceListName* + *DTSSeqNum*
2. Unique key: *DTSSeqNum* + *DTSSndrMBName* + *DTSRcvrMBName* + *DTSTypeName*
3. Foreign keys: *DTSTypeName* into *Services* (*SVRName*);
   DTSSVRId into *Services* (*SVRId*);
   DTSSndrMBName and *DTSRcvrMBName* into *Members* (*MBName*);

UniqueKeys

Table A-8 stores control information for all unique keys that are generated by the ECXpert System.

Table A-8 Unique Keys Table

Name	Req	Type (Len)	Description
UKName [1]	Y	varchar2(60)	Unique key name
UKLastValue		integer	Last value assigned to this key
UKModDt		date	Modification date. Default: system date.

1. Primary key

BlobInfo

Table A-9 stores blobs used by the ECXpert System.

Table A-9 BlobInfo Table

Name	Req	Type (Len)	Description
BLOBId [1]	Y	integer	Blob ID
BLOBType		integer	Kind of blob. Valid values are:
			0 = BTunknown
			1 = BTcertificate
			2 = BTsubject
			3 = BTtrackfile
			4 = BTjob
BLOBValue		long raw	Contents of blob
BLOBValueLen		integer	Length of contents

Table A-9 BlobInfo Table *(Continued)*

Name	Req	Type (Len)	Description
BLOBObjPerm		integer	Object permission
BLOBModByGroup		varchar2(60)	ID of group modified by
BLOBModByUser		varchar2(60)	ID of user modified by
BLOBModDt		date	Modification date. Default: system date.

1. Primary key

Membership-related Tables

The membership-related group of tables store information related to members in the ECXpert System.

Note that ECXpert supports the use of LDAP directory servers for storage of memebership data. Refer to Chapter 5 of the *iPlanet ECXpert Administrator's Guide* for information on migrating membership data from Oracle to the iPlanet Directory Server.

Members

Table A-10 stores the basic definitions of individual members within the ECXpert System.

Table A-10 Members Table

Name	Req	Type (Len)	Description
MBName [1]	Y	varchar2(60)	Member name.
MBType		integer	Member type. LDAP name: BusinessCategory Valid values: 0 = MBTunknown 1 = MBTsysAdmin 2 = MBTmembershipAdmin 3 = MBTgroupAdmin 4 = MBTinternalMember 5 = MBTtradingPartner (external member)
MBParentName [2]	Y	varchar2(60)	Member parent name

Table A-10 Members Table *(Continued)*

Name	Req	Type (Len)	Description
MBIsGroup		integer	Is member a group?
MBActive		integer	Is member active? LDAP name: EmployeeType, bit 0x01
MBPassword		varchar2(255)	Member password
MBPKPwd		varchar2(255)	(internal use) LDAP name: SeeAlso
MBInfoSource		varchar2(255)	LDAP name: LabeledURI
MBTrusted		integer	Is member trusted? LDAP name: EmployeeType, bit 0x02
MBOftpFlag		Integer	Indicates whether an ECX member is allowed to change password at beginning of OFTP session.
MBReadSpan		Integer	The number of days back that TradingXpert shows documents to this member in TradingXpert inbound and outbound document lists.
MBContactName		varchar2(60)	Member contact's name LDAP name: FullName
MBContactAddress1		varchar2(60)	Contact's address line 1 LDAP name: Address, bytes 0-59
MBContactAddress2		varchar2(60)	Contact's address line 2 LDAP name: Address, bytes 60-119
MBContactCity		varchar2(60)	Contact's city LDAP name: Locality
MBContactState		varchar2(60)	Contact's state or province LDAP name: State
MBContactZip		varchar2(60)	Contact's zip or postal code LDAP name: PostalCode
MBContactCountry		varchar2(60)	Contact's country LDAP name: Address, bytes 120-179
MBContactPhone		varchar2(60)	Contact's phone number LDAP name: PhoneNo
MBContactFax		varchar2(60)	Contact's fax number LDAP name: Fax

Table A-10 Members Table *(Continued)*

Name	Req	Type (Len)	Description
MBContactDesc		varchar2(255)	Contact's description LDAP name: Description
MBContactEmailId		varchar2(255)	Contact's email. LDAP name: Email
MBObjPerm		integer	Object permission
MBModByGroup		varchar2(60)	ID of group modified by
MBModByUser		varchar2(60)	ID of user modified by
MBModDt		date	Modification date. Default: system date.

1. Primary key
2. Foreign key: *MBParentName* into *Members* (MBName)

MBAddresses

Table A-11 stores trading addresses for members. Each member defined in Members table may have multiple trading addresses stored in MBAddresses table.

Table A-11 MBAddresses Table

Name	Req	Type (Len)	Description
MBAName [1]	Y	varchar2(60)	Member name
MBAQual [2, 3]	Y	varchar2(60)	Qualifier for trading address
MBAQualId [2, 3]	Y	varchar2(60)	Main trading address
MBAObjPerm		integer	Object permission
MBAModByGroup		varchar2(60)	ID of group modified by
MBAModByUser		varchar2(60)	ID of user modified by
MBAModDt		date	Modification date. Default: system date.

1. Foreign keys: MBAName into *Members* (*MBName*)
2. Primary key: MBAQual + MBAQualId
3. Unique key: MBAQual + MBAQualId

Partnership-related Tables

The partnership-related group of tables store information on trading partnerships.

Partnerships

Table A-12 stores the basic information defining a trading partnership.

Table A-12 Partnerships Table

Name	Req	Type (Len)	Description
PNId [1]P	Y	integer	Partnership ID
PNSndrMBName [2]		varchar2(60)	Sending member name
PNSndrQual [2, 3]	Y	varchar2(60)	Qualifier for sending member's trading address
PNSndrQualId [2, 3]	Y	varchar2(60)	Sending member's main trading address
PNRcvrMBName [2]		varchar2(60)	Receiving member name
PNRcvrQual [2, 3]	Y	varchar2(60)	Qualifier for receiving member's trading address
PNRcvrQualId [2, 3]	Y	varchar2(60)	Receiving member's main trading address
PNActive		integer	Is partnership active?
PNSndrCertType		integer	Certificate type. Valid values: 0 = CTUnknown 1 = CTSelf 2 = CTVerisignC3 3 = CTVerisignC2 4 = CTVerisignC1 5+ Other CA root(s) user imports
PNRcvrCertType		integer	Certificate type. Valid values: 0 = CTUnknown 1 = CTSelf 2 = CTVerisignC3 3 = CTVerisignC2 4 = CTVerisignC1 5+ Other CA root(s) user imports

Table A-12 Partnerships Table *(Continued)*

Name	Req	Type (Len)	Description
PNSecurity		integer	SMTP security. Valid values: 0 = Plain MIME (send as base64 encoding only) 1 = Encrypted (encrypted with receiver's public key) 2 = Signed (signed with sender's private key) 3 = SignedAndEncrypted (signed first, then encrypted)
PNGenOptEnv		integer	Enveloping Options: 0 = No UNA, No UNG 1 = UNA only 2 = UNG only 3 = UNA and UNG
PNGenIntgAckFlags		integer	Generate interchange acknowledgments flags (internal use)
PNIntgAckWait		integer	The number of minutes to wait before the acknowledgment becomes overdue. Default: 525600.
PNDesc		varchar2(255)	Partnership description
PNObjPerm		integer	Object permission
PNModByGroup		varchar2(60)	ID of group modified by
PNModByUser		varchar2(60)	ID of user modified by
PNModDt		date	Modification date. Default: system date.

1. Primary key
2. Foreign keys: *PNSndrMBName* into *Members* (*MBName*); *PNSndrQual* into *MBAddresses* (*MBAQual*); *PNSndrQualId* into *MBAddresses* (*MBAQualId*); *PNRcvrMBName* into *Members* (*MBName*); *PNRcvrQual* into *MBAddresses* (*MBAQual*); *PNRcvrQualId* into *MBAddresses* (*MBAQualId*)
3. Unique key: *PNSndrQual* + *PNSndrQualId* + *PNRcvrQual* + *PNRcvrQualId*

PNDocs

Table A-13 stores partnership document information.

Table A-13 PNDocs Table

Name	Req	Type (Len)	Description
PDPGId [1, 2]	Y	integer	Partnership ID
PDDocType [1]	Y	varchar2(60)	Document type
PDActive		integer	1 if active
PDPriority		integer	Processing priority. Valid values: 0 = PDunknown 1 = PDhigh 2 = PDmedium 3 = PDlow
PDAppDOTName		varchar2(60)	Application data object type name
PDMapName		varchar2(60)	Map file name Note: The Import utility does not verify if the specified map exists in the `maps` directory. If it does not exist, documents sent using the partnership will not be translated.
PDMapDirection		integer	Translation type. Valid values: 0 = XLTunknown 1= XLTinbound (EDI-to-Application) 2 = XLToutbound (Application-to-EDI) 3 = XLTedi2edi (EDI-to-EDI) 4 = XLTapp2app (Application-to-Application) 5 = XLTnoxlat (None; pass-through mode) 6 = XML2EDI (XML-to-EDI) 7 = XML2App (XML-to-Application/XML)
PDMapCommentSegId		varchar2 (8)	The segment ID used as “comment” type in the `Mercator` map. Default is `NTE`.

Table A-13 PNDocs Table *(Continued)*

Name	Req	Type (Len)	Description
PDAckExpected		integer	Is functional acknowledgment expected?
PDAckWait		integer	The number of minutes to wait before the acknowledgment becomes overdue. Default: 5259600.
PDLastCtrlNum		varchar2(60)	Last control number generated
PDLock		integer	(internal use)

Table A-13 PNDocs Table *(Continued)*

Name	Req	Type (Len)	Description
PD1stXportType		varchar2(60)	Primary transport protocol. Requires a delimiter of (|) or (;). Valid values include: “comm-ftp-geis” for GEIS FTP “commhttp-aiag” for HTTP AIAG “commhttp-gisb” for HTTP GISB “commhttp-ssl for obi” for HTTP SSL for OBI “commhttp-ssl for xml” for HTTP SSL for XML “commsmtp-send” for SMTP "ecxoftp-server" for Odette FTP (OFTP) “eXML-connector” for eXML Connector “ftp-local-application” for local FTP (application) “ftp-local-edi” for FTP (EDI) “http-retrieve” for HTTP Receive “legacy-mq-series” for Legacy Server (MQ Series) “legacy-oracle-apps” for Legacy Server (Oracle) “legacy-sap” for Legacy Server (SAP) “retrieve” for POLL “submit” for submit utility “TradingXpert” for sending to TradingXpert See Table A-14 on page 393 for the required parameters for each protocol.
PD1stXportParam		long	Primary transport protocol parameter See Table A-14 on page 393 for the required parameters for each transport protocol.

Table A-13 PNDocs Table *(Continued)*

Name	Req	Type (Len)	Description
PD2ndXportType		varchar2(60)	Alternate transport protocol. Valid values include: "submit" for submit utility "comm_ftp_geis" for GEIS FTP "ftp-local-application" for local FTP (application) "ftp-local-edi" for local FTP (EDI) "commhttp-aiag" for HTTP AIAG "commhttp-gisb" for HTTP GISB "commsmtp-receive-plain" for SMTP receive server (plain) "commsmtp-receive-smime" for SMTP receive server (S/MIME)
PD2ndXportParam		varchar2(255)	Alternate transport protocol parameter
PDSendType		integer	Immediate or scheduled
PDDeleteWait		integer	Retention period (days) before delete
PDArchiveWait		integer	Retention period (days) before archiving
PDPreEnveloped		integer	Is data pre-enveloped? Valid values: 0 = PEunknown 1 = PEenveloped (bundle preserves all envelopes) 2 = PEnonenveloped (bundle generates and/or replaces all envelopes) 3 = PEpreenvelopedEDI(not used) 4 = PEGetCtrlNo (Bundle only supplies the control number and preserves everything else in envelope) 5 = PEPreserveCtrlNo (Bundle only preserves the envelope control number)
PNPreCommSVRId		integer	Service ID of service to execute before sending to a communications agent

Table A-13 PNDocs Table *(Continued)*

Name	Req	Type (Len)	Description
PDDesc		varchar2(255)	Document description
PDObjPerm		integer	Object permission
PDModByGroup		varchar2(60)	ID of group modified by
PDModByUser		varchar2(60)	ID of user modified by
PDModDt		date	Modification date. Default: system date.

1. Primary key: *PDPGId* + *PDDocType*
2. Foreign keys: *PDPGId* into *PNStd* (*PSId*)

Partnership Transport Protocol Parameters

Table A-14 indicates the required parameters for each transport protocol value (PD1stXportType and PD2ndXportType fields).

Table A-14 Protocol Parameters Table

PD1stXportType or PD2ndXportType Value Delimiters: (|) or (;)	Corresponding PD1stXportParam and PD2ndXportParam Values Delimiter: (;)	Example of Control Structure Syntax for Imported Partnership Object

comm_ftp_geis	TP | *transport protocol;* (*comm_ftp_geis*)
	OO | *operation;* (*send*, *recv*)
	HN | *hostname;* (name or IP address)
	PT | *portnumber;* (Optional)
	UN | *username;*
	PW | *userPassword*
		`comm_ftp_geis,`
		`TP|comm_ftp_geis;OO|send;`
		`HN|hostname;PT|999;`
		`UN|username;`
		`PW|userpassword,`

Table A-14 Protocol Parameters Table *(Continued)*

PD1stXportType or PD2ndXportType Value	Corresponding PD1stXportParam and PD2ndXportParam Values	Example of Control Structure Syntax for Imported Partnership Object
Delimiters: (|) or (;)	**Delimiter: (;)**	

commhttp-aiag	HN | *hostname*; (name or IP address)
	PT | *portnumber*; (Optional)
	UN | *username*;
	OO | *operation*; (*DELIVER*, *OBTAIN*)
	PW | *password*;
	SS | *sender*;
	RR | *receiver*;
	RN | *reference number*; (Optional)
	AN | *application type*; (e.g., *EDI*, *application*, etc.)
	UP | *user parameter*; (Optional)
	PL | *login cgi-pathname*;
	PD | *deliver cgi-pathname* (if *OO=DELIVER*)
	PO | *obtain cgi-pathname* (if *OO=OBTAIN*)
		`commhttp-aiag,`
		`HN|hillary.mcom.com;PT|2000;`
		`UN|actraadm;OO|DELIVER;`
		`PW|actraadm;SS|user1;`
		`RR|user2;RN|1;AN|EDI;`
		`UP|actraadm;`
		`PL|/bin/aiag-logon;`
		`PD|/bin/aiag-deliver;PO|,`

commhttp-gisb	HN | *hostname*; (name or IP address) PT | *portnumber*; (Optional) UN | *username*; OO | *operation*; (*DELIVER*) PW | *password*; SS | *sender*; RR | *receiver*; IF | *input format*; (e.g., *EDI*, *application*, etc.) PD | *deliver cgi-pathname*	*commhttp-gisb,* *HN|hillary.mcom.com;PT|2000;* *UN|actraadm;OO|DELIVER;* *PW|actraadm;SS|user1;* *RR|user2;IF|EDI;* *PD|/bin/gisb-deliver,*

Table A-14 Protocol Parameters Table *(Continued)*

PD1stXportType or PD2ndXportType Value Delimiters: (|) or (;)	Corresponding PD1stXportParam and PD2ndXportParam Values Delimiter: (;)	Example of Control Structure Syntax for Imported Partnership Object

commsmtp-send	Note: Only MT is string; all other parameters are numbers.
	
	PR|*ProcessMethod*;
	(0=SimpleMime,
	1=EncryptedOnly,
	2=SignedOnly,
	3=Signed&Encrypted)
	
	MA|*MIC_Algorithm*; (If *PR=1*, *2*, or *3*)
	(*28*=SHA_1, *5*=MD5)
	
	MR|*MDN_Requested*;
	(*0*=No MDN, *1*=Plain MDN,
	2=Signed MDN)
	
	MT|*Mime_subtype*; (Optional)
	
	KL|*Key_length*; (If *PR=1*, *2*, or *3*)
	(*56*, *64*, *75*, *128*, *255*, *512*, *1024*)
	
	CS|*senderCertType*;
	(*0*=CTUnknown, *1*=CTSelf,
	2=CTVerisignC3,
	3=CTVerisignC2,
	4=CTVerisignC1, *5*=Other CA root(s) user imports)
	
	CR|*receiverCertType*
	(see values for CS)
		`commsmtp-send,PR|0;MA|28;MR|1;`
		`MT|application;KL|128;`
		`CS|;CR|,`

Table A-14 Protocol Parameters Table *(Continued)*

PD1stXportType or PD2ndXportType Value	Corresponding PD1stXportParam and PD2ndXportParam Values	Example of Control Structure Syntax for Imported Partnership Object
Delimiters: (|) or (;)	**Delimiter: (;)**	

ecxoftp-server	OU | *username;*
	OL | *userPassword;*
	OX | *transportMethod;* (*X.25, X.28,* or *TCP/IP*)
	XN | *destination_X.121Address*; (Optional; if *OX*=X.25, defaults to local network user address)
	XL | *LogicalChannelNumber*; (Optional; numeric)
	XC | *CallUserData* (Optional; numeric {typically hex})
	XF | *X.25 FacilityInformation*; (Optional; numeric {typically hex})
	XR | *RoutingEntry*; (Optional)
	XT | *X.28 Tel number;* (Optional)
	XS | *X.28 Connection script pathname;*
	XU | *X.28 PAD user name;* (Optional)
	XY | *X.28 PAD user password;* (Optional)
	XZ | *X.28 destination NUA;* (Optional; numeric network user address)
	TX | *TCP/IP destination port;* (Optional for *OX*=TCP/IP; defaults to 3305)
	TH | *TCP/IP destination host;* (Optional for *OX*=TCP/IP; name or IPaddress; defaults to local host name)
		`TCP/IP:`
		`ecxoftp-server,OU|actraadm;`
		`OL|actraadm;OX|TCP/IP;`
		`XN|;XL|;XC|;XF|;XR|;XT|;XS|;`
		`XU|;XY|;XZ|;TX|9999;`
		`TH|destination.mcom.com,`
		`X.25:`
		`ecxoftp-server,OU|actraadm;`
		`OL|actraadm;OX|X.25;`
		`XN|123456789;XL|;XC|1234;`
		`XF|;XR|;XT|;XS|;XU|;XY|;XZ|;`
		`TX|;TH|,`
		`X.28:`
		`ecxoftp-server,OU|actraadm;`
		`OL|actraadm;OX|X.28;XN|;`
		`XL|;XC|;XF|;XR|;XT|;`
		`XS|/tmp/conn_script;XU|;`
		`XY|;XZ|;TX|;TH|,`

Table A-14 Protocol Parameters Table *(Continued)*

PD1stXportType or PD2ndXportType Value Delimiters: (|) or (;)	Corresponding PD1stXportParam and PD2ndXportParam Values Delimiter: (;)	Example of Control Structure Syntax for Imported Partnership Object

eXML-connector	HN | *hostname*; (name or IP address)
	PT | *portnumber*; (Optional)
	AI | *informationFilePath;* (file path and file name)
	XT | *fileTransport*; (*file*—to transmit filename only, *stream*—to transmit entire file)
		`eXML-connector,`
		`HN|hillary.mcom.com;PN|555;`
		`AI|/tmp/infofile.dat;`
		`XT|file,`

ftp-local-application	TP | *ftp-local-application*;
	HN | *hostname*; (name or IP address)
	PT | *portnumber*; (Optional)
	UN | *username*;
	PW | *userPassword*;
	SS | *SendPattern*;
	RS | *ReceivePattern*;
	OD | *outboundDirectory*;
	ID | *InboundDirectory*;
	IT | *InboundFileType*;
	MD | *transfer mode* ("*A*" for ASCII, "*I*" for Binary)
		`ftp-local-application,`
		`TP|ftp-local-application;`
		`HN|hostname;PT|;`
		`UN|username;PW|userpassword;`
		`SS|test1;RS|;OD|/tmp;`
		`ID|;IT|;MD|I,`

Table A-14 Protocol Parameters Table *(Continued)*

PD1stXportType or PD2ndXportType Value	Corresponding PD1stXportParam and PD2ndXportParam Values	Example of Control Structure Syntax for Imported Partnership Object
Delimiters: (|) or (;)	**Delimiter: (;)**	

ftp-local-edi	TP | *ftp-local-application*;
	HN | *hostname*; (name or IP address)
	PT | *portnumber*; (Optional)
	UN | *username*;
	PW | *userPassword*;
	SS | *SendPattern*;
	RS | *ReceivePattern*;
	OD | *outboundDirectory*;
	ID | *InboundDirectory*;
	IT | *InboundFileType*;
	MD | *transfer mode* ("*A*" for ASCII, "*I*" for Binary)
		`ftp-local-edi,`
		`TP|ftp-local-application;`
		`HN|hostname;PT|;`
		`UN|username;PW|userpassword;`
		`SS|test1;RS|;OD|/tmp;`
		`ID|;IT|;MD|I,`

commhttp-ssl	HN | *hostname*; (name or IP address)
	PT | *portnumber*; (Optional)
	PN | *cgi pathname*;
	SE | *sender*;
	PW | *password*;
	RE | *receiver*;
	FT | *file type*;
	CY | *certificate type*
		`commhttp-ssl (for OBI)`
		`,HN|hostname;`
		`PT|portname;`
		`PN|cgiPathname;`
		`SE|sender;`
		`FT|filetype;`
		`CY|certificateType,`
		
		Example: `commhttp-ssl,`
		`HN|hostname;PT|999;PN|cgiPathn`
		`ame;SE|sender;FT|filetype;CY|c`
		`ertificateType`
		`/`

Table A-14 Protocol Parameters Table *(Continued)*

PD1stXportType or PD2ndXportType Value Delimiters: (|) or (;)	Corresponding PD1stXportParam and PD2ndXportParam Values Delimiter: (;)	Example of Control Structure Syntax for Imported Partnership Object

commhttp-ssl-XML	HN | *hostname*; (name or IP address)
	PT | *portnumber*; (Optional)
	PN | *cgi pathname*;
	SE | *sender*;
	PW | *password*;
	RE | *receiver*;
	FT | *file type*;
	CT | *content type*
	CY | *certificate type*
		`commhttp-ssl for OBI`
		`,HN|hostname;`
		`PT|portname;`
		`PN|cgiPathname;`
		`SE|sender;`
		`FT|filetype;`
		`CY|certificateType,`
		`CT|contentType`
		
		Example: `commhttp-ssl, HN|hostname;PT|999;PN|cgiPathname;SE|sender;FT|filetype;CT|contentType;CY|certificateType`
		`/`

http-retrieve	—	`http-retrieve,,`

legacy-mq-series	QN | *QUEUENAME*; (uppercase)
	QM | *queue manager*;
	MH | *message header file*
		`legacy-mq-series,`
		`QN|TESTQ;QM|testqmgr;`
		`MH|/tmp/msgheadr.txt,`

legacy-oracle-apps	MN | *map name*;
	DB | *database name*;
	UN | *username*;
	PW | *password*
		`legacy-oracle-apps,`
		`MN|xobi850.sun;DB|orafindb;`
		`UN|apps;PW|apps,`

legacy-sap	CN | *client number*; UI | *user id*; PW | *password*	`legacy-sap,CN|800;UI|scott;` `PW|scott,`
retrieve	—	`retrieve,,`

PNCard

Table A-15 stores information about the Mercator input and output cards associated with a partnership document.

Table A-15 PNCard Table

Name	Req	Type (Len)	Description
PDDPGId [1, 2]	Y	integer	Partnership ID
PDDDocType [1, 2]	Y	varchar2(60)	Document type
PDDCardNum [1]	Y	integer	Card number
PDDSndrMBName		varchar2(60)	Sending member name
PDDRcvrMBName		varchar2(60)	Receiving member name
PDDCardDocType		varchar2(60)	Card document type
PDDObjPerm		integer	Object permission
PDDModByGroup		varchar2(60)	ID of group modified by
PDDModByUser		varchar2(60)	ID of user modified by
PDDModDt		date	Modification date. Default: system date.

1. Primary key: *PDDPGId + PDDDocType + PDDCardNum*
2. Foreign keys: PDDPGId into PNDocs; PDDDocType into PNDocs

PNGroup

Table A-16 stores information on expected document groups, especially of the GS/GE and UNG/UNE segments, of incoming files for a given partnership.

Table A-16 PNGroup Table

Name	Req	Type (Len)	Description
PGId	Y	integer	Unique ID number of partnership group.
PGPSId [1, 2]	Y	integer	Standard associated with partnership group.
PGGroupType [1]	Y	varchar2(60)	Partnership group
PGSndrQual [1]	Y	varchar2(60)	Qualifier for the application sender code. Used only in EDIFACT.

Table A-16 PNGroup Table *(Continued)*

Name	Req	Type (Len)	Description
PGSndrAppCode	Y	varchar2(60)	Application sender code.
PGRcvrQual	Y	varchar2(60)	Qualifier for the application receiver code.
PGRcvrAppCode	Y	varchar2(60)	Application receiver code.
PGLastGroupCtrlNum		varchar2(60)	Last group control number generated
PGLockGroup		integer	(internal use)
PGGenDocAck		integer	Generate document acknowledgments flags (internal use)
PGGrpAckWait		integer	The number of minutes to wait before the acknowledgment becomes overdue. Default: 525600.
PGObjPerm		integer	Object permission
PGModByGroup		varchar2(60)	ID of group modified by
PGModByUser		varchar2(60)	ID of user modified by
PGModDt		date	Modification date. Default: system date.

1. Primary key: *PGPSId + PGGroupType + PGSndrQual + PGSndrAppCode + PGRcvrQual + PGRcvrAppCode*
2. Foreign key: *PGPSId* into *PNStd* (*PSId*)

PNStd

Table A-17 stores EDI standard information for a partnership defined in the Partnership table.

Table A-17 PNStd Table

Name	Req	Type (Len)	Description
PSId [1]	Y	integer	Standards ID
PSPNId [2, 3]	Y	integer	Partnership ID
PSStandard [2]	Y	varchar2(60)	EDI standard
PSVersion [2]	Y	varchar2(60)	EDI standard version number
PSRelease [2]	Y	varchar2(60)	EDI standard release number
PSLastIntgCtrlNum		varchar2(60)	Last interchange control number generated

Table A-17 PNStd Table *(Continued)*

Name	Req	Type (Len)	Description
PSLockIntg		integer	(internal use)
PSGenOptEnv		integer	Generate an envelope?
PSGenIntgAckFlag		integer	Generate an acknowledgment upon receipt of a document?
PSIntgAckWait		integer	Time (in seconds) after document receipt that an acknowledgment must be generated and sent
PSTestProdFlag		integer	Test vs. production data flag. Valid values: 0 = TPFunknown 1 = TPFproduction (production data) 2 = TPFtest (test data)
PSSegTerm		varchar2(6)	Segment terminator character
PSElmtSep		varchar2(6)	Data element separator character
PSSubElmtSep		varchar2(6)	Data sub-element separator character
PSDecPtChar		varchar2(6)	Decimal point character
PSRelChar		varchar2(6)	Release character
PSOutStandard		varchar2(60)	Interchange standard user wishes to appear in bundled EDI documents
PSOutVersion		varchar2(60)	Interchange version user wishes to appear in bundled EDI documents
PSOutRelease		varchar2(60)	Interchange release user wishes to appear in bundled EDI documents
PSObjPerm		integer	Object permission
PSModByGroup		varchar2(60)	ID of group modified by
PSModByUser		varchar2(60)	ID of user modified by
PSModDt		date	Modification date. Default: system date.

1. Primary key: *PSId*
2. Unique key: *PSPNId* + *PSStandard* + *PSVersion* + *PSRelease*
3. Foreign key: *PSPNId* into *Partnerships* (*PNId*)

Certificate-related Tables

The certificate-related group of tables store information supporting public key encryption in the ECXpert System.

Certificates

Table A-18 stores information on certificates.

Table A-18 Certificates Table

Name	Req	Type (Len)	Description
CRTDigest [1, 2]	Y	varchar2(60)	Certificate issuer name and serial number digest
CRTCertType [1, 2]	Y	integer	Certificate type. Valid values: 0 = CTUnknown 1 = CTSelf 2 = CTVerisignC3 3 = CTVerisignC2 4 = CTVerisignC1 5+ Other CA root(s) user imports
CRTCertUsage	Y	integer	Indicates how the certificate is being used (i.e. to digitally sign, encrypt, or both)
CRTSubjectDigest	Y	varchar2(60)	Subject named digest
CRTPublicKeyDigest [3]	Y	varchar2(30)	Public key digest
CRTBlobId		integer	(internal use)
CRTSubjectBlobId		integer	(internal use)
CRTExpireDt	Y	integer	Certificate expiration date
CRTName		varchar2(60)	Name of issuing certificate authority
CRTIsRoot		integer	Indicates if certificate is a root certificate
CRTMBName [2, 3]	Y	varchar2(60)	Member name
CRTMBEmailId		varchar2(60)	Member's e-mail address
CRTDesc		varchar2(255)	Certificate description
CRTObjPerm		integer	Object permission
CRTModByGroup		varchar2(60)	ID of group modified by

Table A-18 Certificates Table *(Continued)*

Name	Req	Type (Len)	Description
CRTModByUser		varchar2(60)	ID of user modified by
CRTModDt		date	Modification date. Default: system date.

1. Primary key: *CRTDigest* + *CRTCertType*
2. Unique key: *CRTCertType* + *CRTDigest* + *CRTMBName*
3. Foreign key: *CRTPublicKeyDigest* into *KeyPairs* (*KPDigest*); *CRTMBName* into *Members* (*MBName*)

CRL

Table A-19 stores the certificate revocation list.

Table A-19 CRL Table

Name	Req	Type (Len)	Description
CRLIssuerDigest [1]	Y	varchar2(60)	Certificate issuer digest
CRLTime		integer	Time stamp
CRLValue		long raw	Certificate revolution list
CRLValueLen		integer	Length of certificate (bytes)
CRLDesc		varchar2(255)	Description
CRLObjPerm		integer	Object permission
CRLModByGroup		varchar2(60)	ID of group modified by
CRLModByUser		varchar2(60)	ID of user modified by
CRLModDt		date	Modification date. Default: system date.

1. Primary key

CertTypeInfo

Table A-20 stores information on certificates for display through the user interface.

Table A-20 CertTypeInfo Table

Name	Req	Type (Len)	Description
CTICertType [1]	Y	integer	Certificate type. Valid values: 0 = CTUnknown 1 = CTSelf 2 = CTVerisignC3 3 = CTVerisignC2 4 = CTVerisignC1 5+ Other CA root(s) user imports
CTICertTypeName	Y	varchar2(60)	Name of certificate authority
CTICertTypeDesc		varchar2(60)	Certificate authority description
CTIObjPerm		integer	Object permission
CTIModByGroup		varchar2(60)	ID of group modified by
CTIModByUser		varchar2(60)	ID of user modified by
CTIModDt		date	Modification date. Default: system date.

1. Primary key

KeyPairs

Table A-21 stores public/private key pair information.

Table A-21 KeyPairs Table

Name	Req	Type (Len)	Description
KPDigest [1, 2]	Y	varchar2(30)	Public key digest
KPPrivateKey		long raw	Private key (encrypted)
KPPrivateKeyLen		integer	Private key length
KPDesc		varchar2(255)	Key pair description
KPObjPerm		integer	Object permission
KPModByGroup		varchar2(60)	ID of group modified by
KPModByUser		varchar2(60)	ID of user modified by

Table A-21 KeyPairs Table

Name	Req	Type (Len)	Description
KPModDt		date	Modification date. Default: system date.

1. Primary key
2. Foreign key: *KPDigest* into *Certificates* (*CRTPublicKeyDigest*)

Tracking-related Tables

The tracking-related group of tables supports document tracking in the ECXpert System.

Tracking

Table A-22 stores information associated with tracking IDs.

Table A-22 Tracking Table

Name	Req	Type (Len)	Description
TRKId [1]	Y	integer	Tracking ID - ECXpert internal tracking number for the submission unit
TRKServiceListName		varchar2(60)	Service list name associated with tracking ID
TRKDOTName		varchar2(60)	Data object type name (e.g. the document type specified in the partnership)
TRKSndrMBName		varchar2(60)	Sending member name
TRKRcvrMBName		varchar2(60)	Receiving member name
TRKCurServiceIdx		integer	Current service index, , which indicates which service is currently running: either 1 for submission or the offset corresponding to the service in the service list -1
TRKCurServiceName		varchar2(60)	Current service name
TRKCurServiceParamFile		varchar2(255)	Stores the custom service parameter file name

Table A-22 Tracking Table *(Continued)*

Name	Req	Type (Len)	Description
TRKPrimaryState		integer	Primary tracking state. Valid values: 0 = TSunknown - indicates NULL value 1 = TSready - indicates service has yet to be invoked 2 = TSinProgress - indicates service has been invoked 3 = TSdoneOK - indicates service is done with no errors 4 = TSdoneBad - indicates service is done with errors 5 = TSalldoneOK - indicates last service on service list is done and TRKState is TSdoneOK 6 = TSbundled - identifies bundle generated trackings
TRKState		integer	Tracking state. Valid values: 0 = TSunknown - indicates NULL value 1 = TSready - indicates service has yet to be invoked 2 = TSinProgress - indicates service has been invoked 3 = TSdoneOK - indicates service is done with no errors 4 = TSdoneBad - indicates service is done with errors 5 = TSalldoneOK - indicates last service on service list is done and TRKState is TSdoneOK 6 = TSbundled - identifies bundle generated trackings
TRKErrnum		integer	Tracking error number. Default: 0.
TRKPriority		integer	Processing priority. Valid values: 0 = PDunknown 1 = PDhigh 2 = PDmedium 3 = PDlow
TRKCreationDt		date	Tracking ID creation date. Default: system date.
TRKLock		integer	Is Tracking ID locked?

Table A-22 Tracking Table *(Continued)*

Name	Req	Type (Len)	Description
TRKMapDirection		integer	Translation type. Valid values: 0 = XLTunknown 1= XLTinbound (EDI-to-Application) 2 = XLToutbound (Application-to-EDI) 3 = XLTedi2edi (EDI-to-EDI) 4 = XLTapp2app (Application-to-Application) 5 = XLTnoxlat (None; pass-through mode)
TRKXportType		varchar2(60)	Transport protocol. Valid values include: "submit" for submit utility "comm_ftp_geis" for GEIS FTP "ftp-local-application" for local FTP (application) "ftp-local-edi" for local FTP (EDI) "commhttp-aiag" for HTTP AIAG "commhttp-gisb" for HTTP GISB "commsmtp-receive-plain" for SMTP receive server (plain) "commsmtp-receive-smime" for SMTP receive server (S/MIME)
TRKMDNState		integer	Message disposition notification state. Valid values: 0 = MSunknown 1 = MSready (MDN generated and ready to send) 2 = MSsent (MDN is sent) 3 = MSwaiting (email is sent and is waiting for an incoming MDN) 4 = MSreconciled (Received MDN and reconciled it with original email)

Table A-22 Tracking Table *(Continued)*

Name	Req	Type (Len)	Description
TRKMDNOverDueDt		date	The date after which the message disposition notification becomes over due. Default: system date + 3652.
TRKExtReference		varchar2(60)	External reference
TRKExtPathName		varchar2(255)	External pathname
TRKPartNum		integer	Current part number
TRKPartTotal		integer	Part total
TRKPartType		integer	Attachment or parts
TRKCustomInfo		varchar2(255)	(internal use)
TRKMisc		varchar2(255)	(internal use)
TRKFullPathName		varchar2(255)	Full path name
TRKSize		integer	Submission Unit file size (bytes)
TRKBlobId		integer	(internal use)
TRKObjPerm		integer	Object permission
TRKModByGroup		varchar2(60)	ID of group modified by
TRKModByUser		varchar2(60)	ID of user modified by
TRKModDt		date	Modification date. Default: system date.

1. Primary key

TrkIntchg

Table A-23 stores information on the interchange level of the EDI envelope.

Table A-23 TrkIntchg Table

Name	Req	Type (Len)	Description
TITrkId [1, 2]	Y	integer	ECXpert internal tracking number for the submission unit
TIId [1]	Y	integer	Interchange identifier
TICurServiceIdx		integer	Current service index

Table A-23 TrkIntchg Table *(Continued)*

Name	Req	Type (Len)	Description
TIState		integer	Tracking state. Valid values: 0 = TSunknown 1 = TSready 2 = TSinProgress 3 = TSdoneOK 4 = TSdoneBad 5 = TSalldoneOK 6 = TSbundled
TIErrnum		integer	Interchange error number. Default: 0.
TIParseErrnum		integer	(internal use)
TIPriority		integer	Processing priority. Valid values: 0 = PDunknown 1 = PDhigh 2 = PDmedium 3 = PDlow
TIPSId		integer	Partnership ID
TISndrQual		varchar2(60)	Sending member qualifier for trading address
TISndrQualId		varchar2(60)	Sending member main trading address
TIRcvrQual		varchar2(60)	Receiving member qualifier for trading address
TIRcvrQualId		varchar2(60)	Receiving member main trading address
TIStandard		varchar2(60)	EDI standard used
TIVersion		varchar2(60)	Version number of EDI standard used
TIRelease		varchar2(60)	Release number of EDI standard used
TITestProdFlag		integer	Test vs. production data flag. Valid values: 0 = TPFunknown 1 = TPFproduction (production data) 2 = TPFtest (test data)

Table A-23 TrkIntchg Table *(Continued)*

Name	Req	Type (Len)	Description
TIAckState		integer	Functional acknowledgment state. A single value is computed by adding the following component values as events occur: 0 = ASunknown 1 = ASwaiting 2 = ASok 4 = ASerror 8 = ASreject 16 = ASpreject 32 = ASsent 64 = ASsendFailed 128 = ASreconciled For detailed breakdown of actual values and messages displayed, see "Values of AckState" on page 371.
TIAckOverDueDt		date	The number of minutes to wait before the acknowledgment becomes overdue. Default: system date + 3652.
TIGenIntgAckFlags		integer	Generate functional acknowledgment? Valid values: 0 = GAunknown 1 = GAnoack (none) 2 = GAgroup (group level) 3 = GAdoc (document level)
TICtrlNum		varchar2(60)	EDI standard control number, determined by trading partner relationship
TIFileName		varchar2(255)	Submission Unit file name
TICreationDt		date	Submission Unit file creation date
TISize		integer	Interchange size (bytes)
TIHdrStartOff		integer	Interchange header start offset (bytes)
TIHdrSize		integer	Interchange header size (bytes)
TITlrStartOff		integer	Interchange trailer start offset (bytes)
TITlrSize		integer	Interchange trailer size (bytes)
TISegTerm		varchar2(6)	Segment terminator
TIElmtSep		varchar2(6)	Element separator
TISubElmtSep		varchar2(6)	Sub-element separator

Table A-23 TrkIntchg Table *(Continued)*

Name	Req	Type (Len)	Description
TIDecPtChar		varchar2(6)	Decimal point character
TIRelChar		varchar2(6)	Release character
TIObjPerm		integer	Object permission
TIModByGroup		varchar2(60)	ID of group modified by
TIModByUser		varchar2(60)	ID of user modified by
TIModDt		date	Modification date. Default: system date.

1. Primary key: *TITrkId* + *TIId*
2. Foreign key: *TITrkId* into *Tracking* (*TRKId*)

TrkGroup

Table A-24 stores information on the group level of the EDI envelope.

Table A-24 TrkGroup Table

Name	Req	Type (Len)	Description
TGTrkId [1, 2]	Y	integer	BDG internal tracking number for the submission unit
TGIntgId [1]	Y	integer	Interchange identifier
TGId [1]	Y	integer	Group identifier
TGType		varchar2(10)	Group document type.
TGCurServiceIdx		integer	Current service index
TGState		integer	Tracking state. Valid values: 0 = TSunknown 1 = TSready 2 = TSinProgress 3 = TSdoneOK 4 = TSdoneBad 5 = TSalldoneOK 6 = TSbundled
TGErrnum		integer	Tracking error number. Default: 0.
TGParseErrnum		integer	Parse error number. Default: 0.

Table A-24 TrkGroup Table *(Continued)*

Name	Req	Type (Len)	Description
TGPriority		integer	Processing priority. Valid values: 0 = PDunknown 1 = PDhigh 2 = PDmedium 3 = PDlow
TGSndrQual		varchar2(60)	Sending member main trading address
TGSndrAppCode		varchar2(60)	Application sender code.
TGRcvrQual		varchar2(60)	Receiving member main trading address
TGRcvrAppCode		varchar2(60)	Application receiver code.
TGStandard		varchar2(60)	EDI standard used
TGVersion		varchar2(60)	Version number of EDI standard used
TGRelease		varchar2(60)	EDI standard release number
TGCtrlNum		varchar2(60)	EDI standard control number, determined by trading partner relationship
TGIncludedSets		integer	The number of transaction sets (documents) included in the group.
TGAckState		integer	Functional acknowledgment state. A single value is computed by adding the following component values as events occur: 0 = ASunknown 1 = ASwaiting 2 = ASok 4 = ASerror 8 = ASreject 16 = ASpreject 32 = ASsent 64 = ASsendFailed 128 = ASreconciled For detailed breakdown of actual values and messages displayed, see "Values of AckState" on page 371.
TGAckOverDueDt		date	The number of minutes to wait before the acknowledgment becomes overdue. Default: system date + 3652.
TGCreationDt		date	Submission Unit file creation date. Default: system date.
TGSize		integer	Submission Unit file size

Table A-24 TrkGroup Table *(Continued)*

Name	Req	Type (Len)	Description
TGHdrStartOff		integer	Header start offset (bytes)
TGHdrSize		integer	Header size (bytes)
TGTlrStartOff		integer	Trailer start offset (bytes)
TGTlrSize		integer	Trailer size (bytes)
TGObjPerm		integer	Object permission
TGModByGroup		varchar2(60)	ID of group modified by
TGModByUser		varchar2(60)	ID of user modified by
TGModDt		date	Modification date. Default: system date.

1. Primary key: *TGTrkId* + *TGIntgId* + *TGId*
2. Foreign key: *TGTrkId* into *Tracking* (*TRKId*); *TGIntgId* into *TrkIngchg* (*TITrkId*)

TrkDoc

Table A-25 stores information on the document level of the EDI envelope.

Table A-25 TrkDoc Table

Name	Req	Type (Len)	Description
TDId[1]	Y	varchar(30)	Document-level internal tracking ID
TDTrkId [2]		integer	Tracking ID - internal tracking number for the submission unit
TDIntgId		integer	Interchange identifier
TDGrpId		integer	Group identifier
TDDocId		integer	Document identifier
TDCurServiceIdx		integer	Current service index
TDCurServiceName		varchar2(60)	Current service name

Table A-25 TrkDoc Table *(Continued)*

Name	Req	Type (Len)	Description
TDState		integer	Tracking state. Valid values: 0 = TSunknown 1 = TSready 2 = TSinProgress 3 = TSdoneOK 4 = TSdoneBad 5 = TSalldoneOK 6 = TSbundled
TDErrnum		integer	Tracking error number. Default: 0.
TDParseErrnum		integer	Parse error number. Default: 0.
TDXlatState		integer	Translation state
TDXlatErrnum		integer	Translation error number. Default: 0.
TDPriority		integer	Processing priority. Valid values: 0 = PDunknown 1 = PDhigh 2 = PDmedium 3 = PDlow
TDStartXlatDt		date	Date translation started. Default: system date.
TDEndXlatDt		date	Date translation ended. Default: system date.
TDLock		integer	(internal use)
TDPSId		integer	Partnership standard ID
TDDocType		varchar2(60)	Document type
TDTestProdFlag		integer	Test vs. production data flag. Valid values: 0 = TPFunknown 1 = TPFproduction (production data) 2 = TPFtest (test data)
TDSndrMBName		varchar2(60)	Sender member's name
TDSndrQual		varchar2(60)	Sender EDI qualifier
TDSndrQualId		varchar2(60)	Sender EDI qualifier ID
TDRcvrMBName		varchar2(60)	Receiver member's name
TDRcvrQual		varchar2(60)	Receiver EDI qualifier
TDRcvrQualId		varchar2(60)	Receiver EDI qualifier ID

Table A-25 TrkDoc Table *(Continued)*

Name	Req	Type (Len)	Description
TDSndrAppQual		varchar2(60)	Qualifier for the application sender code. Used only in EDIFACT.
TDSndrAppCode		varchar2(60)	Application sender code.
TDRcvrAppQual		varchar2(60)	Qualifier for the application receiver code.
TDRcvrAppCode		varchar2(60)	Application receiver code. Provides for defining trading partnerships at the functional group level.
TDMapName		varchar2(60)	Name of map for translation
TDStandard		varchar2(60)	EDI standard for translation
TDVersion		varchar2(60)	Version of EDI standard
TDRelease		varchar2(60)	Release of EDI standard
TDMapDirection		integer	Translation type. Valid values: 0 = XLTunknown 1= XLTinbound (EDI-to-Application) 2 = XLToutbound (Application-to-EDI) 3 = XLTedi2edi (EDI-to-EDI) 4 = XLTapp2app (Application-to-Application) 5 = XLTnoxlat (None; pass-through mode)

Table A-25 TrkDoc Table *(Continued)*

Name	Req	Type (Len)	Description
TD1stXportType		varchar2(60)	Primary transport protocol. Valid values include: "submit" for submit utility "comm_ftp_geis" for GEIS FTP "ftp-local-application" for local FTP (application) "ftp-local-edi" for local FTP (EDI) "commhttp-aiag" for HTTP AIAG "commhttp-gisb" for HTTP GISB "commsmtp-receive-plain" for SMTP receive server (plain) "commsmtp-receive-smime" for SMTP receive server (S/MIME)
TD1stXportParam		long	Transport parameter
TD2ndXportType		varchar2(60)	Alternate transport protocol. Valid values include: "submit" for submit utility "comm_ftp_geis" for GEIS FTP "ftp-local-application" for local FTP (application) "ftp-local-edi" for local FTP (EDI) "commhttp-aiag" for HTTP AIAG "commhttp-gisb" for HTTP GISB "commsmtp-receive-plain" for SMTP receive server (plain) "commsmtp-receive-smime" for SMTP receive server (S/MIME)
TD2ndXportParam		varchar2(255)	Transport parameter
TDSendType		integer	Sender type: immediate or scheduled
TDSourceDocId		char(30)	Source document ID
TDAckDocId		char(30)	Functional acknowledgment document ID

Table A-25 TrkDoc Table *(Continued)*

Name	Req	Type (Len)	Description
TDAckState		integer	Functional acknowledgment state. A single value is computed by adding the following component values as events occur: 0 = ASunknown 1 = ASwaiting 2 = ASok 4 = ASerror 8 = ASreject 16 = ASpreject 32 = ASsent 64 = ASsendFailed 128 = ASreconciled For detailed breakdown of actual values and messages displayed, see "Values of AckState" on page 371.
TDAckOverDueDt		date	The number of minutes to wait before the acknowledgment becomes overdue. Default: system date + 3652.
TDCreationDt		date	Document creation date. Default: system date + 3652.
TDCtrlNum		varchar2(60)	Control number
TDMapRestrictFlags		integer	Map restriction flags
TDFileName		varchar2(255)	Map file name
TDSize		integer	Document size (bytes)
TDHdrStartOff		integer	Document header start offset (bytes)
TDHdrSize		integer	Document header size (bytes)
TDTlrStartOff		integer	Document trailer start offset (bytes)
TDTlrSize		integer	Document trailer size (bytes)
TDUserLink1Name		varchar2(60)	User link 1 name
TDUserLink1Value		varchar2(60)	User link 1 value
TDUserLink2Name		varchar2(60)	User link 2 name
TDUserLink2Value		varchar2(60)	User link 2 value
TDArchiveWait		integer	Wait time to archive
TDDeleteWait		integer	Wait time to delete

Table A-25 TrkDoc Table *(Continued)*

Name	Req	Type (Len)	Description
TDDataState		integer	Data state. Valid values: 0 = DSunknown 1 = DSreadyForPurge 2 = DSpurged 3 = DSreadyForArchive 4 = DSarchived 5 = DSreadyForRestore 6 = DSrestored
TDPreEnveloped		integer	Is data pre-enveloped? Valid values: 0 = PEunknown 1 = PEenveloped (bundle preserves all envelopes) 2 = PEnonenveloped (bundle generates and/or replaces all envelopes) 3 = PEpreenvelopedEDI(not used) 4 = PEGetCtrlNo (Bundle only supplies the control number and preserves everything else in envelope) 5 = PEPreserveCtrlNo (Bundle only preserves the envelope control number) 6 = PEFill (Bundle fills in the missing envelope information - not used in this release)
TDBundleState		integer	Bundle state. Valid values: 0 = BSunknown 1 = BSreadyForBundle 2 = BSbundleed 3 = BSdeliveredToComm 4 = BSsecondarySubmitted 5 = BSsecondaryError
TDBundleTrkId		integer	Bundle tracking ID
TDPreCommSVRId		integer	Service ID of service to execute before sending to a communications agent
TDObjPerm		integer	Object permission
TDModByGroup		varchar2(60)	ID of group modified by
TDModByUser		varchar2(60)	ID of user modified by
TDModDt		date	Modification date. Default: system date.

1. Primary key

2. Foreign key: *TDTrkId* into *Tracking* (*TRKId*)

TrkSegment

Table A-26 stores document segment-level information.

Table A-26 TrkSegment Table

Name	Req	Type (Len)	Description
TSDocId	Y	varchar (30)	Document-level internal tracking ID
TSSegmentId	Y	varchar (30)	ID of segment within document
TSSegmentPosition		integer	Segment sequence number within the document
TSSegmentErrnum		integer	EDI error code. Valid values: 2 = unexpected segment 3 = mandatory segment missing 8 = segment has data elements in error
TSElementPosition		integer	Data element sequence number within the segment (only used when error is in data element)
TSElementCopy		varchar2 (128)	Copy of data elment data (only used when error is in data element)

TrkDocDetails

Table A-27 stores document card-level information. [Jean: the pseudo-footnotes in the following table had 2 footnotes labeld F and none labled U. pz]

Table A-27 TrkDocDetails Table

Name	Req	Type (Len)	Description
TDDId [1, 2]	Y	varchar(30)	Detail ID
TDDCardNum [1]	Y	integer	Detail card number
TDDCreationDt		date	Detail creation date. Default: system date.
TDDFullPathName		varchar2(255)	Full pathname

Table A-27 TrkDocDetails Table *(Continued)*

Name	Req	Type (Len)	Description
TDDIOType		integer	I/O type
TDDXlatFlags		integer	Translation flags
TDDTrkId		integer	Tracking ID - internal tracking number for the submission unit.
TDDIntgId		integer	Interchange identifier
TDDGrpId		integer	Group identifier
TDDState		integer	Tracking state. Valid values: 0 = TSunknown 1 = TSready 2 = TSinProgress 3 = TSdoneOK 4 = TSdoneBad 5 = TSalldoneOK 6 = TSbundled
TDDErrnum		integer	Tracking error number. Default: 0.
TDDSndrMBName		varchar2(60)	Sender member's name
TDDRcvrMBName		varchar2(60)	Receiver member's name
TDDDocType		varchar2(60)	Document type
TDDSubmittedTRKId		integer	Tracking ID of the submitter
TDDObjPerm		integer	Object permission
TDDModByGroup		varchar2(60)	ID of group modified by
TDDModByUser		varchar2(60)	ID of user modified by
TDDModDt		date	Modification date

1. Primary key: *TDDId* + *TDDCardNum*
2. Foreign key: *TDDId* into *TrkDoc* (*TDId*)

MDNInfo

Table A-28 stores message disposition notification information used by the ECXpert System.

Table A-28 MDNInfo Table

Name	Req	Type (Len)	Description
MDNId [1]	Y	integer	Message disposition notification ID
MDNSndrMBName		varchar2(60)	Sender's member name
MDNRcvrMBName		varchar2(60)	Receiver's member name
MDNReceiveDt		varchar2(60)	Date received
MDNOrigMsgId		varchar2(128)	Original message ID
MDNOrigMsgDigest		varchar2(60)	Original message digest
MDNMicAlg		integer	(internal use) MDN digest algorithm
MDNObjPerm		integer	Object permission
MDNModByGroup		varchar2(60)	ID of group modified by
MDNModByUser		varchar2(60)	ID of user modified by
MDNModDt		date	Modification date. Default: system date.

1. Primary key

Oftp

Table A-29 stores OFTP EERP (end-to-end-response) reconciliation information used by the ECXpert System.

Table A-29 OFtp Table

Name	Req	Type (Len)	Description
OFFileName [1]	Y	varchar2(255)	The Virtual File Dataset Name. SFIDSN field in the SFID Start File OFTP command. Maximum length is 26 characters.
OFTimeStamp [1]		varchar2(16)	The Virtual File Time Stamp. The SFIDTIME field in the SFID Start File OFTP command. It is exactly 6 characters long, and has the format HHMMSS.
OFDateStamp [1]		varchar2(16)	The Virtual File Date Stamp. The SFIDDATE field in the SFID Start File OFTP command. Exactly 6 characters long, and has the format YYMMDD.
OFTrkId		number	The tracking ID assigned to the submitted file by the ECXpert system.
OFSndrMBName		varchar2(60)	The OFTP Sender ID of the file.
OFRcvrMBName		varchar2(60)	The OFTP Receiver ID of the file.
OFDocType		varchar2(60)	The Document Type of the submitted file.
OFEERPExpected		number	Count of the number of EERP's expected before this node can return an EERP to the originator. Who to return the EERP to is specified in the corresponding EERP relationship. This value is incremented whenever a file is sent outbound that is a descendant of the original file.
OFEERPReceived		number	Count of the number of EERP's received by this OFTP node for this particular OFTP file. Incremented when EERP's corrsponding to this unit of work are received by the ECXpert OFTP server.
OFEERPSchedType		number	Is the file for scheduled or immediate transmission? 0 = immediate 1 = scheduled

Table A-29 OFtp Table *(Continued)*

Name	Req	Type (Len)	Description
OFEERPState		number	Current state of the EERP entry: 0 = AWAITING_FINAL_EERP 1 = READY_TO_SEND 2 = SENT_OK 3 = FAILED_TO_SEND

1. Primary key

EventLog

Table A-30 stores a log of processing events, including error conditions.

Table A-30 Event Log Table

Name	Req	Type (Len)	Description
ELId [1]	Y	integer	Event log ID
ELEventId [2]		integer	Event ID
ELCategory		varchar2(60)	Functional area in which event took place (e.g. bundle, dispatcher, smtp, etc.)
ELSeverity		integer	Severity of the event: 0 = unknown 10 = informational 20 = warning 30 = error
ELEventShortMsg		varchar2(255)	Short message describing event
ELTrkId		integer	Tracking ID associated with event
ELIntgId		integer	Interchange ID associated with event if applicable
ELGrpId		integer	Group ID associated with event if applicable
ELDocId		integer	Document ID associated with event if applicable
ELTDId		varchar2(30)	Non-numeric document identifier. Combination of numeric integers according to following syntax: TrackingID-InterchangeID-GroupID-DocID

Table A-30 Event Log Table *(Continued)*

Name	Req	Type (Len)	Description
ELPercolate		integer	Flag to indicate whether severity of event log has been percolated to tracking tables
ELModByGroup		varchar2(60)	ID of last user to modify the database row. (not used)
ELModByUser		varchar2(60)	Functional area in which event took place (e.g. bundle, dispatcher, smtp, etc.)
ELModDt		date	Date the row was last modified Default: system date.

1. Primary key
2. Foreign key: *ELId* into *MsgFormats* (*MFId)*

MsgFormats

Table A-31 stores text strings describing processing events, including error conditions, that are entered into the EventLog table during processing.

Table A-31 MsgFormats Table

Name	Req	Type (Len)	Description
MFId [1]	Y	integer	Message format ID
MFCategory		varchar2(60)	Event category
MFSeverity		integer	Event severity
MFShortMsgFmt		varchar2(255)	Short message format
MFLongMsgFmt		long varchar	Long message format
MFObjPerm		integer	Object permission
MFModByGroup		varchar2(60)	ID of group modified by
MFModByUser		varchar2(60)	ID of user modified by
MFModDt		date	Modification date. Default: system date.

1. Primary key

Index

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X